C プログラミング

— 非線形方程式の解法 (1) —

早稲田大学

本日の目標

- **非線形方程式の解法**
  - 2 **分法**
  - **ニュートン法**

方程式の根

- 非線形方程式 $f(x) = 0$ の実数解 x を求める方法はどういうものがあるのだろうか？
  - $f(x)$ が４次以下の多項式であれば，解の公式が存在．
  - 一般の $f(x)$ に対しては，直接求める公式がない．

$\Downarrow$

  - 繰り返し計算によって解を求める方法（反復解法）がある．
- 今回の授業では，反復解法でも良く知られている
  - 二分法
  - ニュートン法

  を用いて方程式の解を求めるプログラムを作成しよう．

方程式の根

- 非線形方程式 $f(x) = 0$ の実数解 x を求める方法はどういうものがあるのだろうか？
  - $f(x)$ が４次以下の多項式であれば，解の公式が存在．
  - 一般の $f(x)$ に対しては，直接求める公式がない．

$$\Downarrow$$

  - 繰り返し計算によって解を求める方法（反復解法）がある．
- 今回の授業では，反復解法でも良く知られている
  - 二分法
  - ニュートン法

  を用いて方程式の解を求めるプログラムを作成しよう．

方程式の根

- 非線形方程式 $f(x) = 0$ の実数解 x を求める方法はどういうものがあるのだろうか？
  - $f(x)$ が４次以下の多項式であれば，解の公式が存在．
  - 一般の $f(x)$ に対しては，直接求める公式がない．

$$\Downarrow$$

  - 繰り返し計算によって解を求める方法（反復解法）がある．
- 今回の授業では，反復解法でも良く知られている
  - 二分法
  - ニュートン法

  を用いて方程式の解を求めるプログラムを作成しよう．

二分法

- $f(a)$ と $f(b)$ が異符号（$+$ と $-$ または $-$ と $+$）となるような $a,\ b$（$a < b$ とする）を初期値として与える必要あり．
- 関数 f は区間 $[a, b]$ において連続でなければならない．
- 区間 (a, b) における解を数値的に求める．解が複数ある場合は，初期値に依存してどれか一つが求まる．

二分法のアルゴリズム

Step 1. $f(a)$ と $f(b)$ が異符号であるような初期値 $a,\ b$ を設定する．

Step 2. $c \equiv (a+b)/2$ を計算する．収束条件 $|a-b| < 2\varepsilon$ が満たされれば，Step 3 に移る（但し，ε は小さい正の定数）．そうでない場合は，

- $f(c) = 0$ であれば，Step 3 に移る．
- $f(c)$ と $f(a)$ が同符号であれば，a に c の値を代入する．
- $f(c)$ と $f(b)$ が同符号であれば，b に c の値を代入する．

Step 2 を繰り返す．

Step 3. c を解とする．

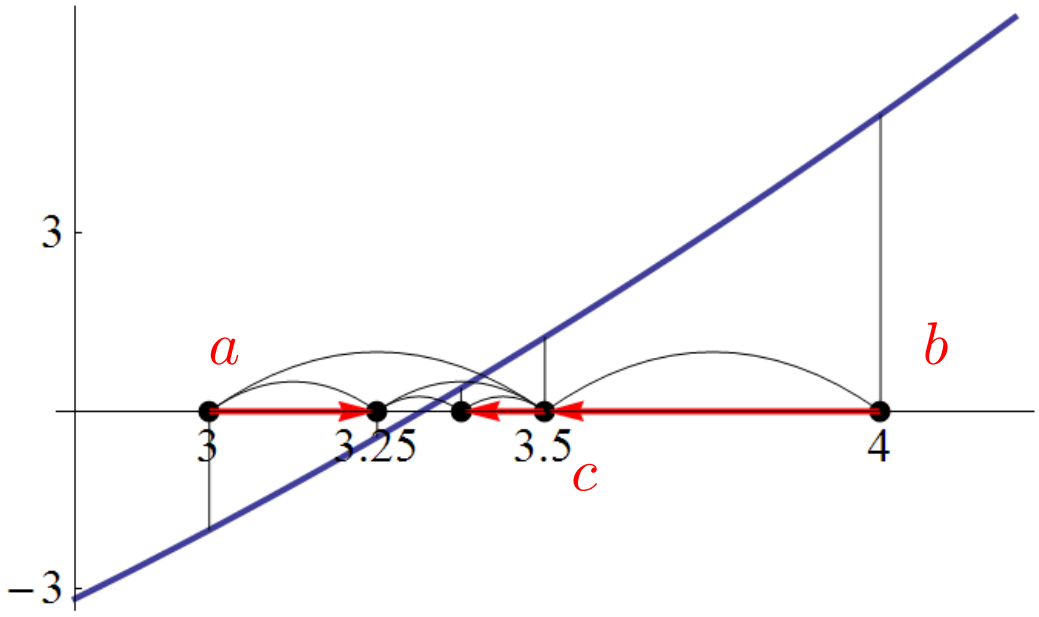
- 二分法による解 c は，真の解を x^* とすると，$|c - x^*| < \varepsilon$ を満たす．

二分法の原理

中間値の定理

連続な関数 $f(x)$ に対して，$f(a)$ と $f(b)$ が異符号となる $a,\ b$（$a < b$ とする）が存在すれば，$f(x) = 0$ となるような解は区間 (a, b) に少なくとも 1 つは存在する．

- **中間値の定理を繰り返し適用し，解の範囲を半分，半分と狭めていく．**



例題

方程式 $x = 1.2e^{-x}$ の解を二分法で求めよ．

- **$f(x) = x - 1.2e^{-x}$ として，関数 $f(x)$ を引数，戻り値の型を** double **で作成せよ．**

  double func(double x);

- **$a,\ b$ の初期値および ε を引数にとり，二分法の解を返す関数を作成せよ．**

  double Bisection(double a, double b, double eps);

- **初期値を $a = 0,\ b = 3$，$\varepsilon = 10^{-6}$ として呼び出せ．**

例題

- **途中経過が分かるよう，表示は次のようにせよ．書式指定は`"%+.6f"`として，正負の符号も付けよ．**

```
a = +0.000000, b = +3.000000, c = +1.500000
a = +0.000000, b = +1.500000, c = +0.750000
a = +0.000000, b = +0.750000, c = +0.375000
a = +0.375000, b = +0.750000, c = +0.562500
a = +0.562500, b = +0.750000, c = +0.656250
a = +0.562500, b = +0.656250, c = +0.609375
a = +0.609375, b = +0.656250, c = +0.632812
a = +0.632812, b = +0.656250, c = +0.644531
a = +0.632812, b = +0.644531, c = +0.638672
a = +0.632812, b = +0.638672, c = +0.635742
a = +0.632812, b = +0.635742, c = +0.634277
a = +0.634277, b = +0.635742, c = +0.635010
a = +0.635010, b = +0.635742, c = +0.635376
a = +0.635376, b = +0.635742, c = +0.635559
a = +0.635559, b = +0.635742, c = +0.635651
a = +0.635559, b = +0.635651, c = +0.635605
a = +0.635559, b = +0.635605, c = +0.635582
a = +0.635559, b = +0.635582, c = +0.635571
a = +0.635559, b = +0.635571, c = +0.635565
a = +0.635559, b = +0.635565, c = +0.635562
a = +0.635562, b = +0.635565, c = +0.635563
a = +0.635563, b = +0.635565, c = +0.635564
answer = +0.635564
```

例題 (ヒント)

- **アルゴリズムの，**Step 2 **を繰り返す，という箇所は無限ループを用いる．次のどちらの表記でもよい．**

```
while (1) { ... }

for (; ;) { ... }
```

- **収束条件** $|a-b| < 2\varepsilon$ **は以下のようにする．**

```
fabs(a - b) < 2 * eps
```

- **アルゴリズムの，**Step 3 **に移動，という箇所は** `break;` **を用いる．**
- $f(x) = x - 1.2e^{-x}$ **を以下のような関数として定義できる．**

```
double func(double x){
    double y;
    y=x-1.2*exp(-x);
    return y;
}
```

- 10^{-6} **は** 1e-6 **（指数表現）と書く．**

例題 (ヒント)

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

double Function1(double x) {
  計算したい方程式
}

double Bisection(double a, /*下段続く*/
          double b, double eps) {
  double c, fa, fb, fc;

  while (1) {
  無限ループを利用して計算をまわす
  }

  return c;
}

int main(void) {
  double x;

  二分法を行う

  return 0;
}
```

- **方程式**

```
double Function1(double x) {
  全ページヒント参照
}
```

- **二分法**

```
double Bisection(double a, double b, double eps) {
  double c, fa, fb, fc;
  while (1) {  /* 無限ループ */
    c = ;/*c の値は a, b の中点*/
    printf("");/*途中経過の表示*/
    if (判定条件) {
      break;
    }
    fa = Function1(a);
    fb = Function1(b);
    fc = Function1(c);
    if (fc == 0) {
      break;
    }
    else if ((fc > 0 && fa > 0) || (fc < 0 && fa < 0)) {
      a = c;
    }
    else if ((fc > 0 && fb > 0) || (fc < 0 && fb < 0)) {
      b = c;
    }
  }
  return c;
}
```

- **main 関数**

```
int main(void) {
  double x;
  x = Bisection();/*値は？*/
  printf("");/*答えを出力*/
  return 0;
}
```

ニュートン法

- 非線形方程式 $f(x) = 0$ を数値的に解くための反復解法の 1 つ．
- 二分法より高速に解に収束することが多いが，いつも収束するわけではない．
- $f(x)$ は勿論のこと，1 次導関数 $f'(x)$ の計算も必要である．

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 $x^{(i+1)}$ を解とする．
例: $\left| x^{(i+1)} - x^{(i)} \right| < \varepsilon.$

幾何学的解釈

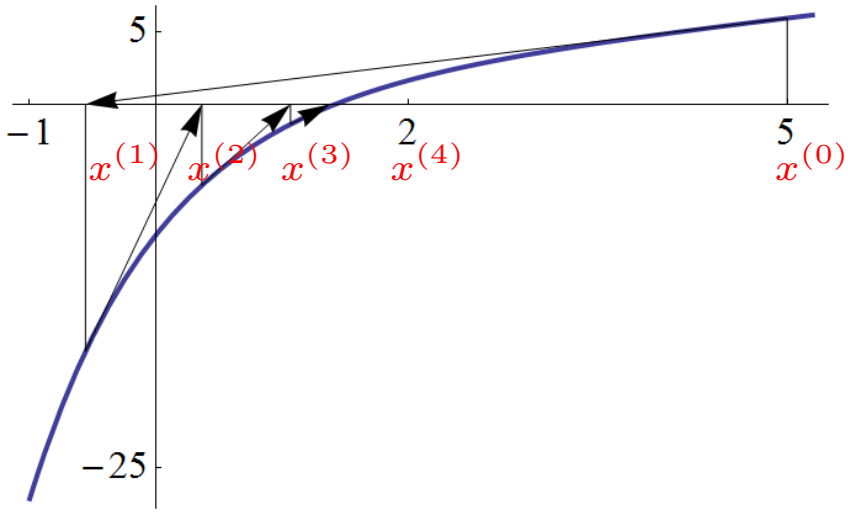
- $x = x^{(i)}$ における $f(x)$ の接線は

$$y = f'(x^{(i)})(x - x^{(i)}) + f(x^{(i)}).$$

- この接線と x 軸 $(y = 0)$ との交点の x 座標

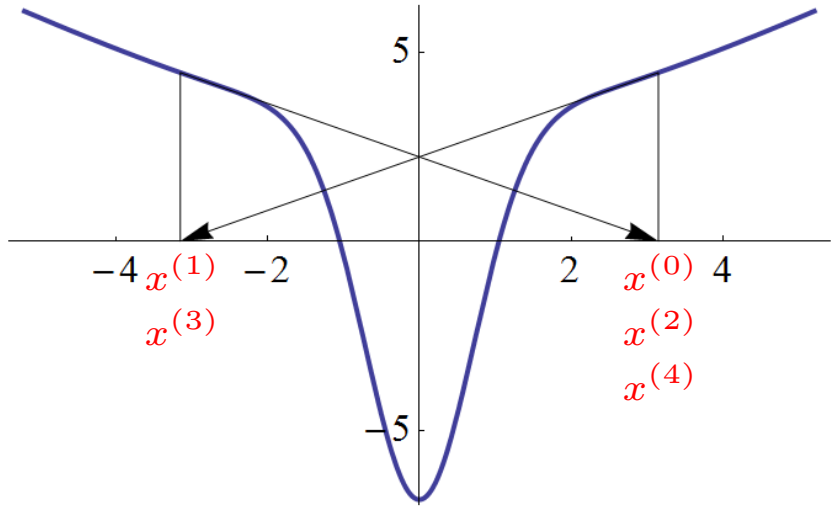
$$x = x^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})}$$

を求める操作を繰り返している．



反復回数制限

- ニュートン法では，解が存在しても，初期値に依存して解に収束しないことがある．そのため，反復回数の上限を予め設定しておき，上限に達した場合は強制的に反復を終了させることにする．



反復回数制限

最大反復回数を I_{MAX} とし，収束条件に該当することなく反復回数がこれに達した場合は，近似解を得ることができなかったと判断する．
例: $I_{\mathrm{MAX}} = 20$ などとする．

プログラムの作成

次の方程式の近似解を求めるプログラム

方程式

$$x = 1.2e^{-x}$$

の解をニュートン法により求めよ．

$$x = 1.2e^{-x} \Longleftrightarrow x - 1.2e^{-x} = 0$$

より， $f(x) = x - 1.2e^{-x}$ **とすると，**

$$f'(x) = 1 + 1.2e^{-x}$$

であり，

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

で定まる反復により解を求める．

プログラム作成のヒント

- $f(x), f'(x)$ の計算：$f(x), f'(x)$ はそれぞれ，

$$f(x) = x - 1.2e^{-x}, \qquad f'(x) = 1 + 1.2e^{-x}$$

  で与えられている．これらの値の計算には，$f(x), f'(x)$ をそれぞれ関数として，定義することにしよう．

$f(x) = x - 1.2e^{-x}$ を以下のような関数として定義できた．

```
double func(double x){
    double y;
    y=x-1.2*exp(-x);
    return y;
}
```

$f'(x)$ も同様に関数名 `func_d( )` などとして定義してみよう．

ニュートン法：プログラム作成のヒント

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

- ニュートン法では，適当な初期値 $x^{(0)}$ を選び，上の式に従って i の値を順次変化させながら次々と次の点を計算する．
- 適当な停止条件を満たしたところで計算を止め，そのときの値を解とする．
- 初期値の設定：現在いる点を表す変数 x を double 型で宣言し，x に適当な値を代入する．
- ニュートン法の反復：while 文, for 文等の繰り返し処理を利用して反復を実現する．

プログラム作成のヒント

反復の停止条件

- ニュートン法の反復を停止する条件として次を利用することにする．

$$|x^{(i+1)} - x^{(i)}| < \varepsilon$$

- eps として予め微小な値（例えば，`eps=1e-6`：10^{-6} の指数表現）を定義しておく必要がある．

最大反復回数の設定

- ニュートン法では，初期点の選び方により解に収束しない場合がある．そのため，最大の反復回数を設定しておき，解に収束しなかった場合は強制的に終了させる．
- 例えば，変数 max に最大の反復回数（例えば 20）を代入しておき，繰り返し文の終了条件として利用する．繰り返し文が終了したとき，i の値が max に等しい場合は解に収束しておらず，それ以外の時は解に収束している

本日のまとめ

- **非線形方程式の解法**
  - 2 **分法**
  - **ニュートン法**