C プログラミング

— 非線形方程式の解法 (2) —

早稲田大学

本日の目標

- 数値微分を用いたニュートン法
- 関数ポインタを用いた引数渡し

ニュートン法

ニュートン法のアルゴリズム

Step 1. 定義域に含まれる適当な初期値 $x^{(0)}$ を設定する．解の値に見当がつくのなら，近い値を設定した方が良い．

Step 2. 次の漸化式に基づいて数列 $x^{(1)}, x^{(2)}, x^{(3)}, \ldots$ を求めていく．

$$x^{(i+1)} = x^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})} \qquad (i = 0, 1, 2, \cdots)$$

但し，$f'(x^{(i)}) \neq 0$ と仮定する．

Step 3. 適当な収束条件を満たしたら，その時の $x^{(i+1)}$ を近似解とする．
例: $\left|x^{(i+1)} - x^{(i)}\right| < \varepsilon$

ニュートン法のプログラム

- 方程式 $f(x) = 0$ に対して，二つの関数

  `double func(double x)` → 関数 $f(x)$

  `double func_d(double x)` → 導関数 $f'(x)$

  を定義している.

$$\Downarrow$$

- 関数 $f(x)$ が複雑な場合，導関数 $f'(x)$ の計算が面倒になる.
- 導関数の記述を間違える可能性がある.

$$\Downarrow$$

- 数値微分を行うことにより，$f(x)$ から自動的に微分値 $f'(x)$ の近似を計算する.

数値微分

1 変数関数 $f(x)$ に対しての微分の定義は

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

である.
この代わりに，有限の小さい正数 h を用いて

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

により微分値の近似を計算する方法を数値微分という.

数値微分

- 数値微分を行う際の刻み幅 h の値を適当な微小値（例えば $h = 1e-4$）として定義することで，ニュートン法のプログラムは例えば，以下のように変更できる．

```
int main(void){
   …


   i=0;
   while (i<max){
      f  = func(x);
      df = func_d(x);
      x_n= x- f/df;
      …
   }
   …
   return 0;
}
```

⇒

```
int main(void){
   …
   h=1e-4;

   i=0;
   while (i<max){
      f  = func(x);
      df = (func(x+h)-func(x))/h;
      x_n= x- f/df;
      …
   }
   …
   return 0;
}
```

例題

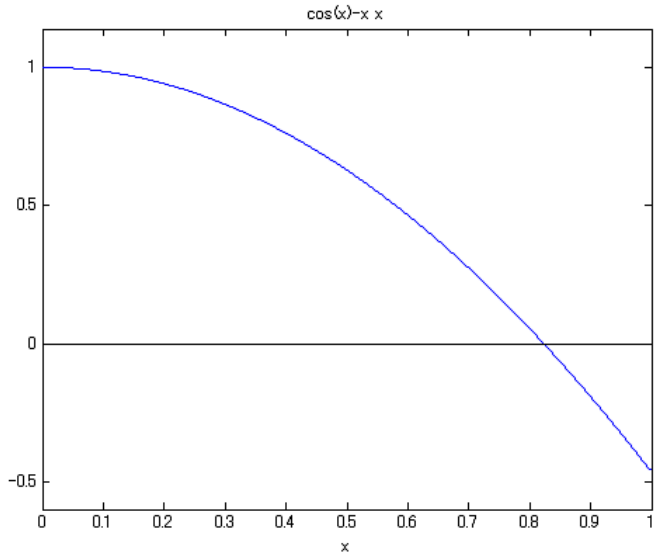
例題

方程式 $\cos(x) = x^2$ の解をニュートン法によって解くプログラムを
$f'(x) = -\sin(x) - 2x$ の計算箇所を数値微分を用いて書け.

- 初期値 $x^{(0)}$ としては 1.0 を取り，表記は以下のようにせよ.

```
initial value: 1
x(001) = ...
x(002) = ...
...
answer = ...
```

- 右の図は,
  $f(x) = \cos x - x^2$ のグラフである.



プログラム例

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

double func(double x){
  return cos(x)-x*x;
}
double Newton(double init, double eps, int imax){
  int i;
  double x_old=init, f, df, x_new,h=1e-04;
  for(i=0;i<imax;i++){
    x_new=x_old-func(x_old)/((func(x_old+h)-func(x_old))/h);
    printf("x(%03d)=%+.6f\n",i+1,x_new);
    if(fabs(x_new-x_old)<eps) return x_new;
    x_old = x_new;
  }
  return 0.0/0.0;/*Nan*/

}
int main(void){

  double error, x, x_n;
  int i, max;
  printf("initial value:");
  scanf("%lf",&x);
  x = Newton(x,1e-6,20);
  if(!isnan(x))/*Determining whether a value is NaN or not*/
    printf("answer =%+.6f\n",x);
  else
    printf("not converged\n");
  return 0;
}
```

プログラムについて

作成したプログラムは…

```
double func(double x){
    return cos(x) - x*x;
}

double  Newton(double x0, double e, int max){
   …
   i=0;
   while(i<max){
      f  = func(x);
      df = (func(x+h)-func(x))/h;
      …
   }
```

- 方程式を表す関数 func( ) をそのままニュートン法を行う関数 Newton( ) の中で利用している.

⇓

- 方程式を表す関数が別の名前（例えば，func2( )) で与えられたとき，関数 Newton( ) の中身も修正する必要がある.

ニュートン法のプログラム

作成したプログラムは…

```
double func2(double x){
    return x - cos(x);
}


double Newton(double x0, double e, int max){
   …
   i=0;
   while(i<max){
      f  = func2(x);
      df = (func2(x+h)-func2(x))/h;
      …
   }
```

⇓

関数へのポインタを利用することで．方程式を表す関数名を Newton( ) への引数として渡すことができる．

例：　Newton (func2,1.0,1e-6,100);

関数へのポインタ

- 変数や配列と同様，関数にもアドレスが与えられる.
- 配列名は，プログラム中では配列の先頭要素へのポインタに読み替えられていた.
- 同様に，関数名はプログラム中では関数へのポインタへと読み替えられる.
- 例えば，関数

```
double func2(double x){
    ...
}
```

が定義されているとき，関数名 func2 は関数 func2( ) の先頭アドレスを指す.

関数へのポインタ

- プログラム例

```
#include <stdio.h>

double add(double x, double y){
    return x+y;
}
double sub(double x, double y){
    return x-y;
}
int main(void){
    double (*func_p)(double,double);      /*func_p を２つの double 型の引数をとる*/
                                          /*関数へのポインタとして定義.     */
    func_p = add;                         /* func_p に関数 add へのポインタを代入*/
    printf("%f\n",func_p(3.0,2.0));

    func_p = sub;                         /* func_p に関数 sub へのポインタを代入*/
    printf("%f\n",func_p(3.0,2.0));

    return 0;
}
```

- 先ほどの例題を関数ポインタを用いて書き直してみよう.

本日のまとめ

- 数値微分を用いたニュートン法
- 関数ポインタを用いた引数渡し