これまでの復習 (前期末試験に向けて)

山本昌志*

2007 年 7 月 25 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．

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	8 月 3 日 (金曜日) 9:55〜10:55(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

2 非線型方程式

2.1 概要

非線形方程式[1]

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

の近似解 x を求める—ことが，ここでの問題である．

*国立秋田工業高等専門学校　電気工学科

[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．

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．ここの講義では，次の 2 通りの方法で近似解を計算した．

1. 2 分法
2. ニュートン-ラフソン法 (ニュートン法)

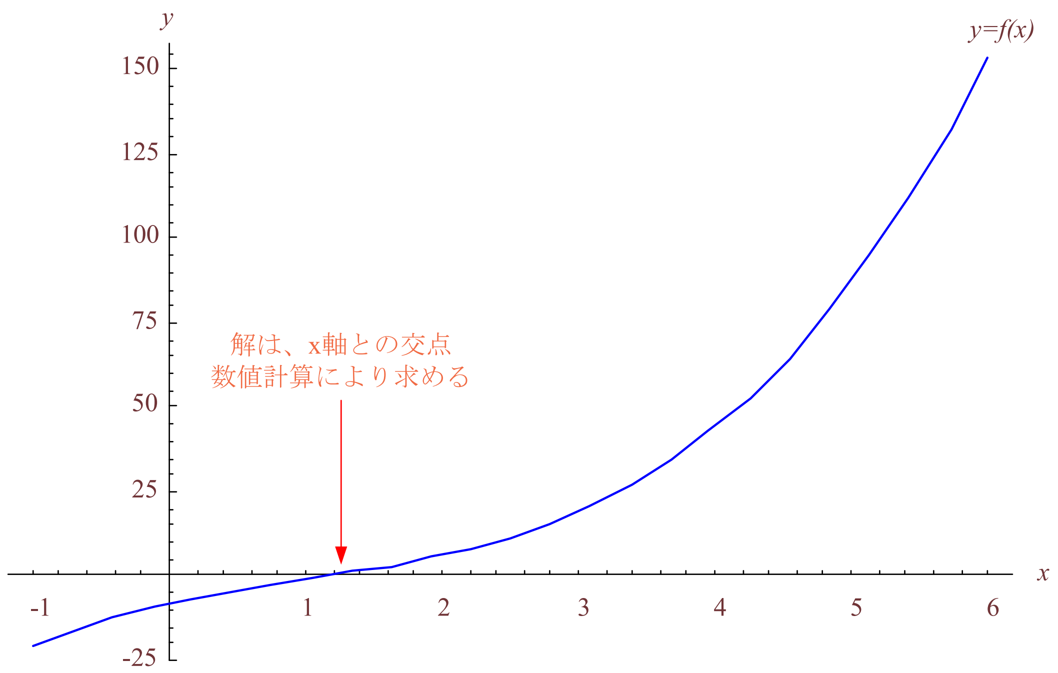

図 1: $f(x) = x^3 - 3x^2 + 9x - 8$ の関数．x 軸との交点が解である．

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 α 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 ε 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 $|b - a|$ 以下である．

短所 収束が遅い (図 6)．一次収束である．

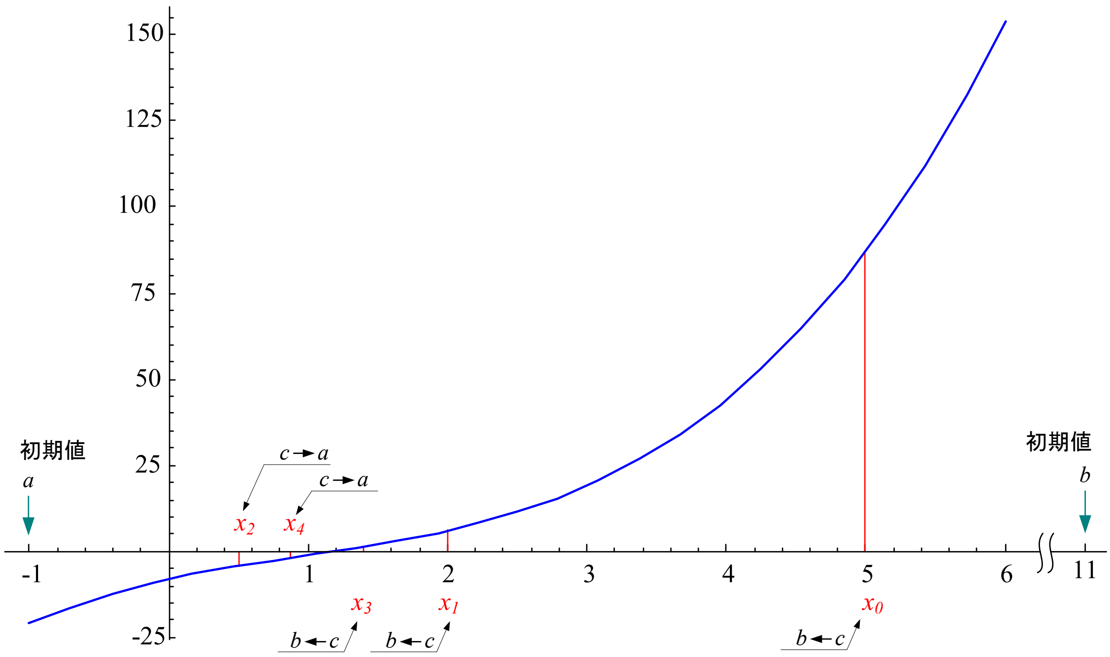

図 2: $f(x) = x^3 - 3x^2 + 9x - 8$ の実数解を 2 分法で解散し，その解の収束の様子を示している．初期値は $a = -1, b = 11$ として，最初の解 $c = x_0 = 5$ が求まり，順次より精度の良い $x_1, x_2, x_3, \cdots$ が求まる．それが，解析解 $x = 1.1659\cdots$ (x 軸との交点) に収束していく様子が分かる．

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．

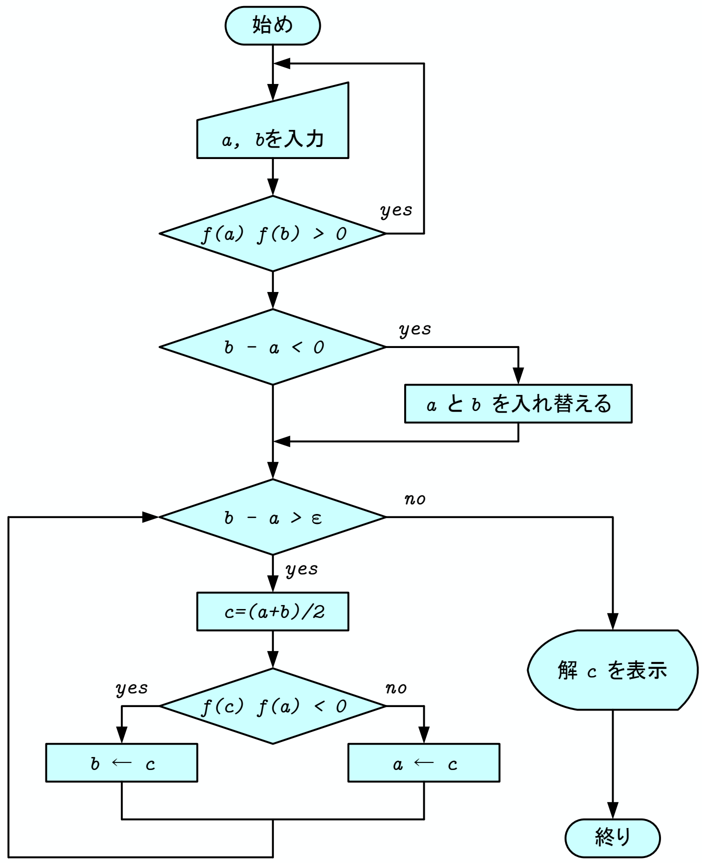

図 3: 2 分法のフローチャート

2.2.3　プログラム

このプログラムをしっかり理解せよ．テストで出題された場合，全く同じプログラムを書く必要はない．同じアルゴリズムでも同一のプログラムになるとは限らない．

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

```
#include <stdio.h>
#define EPS (1.0e-15)  /* precision of calculation */

double func(double x);

/*==================================================================*/
/*        main function                                             */
/*==================================================================*/
int main(void){
  double a, b, c, test;
```

```
  char temp;
  int i=0;

  do{
    printf("\ninitial value a = ");
    scanf("%lf%c", &a, &temp);

    printf("initial value b = ");
    scanf("%lf%c", &b, &temp);

    test=func(a)*func(b);

    if(test >= 0){
      printf("   bad initial value !!  f(a)*f(b)>0\n\n");
    }
  }while(test >= 0);

  if(b-a < 0){
    c=a;
    a=b;
    b=c;
  }

  while(b-a > EPS){

    c=(a+b)/2;
    if(func(c)*func(a) < 0){
      b=c;
    }else{
      a=c;
    }

    i++;
  }

  printf("\nsolution x = %20.15f\n\n",c);

  return 0;
}

/*================================================================*/
/*          definition function                                   */
/*================================================================*/
double func(double x){
  double y;

  y=x*x*x-3*x*x+9*x-8;

  return y;
}
```

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 α 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 α 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．ε 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 x_i がどのように真の解 α に収束するか考える．x_{i+1} と真値 α 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場合，式 (5) を再び計算するだけで，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．

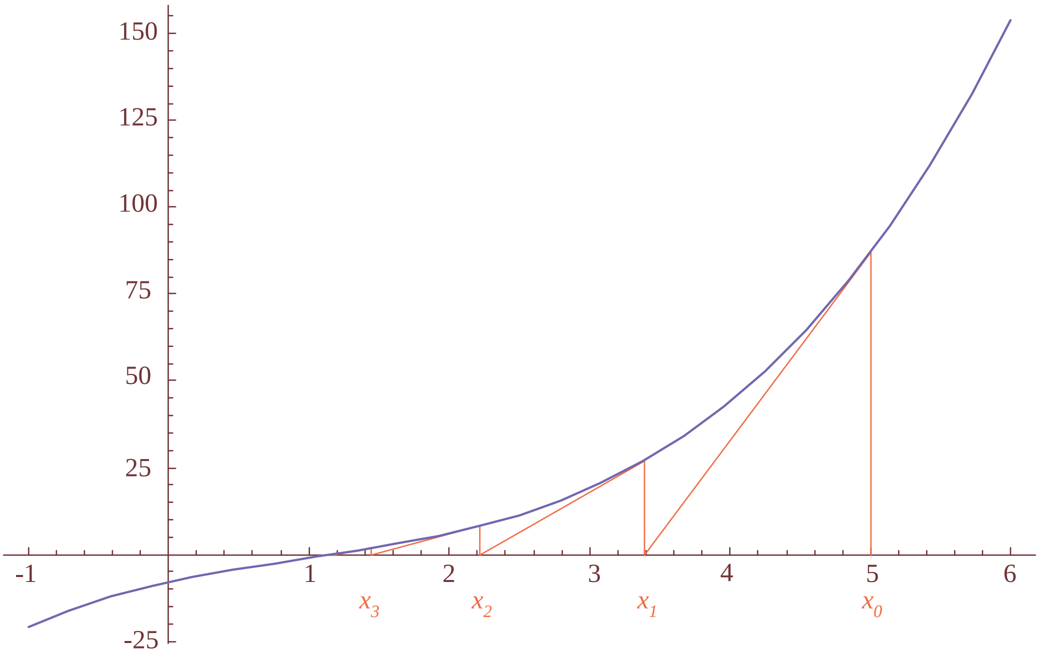

図 4: $f(x) = x^3 - 3x^2 + 9x - 8$ の実数解をニュートン法で計算し，解の収束の様子を示している．初期値 $x_0 = 5$ から始まり，接線と x 軸の交点からより精度の高い回を求めている．

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．

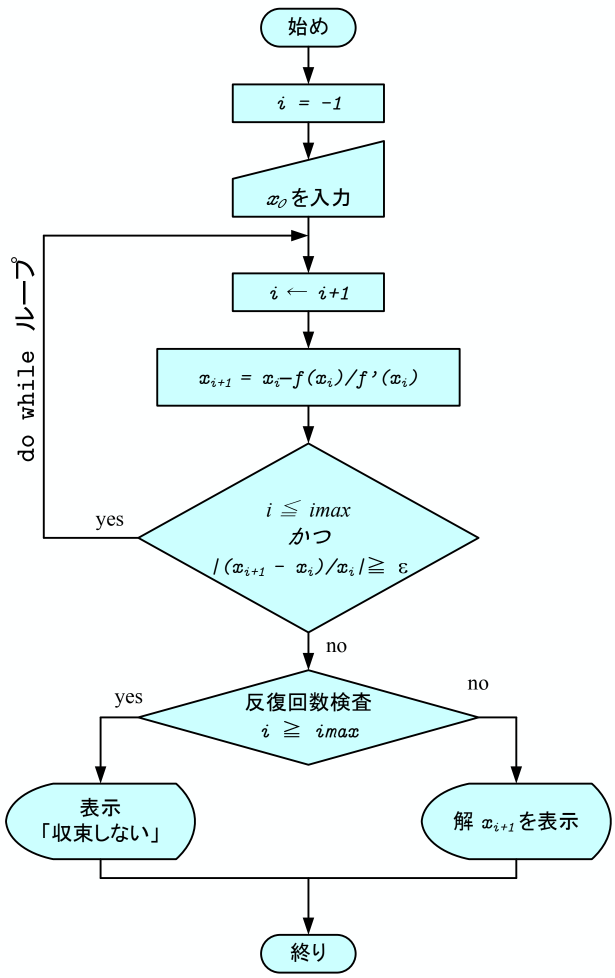

図 5: ニュートン法のフローチャート

2.3.3 プログラム

このプログラムをしっかり理解せよ．テストで出題された場合，全く同じプログラムを書く必要はない．同じアルゴリズムでも同一のプログラムになるとは限らない．

ニュートン法では，導関数の計算が必要である．通常の方程式であれば，導関数は計算できるはずである．計算した導関数をプログラム中に記述する．

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

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

```c
#define EPS (1.0e-15)             /* precision of calculation */

double func(double x);
double dfunc(double x);

/*==================================================================*/
/*        main function                                             */
/*==================================================================*/
int main(void)
{
  double x[IMAX+10];
  char temp;
  int i=-1;

  printf("\ninitial value x0 = ");
  scanf("%lf%c", &x[0], &temp);

  do{
    i++;
    x[i+1]=x[i]-func(x[i])/dfunc(x[i]);
  }while(i<=IMAX && fabs((x[i+1]-x[i])/x[i]) >= EPS);

  if(i>=IMAX){
    printf("\n  not converged !!! \n\n");
  }else{
    printf("\niteration = %d\nsolution  x = %20.15f\n\n",i,x[i+1]);
  }

  return 0;
}
/*==================================================================*/
/*        definition function                                       */
/*==================================================================*/
double func(double x)
{
  double y;

  y=x*x*x-3*x*x+9*x-8;

  return y;
}
/*==================================================================*/
/*        definition derived function                               */
/*==================================================================*/
double dfunc(double x)
{
  double dydx;

  dydx=3*x*x-6*x+9;

  return dydx;
}
```

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．

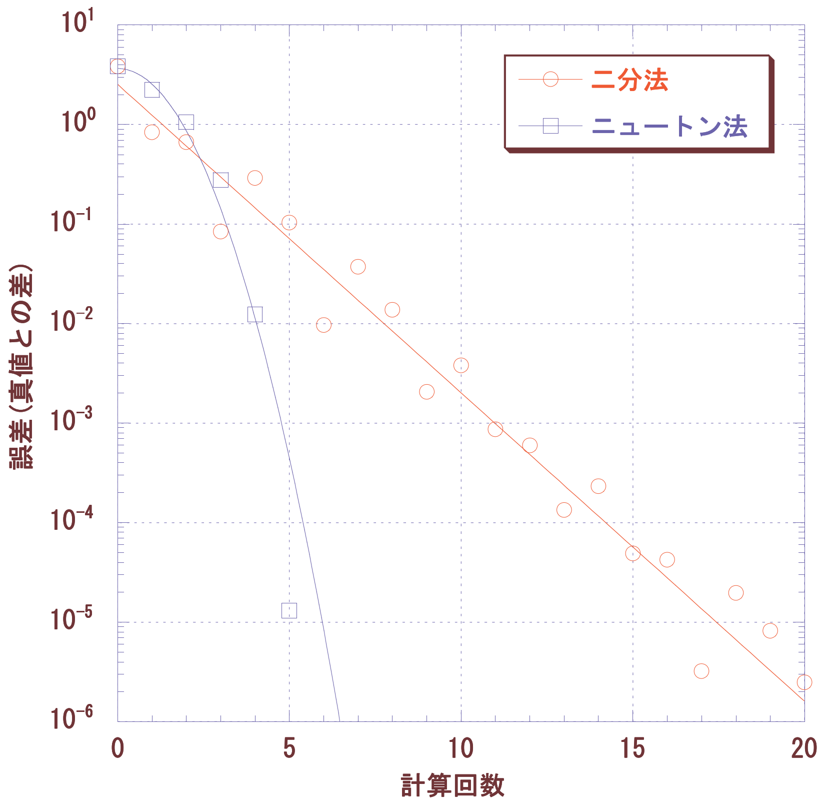

図 6: 二分法とニュートン法の計算回数 (反復回数) と誤差の関係

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．通常は反復回数の上限を決め，無限ループを防ぐ．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．

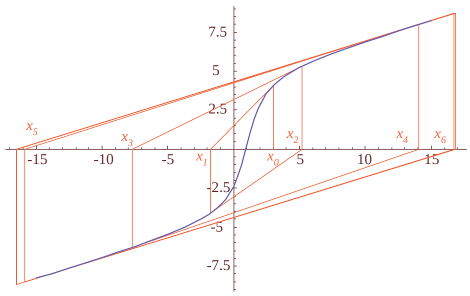

図 7: ニュートン法で解が求まらない場合

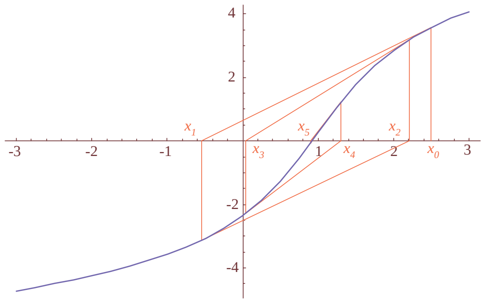

図 8: ニュートン法で解が求まる場合