代数方程式

ー数値計算(09/10/9)ー

関西学院大学理工学部 西谷滋人

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(bisection method) とニュートン法(Newton's 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．さらに収束判定条件について言及する．

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，Newton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Newton法が威力を発揮する．

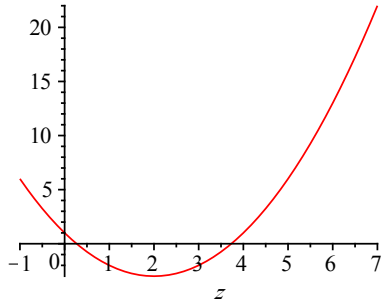
Maple での解

Mapleでは代数方程式の解は，fsolveで求まる．
x^2-4x+1=0
の解を考える．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．

```
> restart;
> func:=x->x^2-4*x+1;
```

$$func := x \to x^2 - 4x + 1$$

```
> plot(func(z),z=-1..7);
```



もし，解析解が容易に求まるなら，その結果を使うほうがよい．Maple scriptの解析解を求めるsolveでは，

```
> solve(func(x)=0,x);
```

$$2 + \sqrt{3}, 2 - \sqrt{3}$$

と即座に求めてくれる．数値解は以下の通り求められる．

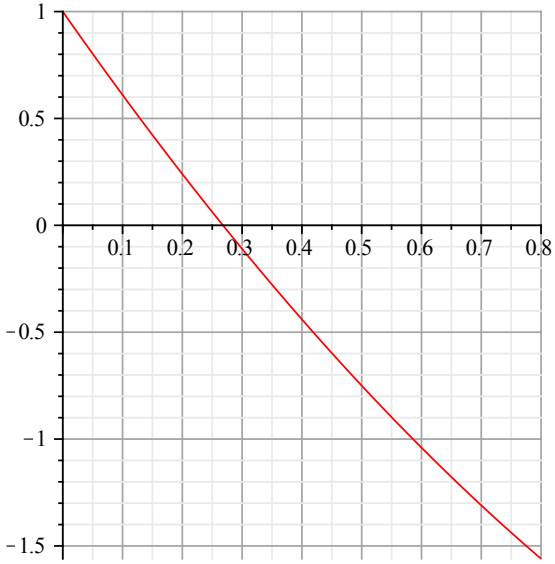
```
> fsolve(x^2-4*x+1=0,x);
```

0.2679491924, 3.732050808

二分法とNewton法の原理

二分法は領域の端x_1, x_2で関数値$f\left(x_1\right)$,$f\left(x_2\right)$を求め，中間の値を次々に計算して，解を囲い込んでいく方法である．

```
> plot(func(z),z=0..0.8,gridlines=true);
```



x_1	x_2	$f(x_1)$	$f(x_2)$
0.0	0.8		

Newton法(あるいはNewton-Raphson法)

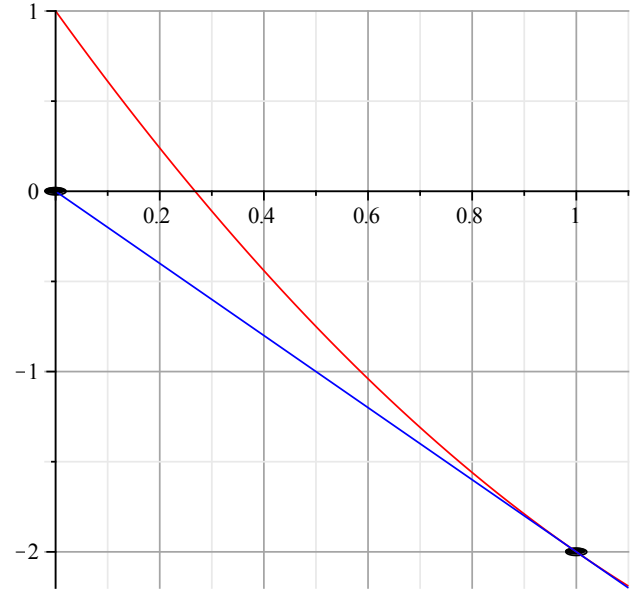
Newton法は最初の点x_1から接線をひき，それがy軸(=0)と交わった点を新たな点x_2とする．さらにそこでの接線を求めて...
という操作を繰り返しながら解を求める方法である．関数の微分を$df(x)$とすると，これらの間には

という関係が成り立つ．

```
> ?disk;
> with(plots):with(plottools):
  x1:=1.0:x0:=0.0:
  df:=unapply(diff(func(x),x),x);
  p:=plot(func(z),z=0..1.1):
  p1:=plot(df(x1)*(z-x1)+func(x1),z=0..1.1,color=blue):
  p2:=[disk([x1,func(x1)],0.02),
  disk([x0,0],0.02)]:
```

$$df := x \to 2x - 4 \qquad (4.1.1)$$

```
> display(p,p1,p2,gridlines=true);
```



x	$f(x)$	$df(x)$
1.0		

二分法とNewton法のコード

二分法(bisection)

```
> x1:=0:
  x2:=0.8:
  f1:=func(x1):
  f2:=func(x2):
  for i from 1 to 5 do
    x:=(x1+x2)/2;
    f:=func(x);
    if f*f1>=0.0 then
      x1:=x;
      f1:=f;
    else
      x2:=x;
      f2:=f;
    end if;
    printf("%20.15f, %20.15f\n",x,f);
  end do:
   0.400000000000000,   -0.440000000000000
   0.200000000000000,    0.240000000000000
   0.300000000000000,   -0.110000000000000
   0.250000000000000,    0.062500000000000
   0.275000000000000,   -0.024375000000000
```

Newton法(あるいはNewton-Raphson法)

```
> dfunc:=unapply(diff(func(z),z),z);
```

$$dfunc := z \rightarrow 2z - 4$$

```
> x:=1:
  f:=func(x):
  printf("%15.10f, %+24.25f\n",x,f);
  for i from 1 to 5 do
    x:=x-f/dfunc(x);
    f:=func(x);
    printf("%15.10f, %+24.25f\n",x,f);
  end do:
   1.0000000000, -2.0000000000000000000000000
   0.0000000000, +1.0000000000000000000000000
   0.2500000000, +0.0625000000000000000000000
   0.2678571429, +0.0003188775510000000000000
   0.2679491900, +0.0000000084726737970000000
   0.2679491924, +0.0000000000000000059821834
```

以下のようにDigitsを変更すれば，Mapleでは浮動小数点演算の有効数字を変えることができる．

```
> Digits:=40;
```

$$Digits := 40$$

収束性と安定性

実際のコードの出力からも分かる通り，解の収束の速さは2つの手法で極端に違う．2分法では一回の操作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．Newton法では関数・初期値が素直な場合($f'(x) <> 0$)に，収束が誤差の2乗に比例する2次収束を示す．以下はその導出を示した．

```
> restart;
  ff:=subs(xi-x[f]=ei,series(f(xi),xi=x[f],4));
```

$$ff := f(x_f) + D(f)(x_f)\,ei + \frac{1}{2}D^{(2)}(f)(x_f)\,ei^2 + \frac{1}{6}D^{(3)}(f)(x_f)\,ei^3 + O(ei^4)$$

```
> dff:=subs({0=x[f],x=ei},series(diff(f(x),x),x,3));
```

$$dff := D(f)(x_f) + D^{(2)}(f)(x_f)\,ei + \frac{1}{2}D^{(3)}(f)(x_f)\,ei^2 + O(ei^3)$$

```
> ei1:=ei-ff/dff;
```

$$ei1 := ei - \left(f(x_f) + D(f)(x_f)\,ei + \frac{1}{2}D^{(2)}(f)(x_f)\,ei^2 + \frac{1}{6}D^{(3)}(f)(x_f)\,ei^3 + O(ei^4)\right) \Big/ \left(D(f)(x_f) + D^{(2)}(f)(x_f)\,ei + \frac{1}{2}D^{(3)}(f)(x_f)\,ei^2 + O(ei^3)\right)$$

```
> ei2:=simplify(convert(ei1,polynom));
```

$$ei2 := -\frac{1}{3}\,\frac{-3D^{(2)}(f)(x_f)\,ei^2 - 2D^{(3)}(f)(x_f)\,ei^3 + 6f(x_f)}{2D(f)(x_f) + 2D^{(2)}(f)(x_f)\,ei + D^{(3)}(f)(x_f)\,ei^2}$$

```
> ei3:=series(ei2,ei,3);
```

$$ei3 := -\frac{f(x_f)}{D(f)(x_f)} + \frac{f(x_f)\,D^{(2)}(f)(x_f)}{D(f)(x_f)^2}\,ei - \frac{1}{6}\,\frac{1}{D(f)(x_f)}\left(-3D^{(2)}(f)(x_f) - \frac{3f(x_f)\,D^{(3)}(f)(x_f)}{D(f)(x_f)} + \frac{6f(x_f)\,D^{(2)}(f)(x_f)^2}{D(f)(x_f)^2}\right)ei^2 + O(ei^3)$$

```
> subs(f(x[f])=0,ei3);
```

$$\frac{1}{2}\,\frac{D^{(2)}(f)(x_f)}{D(f)(x_f)}\,ei^2 + O(ei^3)$$

注意すべきは，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．Newton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)．

二分法では，収束は遅いが，正負の関数値の間に連続関数では必ず解が存在するという意味で解が保証されている．しかし，Newton法では，収束は速いが，必ずしも素直に解に収束するとは限らない．解を確実に囲い込む，あるいは解に近い値を初期値に選ぶ手法が種々考案されている．解が安定であるかどうかは，問題，解法，初期値に大きく依存する．収束性と安定性のコントロールが数値計算のツボとなる．

収束判定条件

どこまで値が解に近づけば計算を打ち切るかを決める条件を収束判定条件と呼ぶ．以下のような条件がある．

ϵ(イプシロン，epsilon)法

δ(デルタ，delta)法

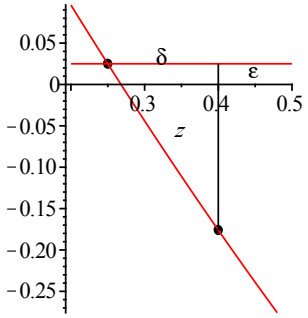
占部法

数値計算の際の丸め誤差までも含めて判定する条件で，

$\left|f\left(x_{i+1}\right)\right| > \left|f\left(x_i\right)\right|$

とする．

```
> with(plots):with(plottools):
> f2:=x->0.4*(x^2-4*x+1):x1:=
  0.25:x0:=0.4:
  p:=plot(f2(z),z=x1-x1/5..x0+
  x0/5):
  p1:=plot([f2(x1)],z=0.2.
  .0.5):
  p2:=[disk([x1,f2(x1)],
  0.005),
  disk([x0,f2(x0)],0.005)]:
  l1:=line([x0,f2(x0)],[x0,f2
  (x1)]):
  t1 := textplot([0.45,0.0,
  `epsilon`],align=above):
  t2 := textplot([0.325,0.05,
  `delta`],align=BELOW):
> display(p,p1,p2,l1,t1,t2);
```



2変数の関数の場合

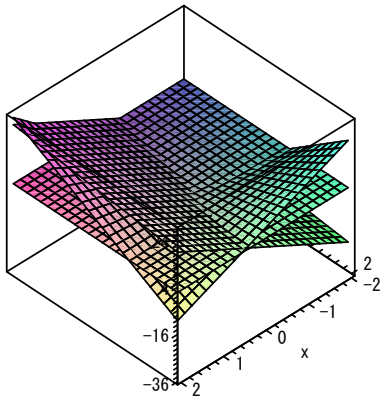
2変数の関数では，解を求める一般的な手法は無い．この様子は実際に2変数の関数で構成される面の様子をみれば納得されよう．解のある程度近くからは，Newton法で効率良く求められることが知られている．

```
> restart;
> f:=(x,y)->4*x+2*y-6*x*y;
  g:=(x,y)->10*x-2*y+1;
```

$$f := (x,y) \to 4x + 2y - 6xy$$

$$g := (x,y) \to 10x - 2y + 1$$

```
> plot3d({f(x,y),g(x,y),0},x=-2..2,y=-2..2);
```




```
> fsolve({f(x,y)=0,g(x,y)=0},{x,y});
```

$\{y = 0.1229854420, x = -0.07540291160\}$

課題

1 Newton法の$f\left(x\right)$, d$f\left(x\right)$の関係を示す式を導け．

2 次の関数

$f\left(x\right)$=exp(-x)-2exp(-2x);

の解を二分法，Newton法で求めよ．

3 収束条件がうまく機能しない例を示せ．

4 割線法は有効な微分が求まらないような場合に効率のよい，二分法を改良した方法である．二分法では新たな点を元の2点の中点に取っていた．そのかわりに下図に示すごとく，新たな点を元の2点を直線で内挿した点に取る．二分法のコードを少し換えて，割線法のコードを書け．また，収束の様子を二分法，Newton法と比べよ．

```
> func:=x->x^2-4*x+1:
  x1:=0:
  x2:=2:
  f1:=func(x1):
  f2:=func(x2):
  plot({(z-x1)*(f1-f2)/(x1-x2)+f1,func(z)},z=0..2);
```