シミュレーション概論ノート第3講：方程式の根について

1　全般的注意

連続関数 $f(x)$ が与えられた場合に $f(x)=0$ を満たす根 x を求めたい。解析的には代数方程式の４次以下のものしか一般解は存在しないが現実に解（それも実根）を求める必要はしばしば生じる。そこで数値計算では繰り返し計算を用いて根を評価する。ここでは２分法とニュートン法を紹介する。大域的な根の求め方もあることはあるがあまり有効ではないので省略しよう。

2　２分法

２分法は中間値の定理を用いて２端点を決めた場合の中間の点での平均値を決め、繰り返し端点の幅を小さくすることで $f(x)=0$ を解こうとする方法である。

まず中間値の定理の復習をすると「$f(a)<0, f(b)>0$ なる a,b が存在すると $f(\alpha)=0$ なる根 α $(a<\alpha<b)$ が少なくとも一つ存在する」というものである。（自明であるから証明は省略）。

２分法を数値的に用いる場合に（計算機の誤差の範囲内で）正しい結果を得るのは効率が悪い（誤差を考えなければ一般に無限回の操作が必要）。従って根を必要な桁だけ計算できた場合に計算を打ち切る操作が必要になる。そのために小さな数 ϵ を導入して、$|a-b|/2<\epsilon$ になるまで区間を縮小したら計算を打ち切るとしてプログラムを組むのがいい。そのときの誤差は高々 ϵ である。

アルゴリズムは以下の通り。

1. $f(x)$ を与える。ϵ を設定する。
2. $f(a)<0, f(b)>0$ を満たすように変数 a,b の値を設定する。
3. 以下

   $c:=(a+b)/2,$

   if $|a-b|/2<\epsilon$ goto step 4

   $fc:=f(c)$

   if $fc>0$ then $b:=c$

   if $fc<0$ then $a:=c$

   if $fc=0$ goto step 4

   を繰り返す
4. c を答とする。

例題として $f(x)=e^{-x}-x^2=0$ の解を $\epsilon=5\times10^{-6}$ と設定して解いてみる。上のアルゴリズムをそのまま Fortran に翻訳すると

```
eps=5.0e-6
```

```
  a=1.0
  b=0.0

  do i=1,30
    c=(a+b)/2.0
    if (abs(a-b)/2.0.lt.eps) goto 1

    fc=f(c)

    if (fc.eq.0) goto 1

    if (fc.gt.0.0) then
      b=c
    else
      a=c
    endif
    write(*,*) i,c
 enddo

 1       write(*,*) i,c

 end

  real function f(x)
    f=exp(-x)-x**2
    return
  end
```

となる。但し初期の a,b はそれぞれ 1,0 に設定してある。この問題では１８回の繰り返しで望みの精度が得られて $x = 0.7034645$ という根を得る。この答えは５桁まで正しい。

因みに C でプログラムを書くと次の様になる。

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

float f(float x);

main()
{
  int i;
  float a, b, c, fc, eps;
```

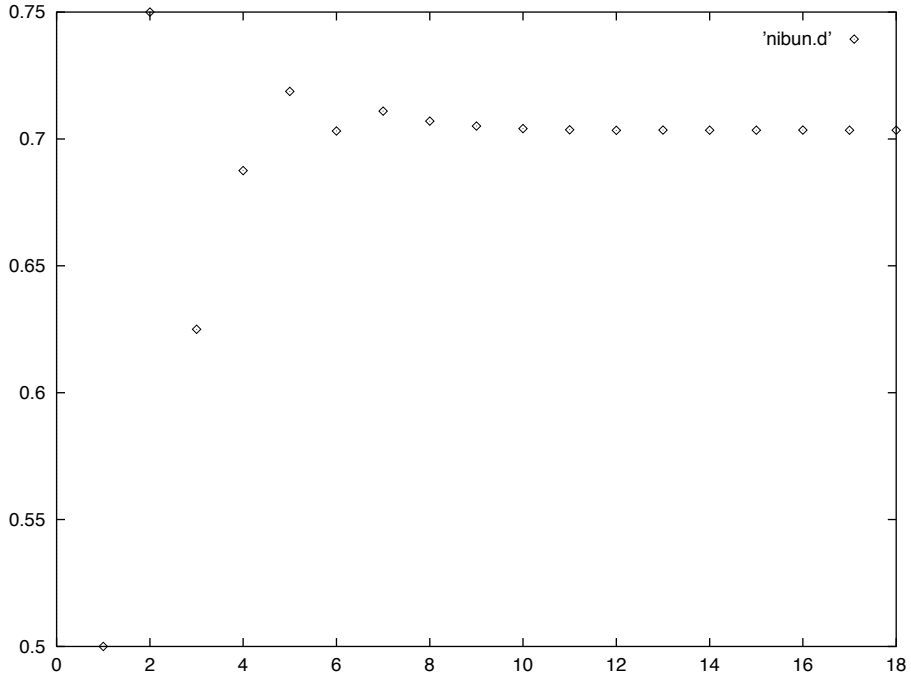

図 1: ２分法の収束の様子。

```
  eps = 5.0e-6;

  a = 1.0;
  b = 0.0;

  for(i = 1; i <= 30; i++) {
    c = (a + b) / 2.0;

    if(fabs(a - b) / 2.0 < eps) break;

    fc = f(c);

    if(fc == 0) break;

    if(fc > 0.0) b = c;
    else a = c;

    printf("%d %9.7f\n", i, c);
  }

  printf("%d %9.7f\n", i, c);

  return 0;
}
```

```
float f(float x)
{
  return exp(-x)-x*x;
}
```

対応は明らかであろう。

２分法の著しい特徴は安全確実に正しい解に収束していくことであり、同時に計算量を見積もる事が可能な点である。ループ計算で繰り返しを１回実行すると区間 (a, b) は半分になる。従って計算終了までに必要な $f(c)$ の計算回数は

$$\frac{|a-b|}{2^{N+1}} < \epsilon \tag{1}$$

をみたす最小の N である。即ち

$$N > \log_2(\frac{|a-b|}{\epsilon}) - 1 \tag{2}$$

となる。上の計算例では $\log_2(|a-b|/\epsilon) - 1 \simeq 16.609$ なので計算終了までに必要な $f(c)$ の計算回数は１７回である。実際に１８回目の計算では収束判定条件を満たして計算が終了しているので $f(c)$ は計算しておらず計算回数は見積り通りと言える。

3　ニュートン法

２分法を安全確実な方法とするとニュートン法は安定性に欠けるが高速な解法である。従ってMathematica などの数値計算ソフトやパッケージなどにも標準的な方法として採用されている。ニュートン法は接線を利用して解を近似的に求めている。まず $y = f(x)$ のグラフの根 $x = \alpha$ の付近の領域を考える。そして x の適当な初期値 x_0 から $(x_0, f(x_0))$ でグラフに接する接線を引く。この接線と x 軸の交点を x_1 とする。さらにグラフ上の点 $(x_1, f(x_1))$ で再び接線を引き、x 軸との交点を求めて以下その操作を収束するまで繰り返す。n 番目に求められた x 軸との交点を x_n とすると点 $(x_n, f(x_n))$ における接線の式は $y = f(x_n) + f'(x_n)(x - x_n)$ である。この式の $y = 0$ の時の x を求めればそれが x_{n+1} であるので $f'(x_n) \neq 0$ であれば

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{3}$$

となる。

一般に微分を実行することは難しくないのでこの漸化式は計算可能である。$n \to \infty$ で $x_n \to a$ に収束したとすると $a = a - f(a)/f'(a)$ であるので確かに $f(a) = 0$ を満たす。

ニュートン法では２分法とは異なって現在の x_n がどれだけ真の値に近いかは一般には分からない。そこで収束条件としてある小さな正の数 ϵ に対して

$$|\frac{x_{n+1} - x_n}{x_{n+1}}| < \epsilon \tag{4}$$

となった時点で計算を終了し x_{n+1} を答えとする。

アルゴリズムを以下に示す。

1. 初期値 x と ϵ の設定。
2. 以下

   new x:= x-f(x)/f'(x)

   if $|newx - x| < \epsilon * |newx|$ goto step 3

   x:=new x

   繰り返す
3. new x を答えとする。

ニュートン法を前と同じ問題に適用してみよう。ここでは $\epsilon = 10^{-5}$ として初期値を $x = 1$ とする。

```
external f,g
 eps=1.0e-5

x=1.0
 do i=1,10
     y=x-f(x)/g(x)
     if (abs(y-x).lt.eps*y) goto 1
     x=y
 enddo

1        write(*,*) i,x

  end

real function f(x)
  f=exp(-x)-x**2
end

real function g(x)
g=-exp(-x)-2.0*x
end
```

このプログラムの実行から、わずか 4 回の計算で望みの精度の結果が得られることはすぐわかるだろう。

また例によって C のプログラムを書くと

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

float f(float);
```

```
float g(float);

main()
{
  int i;
  float x, y, eps;

  eps = 1.0e-5;
  x =1.0;

  for(i = 1; i <= 10; i++) {
    y = x - f(x) / g(x);

    if(fabs(y-x) <= eps * y) break;

    x = y;
  }

  printf("%d %9.7f\n", i, x);

  return 0;
}

float f(float x)
{
  return exp(-x)-x*x;
}

float g(float x)
{
  return -exp(-x)-2.0*x;
}
```

となる。

ニュートン法の計算量がこのように少ない理由を考えてみよう。根を α として誤差を $\epsilon_n = x_n - \alpha$ とし よう。この時 (3) から

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

という関係式がある。充分、ϵ_n が小さいと Taylor 展開 $f(x_n) \simeq \epsilon_n f'(\alpha) + (\epsilon_n^2/2) f''(\alpha)$, $f(\alpha) = 0$ などを $f''(\alpha) \neq 0$ の仮定の下で用いて

$$\epsilon_{n+1} \simeq \epsilon_n - \frac{\epsilon_n f'(\alpha) + (\epsilon_n^2/2) f''(\alpha)}{f'(\alpha) + \epsilon_n f''(\alpha)} \simeq \epsilon_n^2 \frac{f''(\alpha)}{2f'(\alpha)} \tag{6}$$

となる。

従って $f''(\alpha)/f'(\alpha)$ が 1 のオーダーの数であるとすると x_n と α の誤差が 10^{-p} の場合、$|x_{n+1} - \alpha| = O(10^{-2p})$ である。

ニュートン法は安全とは言えない方法であるという点を既に触れていた。容易に想像できる通り、与えられた初期値によっては段々x_n が根から離れていくことが実際に起こり得る。勿論、正しい解の近傍から始めるとニュートン法でうまく根は求めることはできるので順次 x_n を書き出して収束の度合を確認することが必要である。また 2 分法と組み合わせて用いるのも実際的な方法である。

問 1：(1) $f(x) = \cos x - x^2 = 0$, (2) $g(x) = \exp[-x^2] - \sin x = 0$ の根のうち $0 < x < 1$ を満たすものを、2 分法とニュートン法を用いて 6 桁の精度で求めよ。

問 2： $f(x) = (x-2)^2(x-1) = 0$ の根をニュートン法で求めよう。初期値を $x_0 = 1$ 及び 3 として根を 6 桁の精度で求めよ。また $x = 2$ が重根である場合に (5) は使えない。その場合の ϵ_n の漸化式を求めよ。

4 補足

4.1 FUNCTION 文

まず講義では FUNCTION 文の倍精度実数の場合、REAL を「REAL*8 とすれば良い」と置き換えれば結構です。しかしメディアセンターの標準では倍精度の FUNCTION 文を用いる場合、REAL*8 のかわりに DOUBLE PRECISION を使った方が無難なようです。また注意すべきは、引数等を全て倍精度で揃えておいて下さい。そうしないととんでもない答えになるおそれがあります。とりあえずニュートン法のプログラムを倍精度にしたものは以下の様なものです。このうち external 文の行はコメントアウトしても大丈夫です。因みにこのプログラムは固定形式としてだけでなく自由形式としてもメディアセンターでコンパイル可能です。

```
      double precision eps,x,y
      external f,g
      double precision f,g

      eps=1.0d-5

      x=1.0d0
      do i=1,10
          y=x-f(x)/g(x)
          if (dabs(y-x).lt.eps*y) goto 1
          x=y
      enddo

1     write(*,*) i,x

       end

      double precision function f(x)
```

```
      double precision x
        f=dexp(-x)-x**2
      end

     double precision function g(x)
     double precision  x
      g=-dexp(-x)-2.0d0*x
      end
```

4.2 重根の場合のニュートン法

重根の場合でも通常のニュートン法は使用可能です。ですが次の様に誤差がなかなかゼロに収束しません。$x=\alpha$ で重根、即ち $f(\alpha)=f'(\alpha)=0$ を満たすとすると、n 回目の誤差 ϵ_n に対して $f(x_n)\simeq\frac{\epsilon_n^2}{2}f''(\alpha)+\frac{\epsilon_n^3}{6}f'''(\alpha)$ 及び $f'(x_n)\simeq\epsilon_n f''(\alpha)+\frac{\epsilon_n^2}{2}f'''(\alpha)$ が成り立つ。従って、ニュートン法の漸化式は

$$\epsilon_{n+1}-\epsilon_n=-\frac{f(x_n)}{f'(x_n)}\simeq\epsilon_n\frac{\frac{1}{2}f''(\alpha)}{f''(\alpha)}=\frac{\epsilon_n}{2}$$

となる。従って、重根の場合もニュートン法は使える。しかし収束速度は遅くなる。通常の場合、収束速度が ϵ_n^2 に比例していたことを思い出して欲しい（２次収束すると言う)。

実は３次収束するアルゴリズムは存在する。その一つに解と係数の関係を使ったデュラン・ケルナー法と呼ばれるものがある。しかし説明は省略。

5 Fortran90 自由形式の補足

二分法をもちいたプログラム (倍精度)：

```
real(8)::a,b,c,eps,fc
real(8),external::f
eps=1.0d-12

a=1.0_8
b=0.0_8

do i=1,50

   c=(a+b)/2.0_8
   if(abs(a-b)/2.0_8 < eps)exit
   fc=f(c)
   if(fc==0.0_8)exit
   if(fc > 0.0_8)then
      b=c
   else
```

```
      a=c
   endif
   print*,i,c

enddo

print*,i,c
end


real(8) function f(x)
real(8):: x
f=exp(-x)-x**2
end
```

ニュートン法をもちいたプログラム (倍精度)：

```
real(8)::eps,x,y
real(8),external::f,g
eps=1.0d-12
x=1.0_8

do i=1,50

   y=x-f(x)/g(x)
   if(abs(y-x)< y*eps)exit
   x=y
   print*,i,x

enddo
print*,i,x
end


real(8)function f(x)
real(8)::x
f=exp(-x)-x*x
end

real(8) function g(x)
real(8)::x
g=-exp(-x)-2.0_8*x
```

```
end
```

問題の解答例：プログラムは上にのせたものの関数と初期値 (最初の x の値) をかえただけなので省略します。上のプログラムを使い、eps=1.0d-12 としたときの収束回数と得られた根のみのせます。

問 1 (1) 二分法 40 回 0.8241323123029, ニュートン法 5 回 0.8241323123025,

(2) 二分法 40 回 0.6805981743791, ニュートン法 5 回 0.6805981743784,

問 2

(1) 初期値 x=0: 8 回 1.0000000000000, 初期値 x=3: 40 回 2.0000000000033,

(2) ノート三講の式 (6) より eps(n+1)　 eps(n)/2 となる。