# これまでの復習 (前期末試験に向けて)

山本昌志*

2006 年 9 月 19 日

## 1　前期末試験の傾向と対策

前期末試験の内容は，非線型方程式の近似解と常微分方程式の数値解法について，出題する．具体的には，次の数値計算法の計算原理とコンピュータープログラムの方法を理解しておくこと．

- 非線形方程式の数値計算法
  - 2 分法
  - ニュートン法
    - 実数解
    - 複素数解は，試験範囲外とする．
    - 連立非線形方程式は，試験範囲外とする．
- 常微分方程式の数値計算法
  - オイラー法
  - 2 次のルンゲ・クッタ法
  - 4 次のルンゲ・クッタ法は，試験範囲外とする．
  - 高階の常微分方程式は，試験範囲外とする．

以降，学習すべき内容をまとめておくので，よく理解して試験に臨むこと．試験日時と注意事項は，次の通りである．

| | |
|---|---|
| 試験日 | 9 月 28 日 (木曜日) 11:05〜12:05(60 分) |
| 場所 | 教室 (PC ルームではない) |
| 注意事項 | 教科書やノート，プリント類は持ち込み不可 |

*国立秋田工業高等専門学校　電気工学科

# 2　非線型方程式

## 2.1　概要

非線形方程式[1]

$$f(x) = 0 \tag{1}$$

の解の値が必要になることが，工学の問題でしばしばある．工学の問題では，数学でやったような厳密な解の必要はなく，精度の良い近似解で良いことが多い．近似解といっても，$10^{-10}$ 程度の精度のことを言っており，この程度の近似解が必要となる．

この非線形方程式は，図 1 のように $y = f(x)$ の x 軸と交わる点に実数解を持つ．ここだけとは限らないが，少なくともこの交わる点は解である．この点の値は，コンピューターを用いた反復 (ループ) 計算により探すことができる．

この授業では 4 通りの計算テクニックを学習したが，重要[2]なのは

1. 2 分法
2. ニュートン-ラフソン法 (ニュートン法)

である．

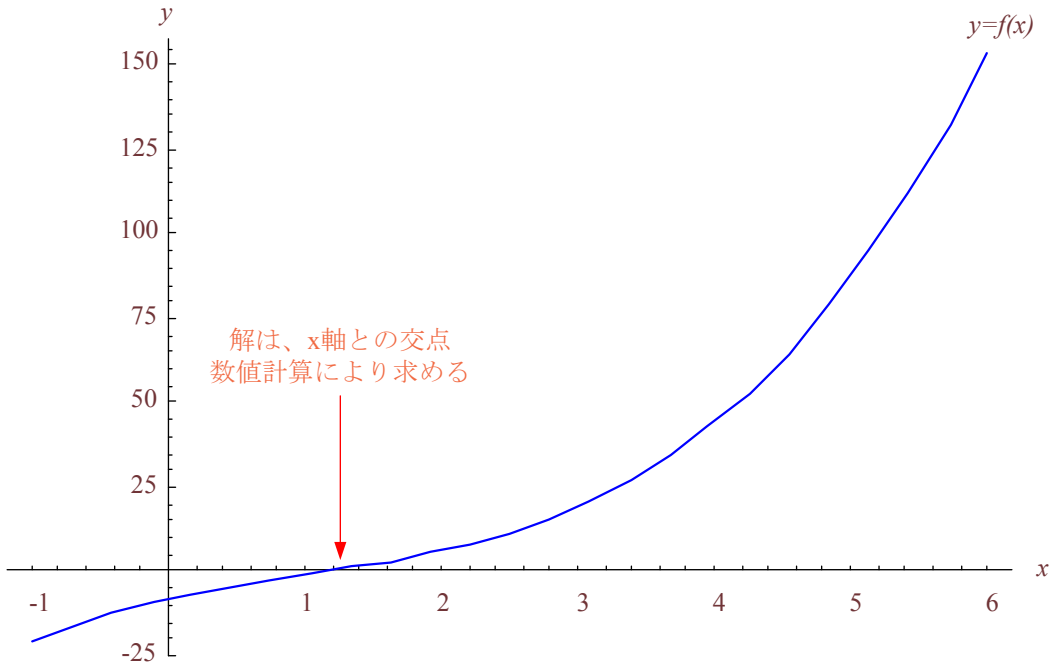


図 1: $f(x) = x^3 - 3x^2 + 9x - 8$ の関数．x 軸との交点が解である．

[1]方程式の右辺がゼロでない場合は，左辺へ移項して式 (1) の形にできる．
[2]諸君の前期末テストにおいて

## 2.2 二分法 (bisection method)

### 2.2.1 計算方法

閉区間 $[a, b]$ で連続な関数 $f(x)$ の値が，

$$f(a)f(b) < 0 \tag{2}$$

ならば，$f(\alpha) = 0$ となる $\alpha$ が区間 $[a, b]$ にある．

実際の数値計算は，$f(a)f(b) < 0$ であるような 2 点 $a, b(a < b)$ から出発する．そして，区間 $[a, b]$ を 2 分する点 $c = (a + b)/2$ に対して，$f(c)$ を計算を行う．$f(c)f(a) < 0$ ならば $b$ を $c$ と置き換え，$f(c)f(a) > 0$ ならば $a$ を $c$ と置き換える．絶えず，区間 $[a, b]$ の間に解があるようにするのである．この操作を繰り返して，区間の幅 $|b - a|$ が与えられた値 $\varepsilon$ よりも小さくなったならば，計算を終了する．解へ収束は収束率 $1/2$ の一次収束である．

実際にこの方法で

$$x^3 - 3x^2 + 9x - 8 = 0 \tag{3}$$

を計算した結果を図 2 に示す．この図より，$f(a)$ と $f(b)$ の関係の式 (2) を満たす区間 $[a, b]$ が $1/2$ ずつ縮小していく様子がわかる．この方法の長所と短所は，以下の通りである．

**長所** 閉区間 $[a, b]$ に解があれば，必ず解に収束する．間違いなく解を探すので，ロバスト (robust：強靭な) な解法と言われている．次に示すニュートン法とは異なり，連続であればどんな形の関数でも解に収束するので信頼性が高い方法と言える．さらに，解の精度も分かり便利である．解の誤差は，区間の幅 $|b - a|$ 以下でる．

**短所** 収束が遅い (図 6)．一次収束である．

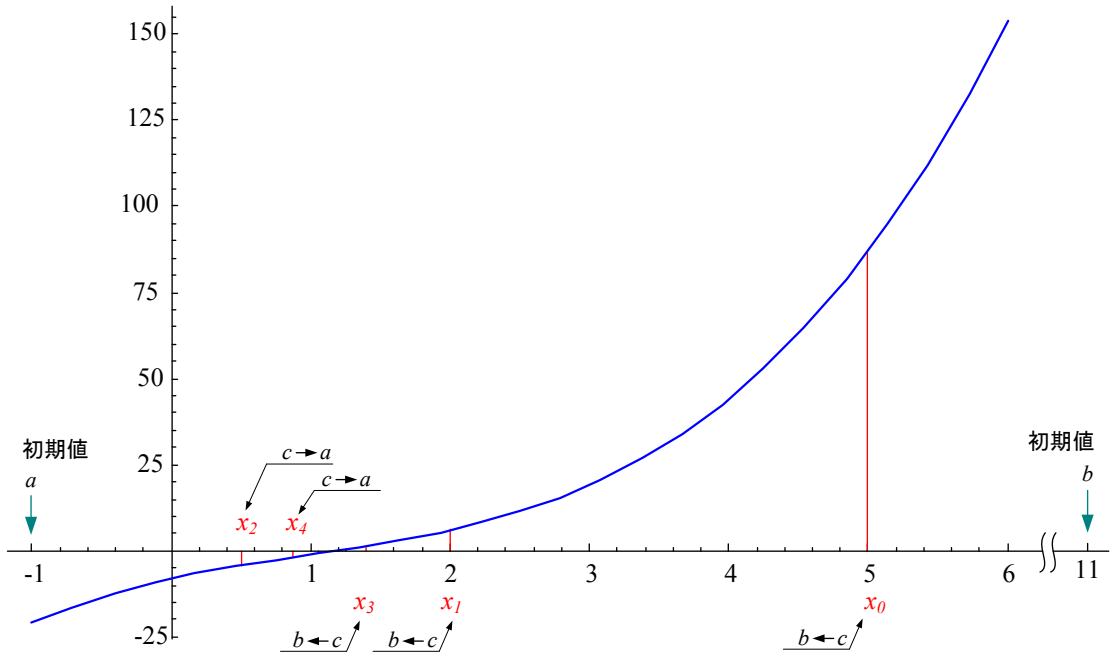


図 2: $f(x) = x^3 - 3x^2 + 9x - 8$ の実数解を 2 分法で解散し，その解の収束の様子を示している．初期値は $a = -1, b = 11$ として，最初の解 $c = x_0 = 5$ が求まり，順次より精度の良い $x_1, x_2, x_3, \cdots$ が求まる．それが，解析解 $x = 1.1659\cdots$ (x 軸との交点) に収束していく様子が分かる．

### 2.2.2 アルゴリズム

関数はあらかじめ，プログラム中に書くものとする．更に，計算を打ち切る条件もプログラム中に書くものとする．そうすると，図 3 のような 2 分法のフローチャートが考えられる．

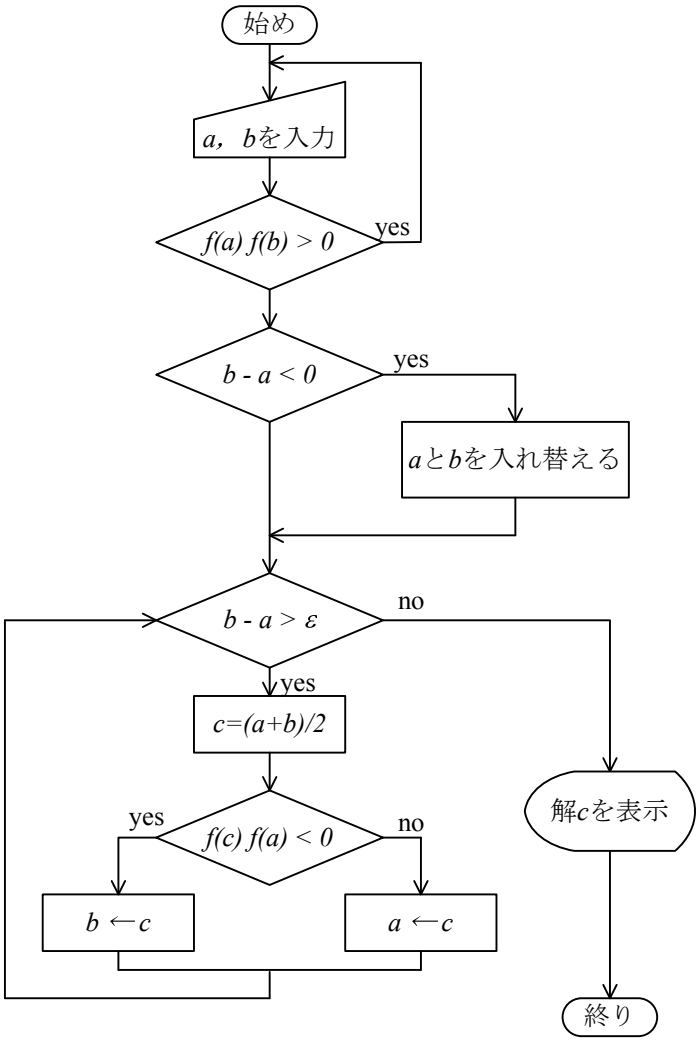


図 3: 2 分法のフローチャート

### 2.2.3 プログラム

このプログラムを暗記する必要はない．テストでは，アルゴリズム上，重要な部分を虫食いにして出題するつもりである (たぶん)．

リスト 1: 二分法で非線形方程式の近似解を求めるプログラム

```
1 #include <stdio.h>
2 double func(double x);
3
4 /*=============================================================*/
5 /*          main function                                      */
6 /*=============================================================*/
```

```
7
8  int main(){
9    double eps=1e-15;          /* precision of calculation */
10   double a, b, c;
11   double test;
12   char temp;
13   int i=0;
14
15   do{
16
17     printf("\ninitial value a = ");
18     scanf("%lf%c", &a, &temp);
19
20     printf("initial value b = ");
21     scanf("%lf%c", &b, &temp);
22
23     test=func(a)*func(b);
24
25     if(test >= 0){
26       printf("   bad initial value !!  f(a)*f(b)>0\n\n");
27     }
28
29   }while(test >= 0);
30
31   if(b-a<0){
32     c=a;
33     a=b;
34     b=c;
35   }
36
37
38   while(b-a>eps){
39
40     c=(a+b)/2;
41
42     if(func(c)*func(a)<0){
43       b=c;
44     }else{
45       a=c;
46     }
47
48     i++;
49     printf(" %d\t%20.15f\n",i,c);
50
51   }
52
53   printf("\nsolution x = %20.15f\n\n",c);
54
55   return(0);
56 }
57
58
59 /*==========================================================*/
60 /*        define function                                   */
61 /*==========================================================*/
62
63 double func(double x){
64   double y;
65
66   y=x*x*x-3*x*x+9*x-8;
67
68   return(y);
```

```
69 }
```

## 2.3 実数解のニュートン法 (Newton's method)

### 2.3.1 計算方法

関数 $f(x)$ のゼロ点 $\alpha$ に近い近似値 $x_0$ から出発する．そして，関数 $f(x)$ 上の点 $(x_0, f(x_0))$ での接線が，x 軸と交わる点を次の近似解 $x_1$ とする．そして，次の接線が x 軸と交わる点を次の近似解 $x_2$ とする．同じことを繰り返して $x_3, x_4, \cdots$ を求める (図 4)．この計算結果の数列 $(x_0, x_2, x_3, x_4, \cdots)$ は初期値 $x_0$ が適当であれば，真の解 $\alpha$ に収束する．

まずは，この数列の漸化式を求める．関数 $f(x)$ 上の点 $(x_i, f(x_i))$ の接線を引き，それと x 軸と交点 $x_{i+1}$ である．まずは，$x_{i+1}$ を求めることにする．点 $(x_i, f(x_i))$ を通り，傾きが $f'(x_i)$ の直線の方程式は，

$$y - f(x_i) = f'(x_i)(x - x_i) \tag{4}$$

である．$y = 0$ の時の $x$ の値が $x_{i+1}$ にである．$x_{i+1}$ は，

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{5}$$

となる．$x_i$ から $x_{i+1}$ 計算できる．これをニュートン法の漸化式と言う．この漸化式を用いれば，次々と近似解を求めることができる．

計算の終了は，

$$\left| \frac{x_{i+1} - x_i}{x_i} \right| < \varepsilon \tag{6}$$

の条件を満たした場合とするのが一般的である．$\varepsilon$ は計算精度を決める定数で，非常に小さな値である．これ以外にも計算の終了を決めることは可能である．必要に応じて，決めればよい．実際に式 (3) を計算した結果を図 4 に示す．接線との交点が解に近づく様子がわかるであろう．

ニュートン法を使う上で必要な式は，式 (5) のみである．計算に必要な式は分かったが，数列がどのように真の解 $\alpha$ に収束するか考える．$x_{i+1}$ と真値 $\alpha$ の差の絶対値，ようするに誤差を計算する．$f(\alpha) = 0$ を忘れないで，テイラー展開を用いて，計算を進めると

$$\begin{aligned} |\alpha - x_{i+1}| &= \left| \alpha - x_i + \frac{f(x_i)}{f'(x_i)} \right| \\ &= \left| \alpha - x_i + \frac{f(\alpha)}{f'(\alpha)} + \left[ 1 - \frac{f(\alpha)f''(\alpha)}{f'^2(\alpha)} \right] (x_i - \alpha) + O\left((\alpha - x_i)^2\right) \right| \\ &= \left| O\left((\alpha - x_i)^2\right) \right| \end{aligned} \tag{7}$$

となる．$i+1$ 番目の近似値は，$i$ 番目に比べて 2 乗で精度が良くなるのである．これを，二次収束と言い，非常に早く解に収束する．例えば，$10^{-3}$ の精度で近似解が得られているとすると，もう一回式 (5) を計算するだけで，$10^{-6}$ 程度の精度で近似解が得られるということである．一次収束の 2 分法よりも，収束が早いことが分かる．

ニュートン法の特徴をまとめると次のようになる．

**長所** 初期値が適当ならば，収束が非常に早い (図 6).

**短所** 初期値が悪いと，収束しない (図 7)．収束しない場合があるので，反復回数の上限を決めておく必要がある．

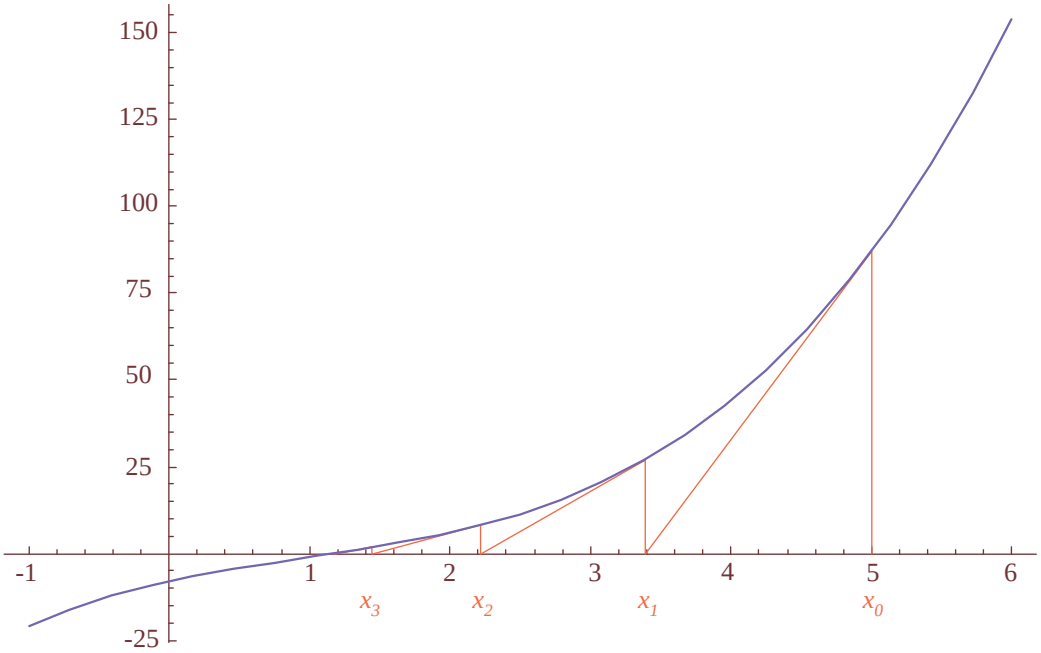


図 4: $f(x) = x^3 - 3x^2 + 9x - 8$ の実数解をニュートン法で計算し，解の収束の様子を示している．初期値 $x_0 = 5$ から始まり，接線と x 軸の交点からより精度の高い回を求めている．

### 2.3.2 アルゴリズム

2 分法同様，関数と計算を打ち切る条件はプログラム中に書くものとする．そうすると，図 5 のようなニュートン法のフローチャートが考えられる．

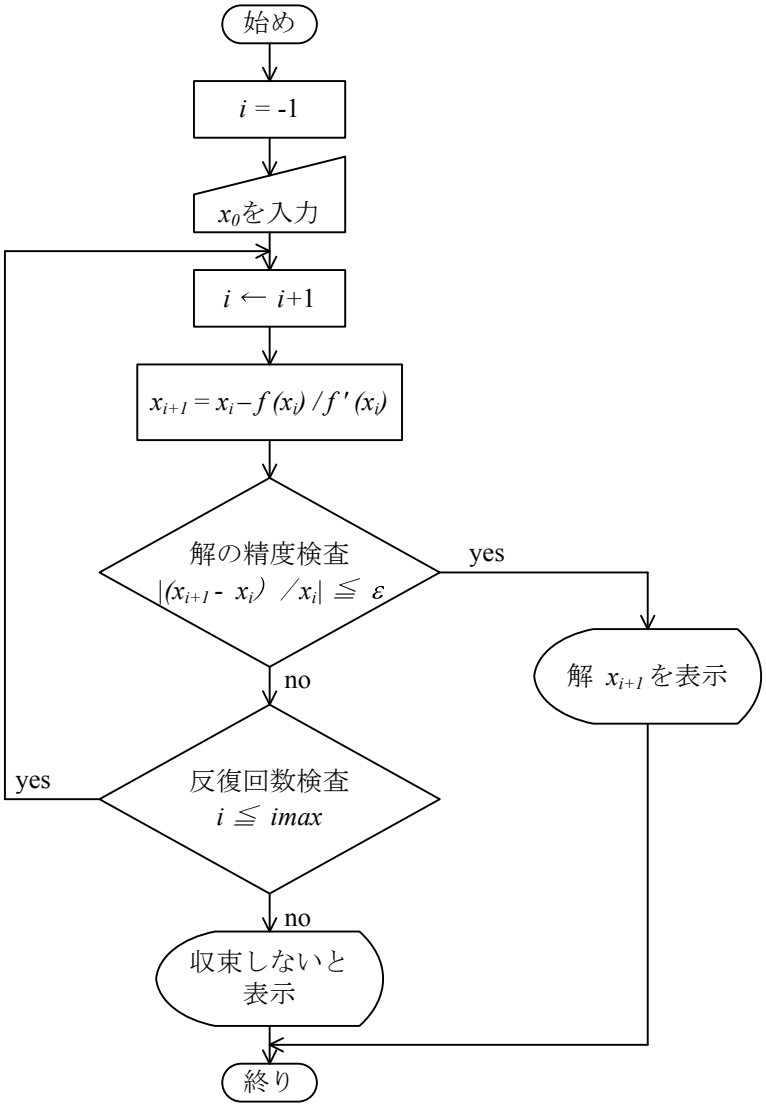


図 5: ニュートン法のフローチャート

### 2.3.3 プログラム

このプログラムを暗記する必要はない．テストでは，アルゴリズム上，重要な部分を虫食いにして出題するつもりである (たぶん)．

リスト 2: ニュートン法で非線形方程式の近似解を求めるプログラム

```
1 #include <stdio.h>
2 #include <math.h>
3 #define IMAX 50
```

```
4  double func(double x);
5  double dfunc(double x);
6
7  /*==========================================================*/
8  /*        main function                                     */
9  /*==========================================================*/
10 int main(){
11   double eps=1e-15;          /* precision of calculation */
12   double x[IMAX+10];
13   char temp;
14   int i=-1;
15
16   printf("\ninitial value x0 = ");
17   scanf("%lf%c", &x[0], &temp);
18
19   do{
20     i++;
21     x[i+1]=x[i]-func(x[i])/dfunc(x[i]);
22
23     printf("  %d\t%e\n", i, x[i+1]);
24
25     if(fabs((x[i+1]-x[i])/x[i])<eps) break;
26   }while(i<=IMAX);
27
28   if(i>=IMAX){
29     printf("\n  not converged !!! \n\n");
30   }else{
31     printf("\niteration = %d  solution  x = %20.15f\n\n",i,x[i+1]);
32   }
33
34   return(0);
35 }
36
37 /*==========================================================*/
38 /*        define function                                   */
39 /*==========================================================*/
40 double func(double x){
41   double y;
42
43   y=x*x*x-3*x*x+9*x-8;
44
45   return(y);
46 }
47
48 /*==========================================================*/
49 /*        define derived function                           */
50 /*==========================================================*/
51 double dfunc(double x){
52   double dydx;
53
54   dydx=3*x*x-6*x+9;
55
56   return(dydx);
57 }
```

## 2.4　ニュートン法と 2 分法の比較

### 2.4.1　解への収束速度

図 6 に，二分法とニュートン法の解への近づき具合を示す．二分法に比べ，ニュートン法が解への収束が早いことがわかる．前者は二次収束で，後者は一次収束であることがグラフより分かる．二分法は，10 回の計算で，$2^{-10} = 1/1024$ 程度になっている．

二分法に比べて，ニュートン法は収束が早く良さそうであるが，次に示すように解へ収束しない場合があり問題を含んでいる．

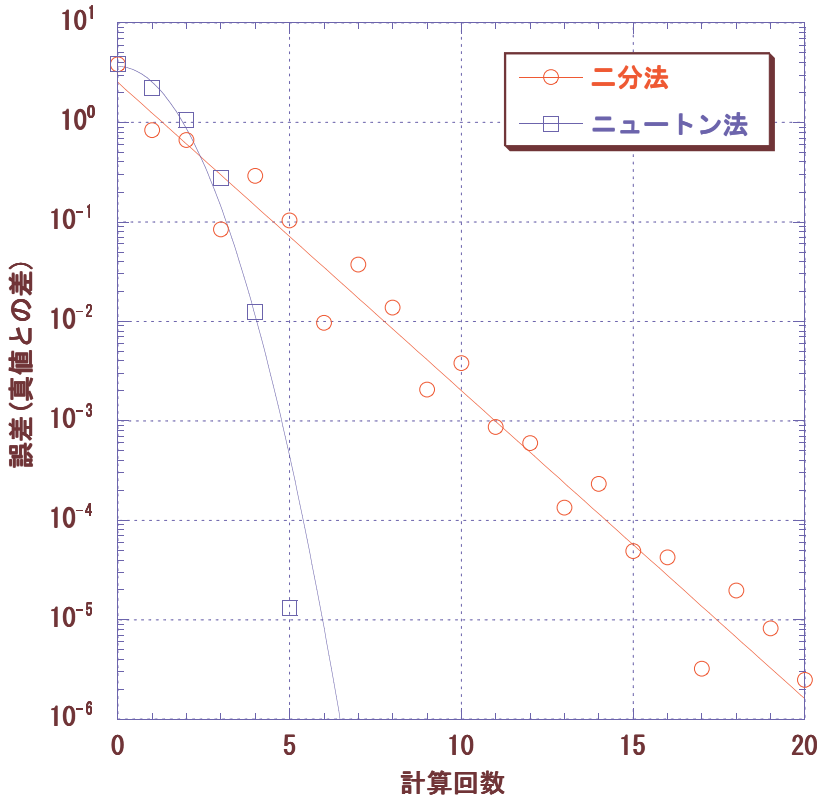


図 6: 二分法とニュートン法の計算回数 (反復回数) と誤差の関係

## 2.5　ニュートン法の問題点

アルゴリズムから，2 分法は解に必ず収束する．ただし，この方法は，収束のスピードが遅く，それが欠点となっている．一方，ニュートン法は解に収束するとは限らない．初期条件に依存する場合がある．厳密にその条件を求めるのは大変なので，初期条件により収束しない実例を示す．

非線形方程式

$$3\tan^{-1}(x-1)+\frac{x}{4}=0 \tag{8}$$

の解を計算することを考える．これは，初期値のより，収束しない場合がある．例えば初期値 $x_0=3$ の場合，図 7 のように収束しない．これを初期値 $x_0=2.5$ にすると図 8 のように収束する．

このようにニュートン法は解に収束しないで，振動する場合がある．こうなると，プログラムは無限ループに入り，永遠に計算し続ける．これは資源の無駄遣いなので，慎むべきである．通常は，反復回数の上限を決めて，それを防ぐ．ニュートン法を使う場合は，この反復回数の上限は必須である．

実際には収束しない場合のほうが稀であるので，ニュートン法は非常に強力な非線型方程式の解法である．ただ，反復回数の上限を決めることを忘れてはならない．

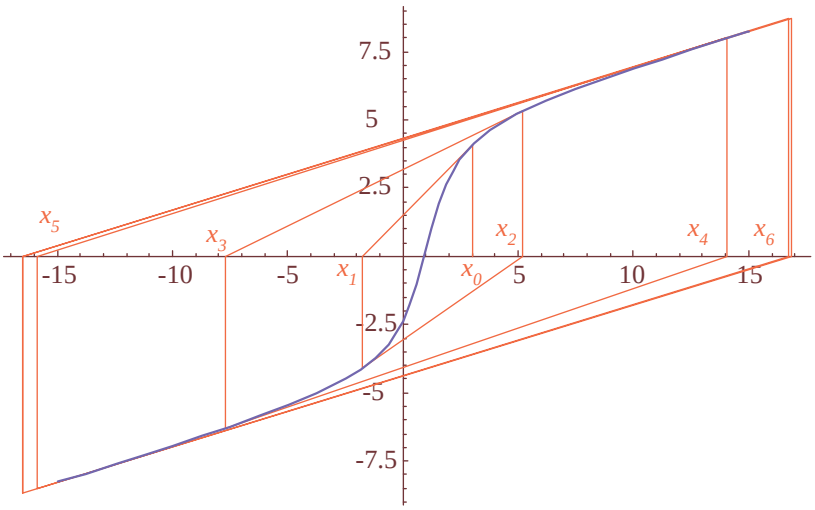


図 7: ニュートン法で解が求まらない場合

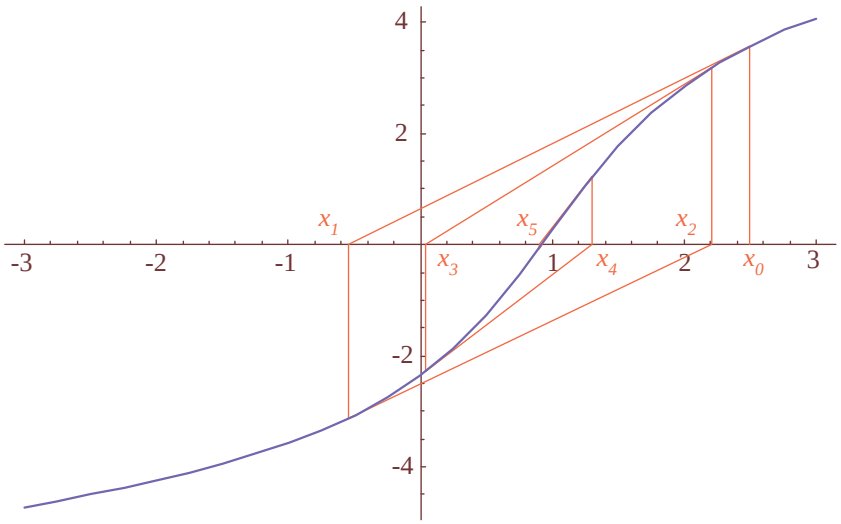


図 8: ニュートン法で解が求まる場合

# 3　常微分方程式の数値計算法

数値計算により，近似解を求める微分方程式は，

$$\frac{dy}{dx} = f(x,y) \qquad \text{初期条件} \quad y(a) = b \tag{9}$$

である．これは問題として与えられ，この式に従う $x$ と $y$ の関係を計算する．これを計算するためには，テイラー展開が重要な役割を果たす．テイラー展開を計算してから，オイラー法，2 次のルンゲ・クッタ法を説明する．

## 3.1　テイラー展開

数値計算は言うに及ばず科学技術全般の考察にテイラー展開 (Taylor expansion) は，重要な役割を果たす．電気の諸問題を考察する場合，いたるところにテイラー展開は顔を出すので，十分理解しなくてはならない．まずは，$x = a$ の回りのテイラー展開は，

$$\begin{aligned} f(x) &= f(a) + f'(a)(x-a) + \frac{1}{2!}f''(a)(x-a)^2 + \frac{1}{3!}f'''(a)(x-a)^3 + \frac{1}{4!}f^{(4)}(a)(x-a)^4 + \cdots \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(a)(x-a)^n \end{aligned} \tag{10}$$

と書ける．0 の階乗は $0! = 1$ となることに注意．$f^{(n)}(a)$ は，関数 $f(x)$ を $n$ 回微分したときの $x = a$ の値である．テイラー展開は，任意の関数 $f(x)$ は，無限のべき級数に展開できると言っている．その係数が，

$\frac{1}{n!}f^{(n)}(a)$ である．次に，$a=0$ としてみる．この場合，

$$\begin{aligned} f(x) &= f(0) + f'(0)x + \frac{1}{2!}f''(0)x^2 + \frac{1}{3!}f'''(0)x^3 + \frac{1}{4!}f^{(4)}(0)x^4 + \cdots \\ &= \sum_{n=0}^{\infty} \frac{1}{n!}f^{(n)}(0)x^n \end{aligned} \tag{11}$$

となる．これがマクローリン展開 (Maclaurin expansion) である．次に $x-a=\Delta x$ とする．そして，$a=x_0$ とすると

$$\begin{aligned} f(x_0+\Delta x) &= f(x_0) + f'(x_0)\Delta x + \frac{1}{2!}f''(x_0)\Delta x^2 + \frac{1}{3!}f'''(x_0)\Delta x^3 + \frac{1}{4!}f^{(4)}(x_0)\Delta x^4 + \cdots \\ &= \sum_{n=0}^{\infty} \frac{1}{n!}f^{(n)}(x_0)\Delta x^n \end{aligned} \tag{12}$$

となる．これは，しばしばお目にかかるパターン．

通常，我々がテイラー展開を使う場合，この式 (10)〜式 (12) のいずれかである．

## 3.2 オイラー法

### 3.2.1 計算原理

常微分方程式の解を $y=y(x)$ とする．その $x_i$ のまわりのテイラー展開は，

$$y_{i+1} = y(x_i+\Delta x) = y(x_i) + \frac{dy}{dx}\Big|_{x=x_i}\Delta x + \frac{1}{2}\frac{d^2y}{dx^2}\Big|_{x=x_i}\Delta x^2 + \frac{1}{6}\frac{d^3y}{dx^3}\Big|_{x=x_i}\Delta x^3 + \ldots \tag{13}$$

である．この式の右辺第 2 項は，式 (9) から計算できる．したがって，テイラー展開は，次のように書き表わせる．

$$y_{i+1} = y_i + f(x_i, y_i)\Delta x + O(\Delta x^2) \tag{14}$$

オイラー法での数値計算では，計算の刻み幅 $\Delta x$ は十分に小さいとして，

$$y_{i+1} = y_i + f(x_i, y_i)\Delta x \tag{15}$$

を計算する．この場合，計算の精度は 1 次と言う[3]．

オイラー法では，次の漸化式に従い数値計算を進める．解である $(x_0,y_0),(x_1,y_1),(x_2,y_2),\cdots$ が同じ手続きで計算できる．実際にプログラムを行うときは，*for* や *while* を用いて繰り返し計算を行い，結果の $x_i$ と $y_i$ は，配列 `x[i]` や `y[i]` に格納するのが常套手段である．

$$\begin{cases} x_0 = a \\ y_0 = b \\ x_{i+1} = x_i + \Delta x \\ y_{i+1} = y_i + f(x_i, y_i)\Delta x \end{cases} \tag{16}$$

この方法の計算のイメージは，図 9 の通りである．明らかに，出発点の導関数のみ利用しているために精度が悪いことが理解できる．式も対称でないため，逆から計算すると元に戻らない．

[3]誤差項が $O(\Delta x^{n+1})$ のとき，方法は $n$ 次の精度という慣わしです

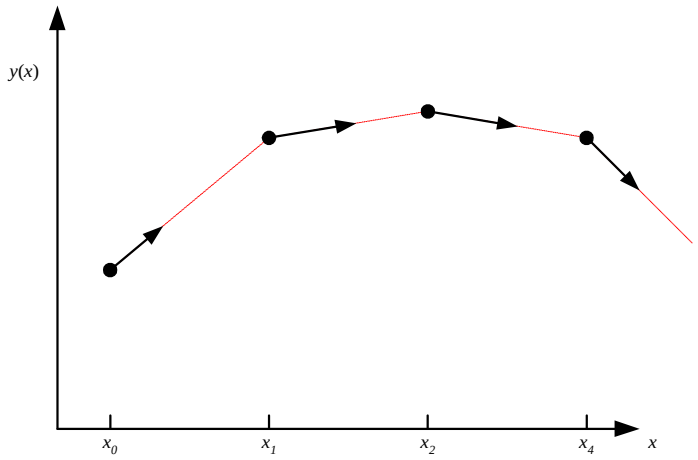


図 9: オイラー法．ある区間での $y$ の変化 $\Delta y$ は，計算の始めの点の傾きに区間の幅 $\Delta x$ を乗じて，求めている．

### 3.2.2 プログラム例

次の微分方程式

$$\frac{\mathrm{d}y}{\mathrm{d}x} = x^2 \sin x + \sqrt{y}\cos y \qquad\qquad y(0) = 0 \tag{17}$$

の近似解をオイラー法で計算するプログラムをリスト 3 に示す．計算結果は，ファイルに格納している．計算領域は $0 \leqq x \leqq 2$，計算のステップ幅は $\mathrm{d}x = 0.001$ となっている．

リスト 3: オイラー法で微分方程式の近似解を求めるプログラム

```
1  #include <stdio.h>
2  #include <math.h>
3  #define NSTEPS 2000                  // 計算ステップ数
4  #define NDIM 30000                   // 用意する配列の大きさ
5  #define X_MAX 2.0                    // 計算する最大 x
6
7  double f(double x, double y);        // プロトタイプ宣言
8
9  //=============================================================
10 // main 関数
11 //=============================================================
12 int main(void){
13   double x[NDIM], y[NDIM];           // 計算結果を入れる
14   double dx;
15   int i;
16   FILE *out;                         // 計算結果を保存するファイル
17
18
19   dx = X_MAX/NSTEPS;                 // 計算のきざみ幅 dxの計算
20   x[0] = 0;
21   y[0] = 0;
22
23
```

```
24    //―――――― オイラー法の計算 ――――――
25
26    for(i=0; i<NSTEPS; i++){
27      x[i+1] = x[0]+(i+1)*dx;          //x[i+1]=x[i]+dxよりも精度が良い
28      y[i+1] = y[i]+f(x[i],y[i])*dx;
29    }
30
31
32    //―――――― 計算結果をファイルに保存 ――――――
33
34    out = fopen("result.txt","w");
35    for(i=0; i<=NSTEPS; i++){
36      fprintf(out,"%20.15f\t%20.15f\n", x[i],y[i]);
37    }
38
39
40    fclose(out);
41
42    return 0;
43  }
44
45  //=============================================================
46  // dy/dx = f(x,y) の  f(x,y)を計算する関数
47  //=============================================================
48  double f(double x, double y){
49    return x*x*sin(x)+sqrt(y)*cos(y);
50  }
```

## 3.3 2次のルンゲクッタ

2次のルンゲ・クッタと呼ばれる方法は，いろいろある．ここでは，ホイン法と中点法をしめす．

### 3.3.1 ホイン法

先に示したように，オイラー法の精度は1次であるが，2次のルンゲ・クッタ法では2次となる．今まで刻み幅を $\Delta x$ と記述していたが，簡単のため $h$ と表現する．

2次の精度ということは，テイラー展開より

$$y(x_0+h) = y(x_0) + y'(x_0)h + \frac{1}{2}y''(x_0)h^2 + O(h^3) \tag{18}$$

となっていることを意味する．即ち，計算アルゴリズムが，

$$\Delta y = y'(x_0)h + \frac{1}{2}y''(x_0)h^2 + O(h^3) \tag{19}$$

となっている．

式 (19) から分かるように，$y$ の増分 $\Delta y$ を計算するためには，1階微分と2階微分の2項を満たす式が必要である．そうすると少なくとも，2点の値が必要となる．2点として，計算区間の両端の導関数の値を使う．この導関数は問題として与えられているので，計算は簡単である．そうして，区間の増分を $\alpha, \beta$ をパラメーターとした和で表すことにする．即ち，以下の通りである．

$$\Delta y = h\{\alpha y'(x_0) + \beta y'(x_0+h)\} \tag{20}$$

この $\alpha, \beta$ を上手に選ぶことにより，式 (19) と同一にできる．この式を $x_0$ の回りでテイラー展開すると

$$\Delta y = (\alpha + \beta) y'(x_0) h + \beta y''(x_0) h^2 + O(h^3) \tag{21}$$

となる．これを，式 (19) と比較すると，

$$\begin{cases} \alpha = \dfrac{1}{2} \\ \beta = \dfrac{1}{2} \end{cases} \tag{22}$$

とすれば良いことが分かる．これで，必要な式は求まった．まとめると，式 (9) を数値計算で近似解を求めるには次式を使うことになる．

$$\begin{cases} k_1 = h f(x_n, y_n) \\ k_2 = h f(x_n + h, y_n + k_1) \\ y_{n+1} = y_n + \dfrac{1}{2}(k_1 + k_2) \end{cases} \tag{23}$$

この式は，図 10 のようになる．オイラー法の図 9 との比較でも，精度が良いことが分かる．

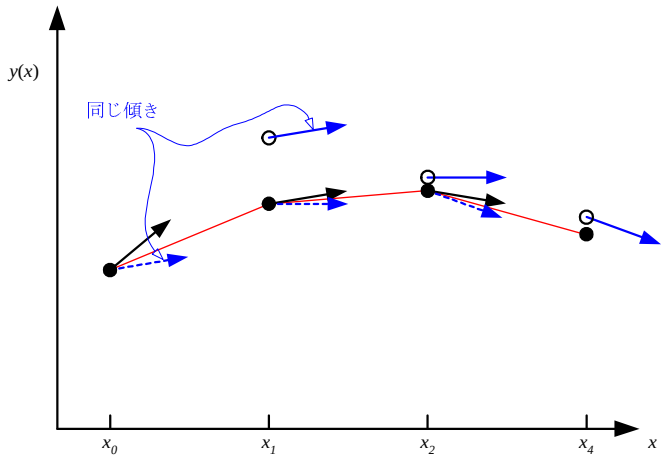


図 10: ホイン法 (教科書の方法)．ある区間での $y$ の変化 $\Delta y$ は，計算の始めと終わりの点付近の平均傾きに区間の幅 $\Delta x$ を乗じて，求めている．

### 3.3.2 中点法

これも，ホイン法と同じ 2 次の精度である．ホイン法は区間の両端の点の導関数を使ったが，中点法は出発点と中点で漸化式を作る．先ほど同様，2 点を使うので，2 次の精度にすることができる．ホイン法の式 (20) に対応するものは，

$$\Delta y = h\{\alpha y'(x_0) + \beta y'(x_0 + \frac{h}{2})\} \tag{24}$$

である．これを $x_0$ の回りでテイラー展開すると，

$$\Delta y = (\alpha + \beta) y'(x_0) h + \frac{\beta}{2} y''(x_0) h^2 + O(h^3) \tag{25}$$

となる．これを，式 (19) と比較すると，

$$\begin{cases} \alpha = 0 \\ \beta = 1 \end{cases} \tag{26}$$

となる必要がある．したがって，中点法の漸化式は，

$$\begin{cases} k_1 = h f(x_n, y_n) \\ k_2 = h f(x_n + \dfrac{h}{2}, y_n + \dfrac{k_1}{2}) \\ y_{n+1} = y_n + k_2 \end{cases} \tag{27}$$

となる．この公式のイメージを，図 11 に示しておく．

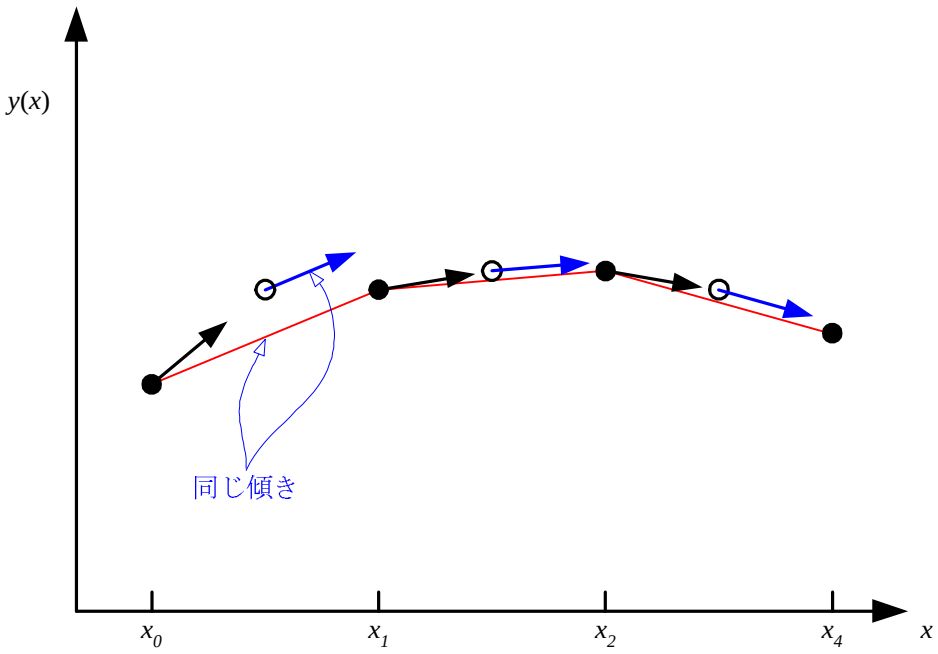


図 11: 中点法．ある区間での $y$ の変化 $\Delta y$ は，中点付近の傾きに区間の幅 $\Delta x$ を乗じて，求めている．