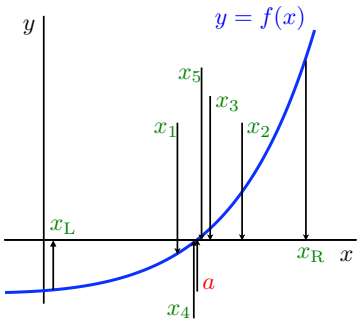
代数方程式の数値解法

方程式

- x の関数 $f(x)$ が与えられたとき
  - $f(x) = 0$ ：x についての方程式
  - $f(a) = 0$ となる a の値: 方程式の解(根)
- 計算機を用いて a の値を数値的に求める
  - 二分法
  - ニュートン法

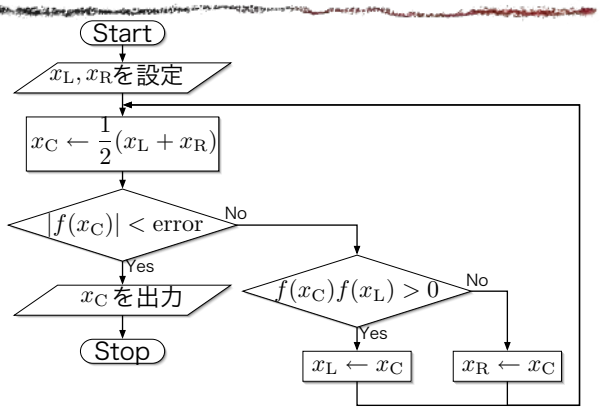
二分法

$$f(a) = 0$$
$$f(x_\mathrm{L}) f(x_\mathrm{R}) < 0$$
$$\Rightarrow x_\mathrm{L} < a < x_\mathrm{R}$$



$$x_1 = \frac{1}{2}(x_\mathrm{L} + x_\mathrm{R})$$
$$x_2 = \frac{1}{2}(x_1 + x_\mathrm{R})$$
$$x_3 = \frac{1}{2}(x_1 + x_2)$$
$$x_4 = \frac{1}{2}(x_1 + x_3)$$
$$x_5 = \frac{1}{2}(x_4 + x_3)$$
$$\vdots$$
$$x_n \longrightarrow a \ (n \to \infty)$$

二分法のアルゴリズム



二分法のプログラム(1)

```
xcentre = (xleft + xright)/2.0;
fcentre = funcv(xcentre);
if (fabs(fcentre) < ERR) {
  printf("x = %f\n", xcentre);
  return 0;
}
fleft = funcv(xleft);
if (fleft*fcentre >= 0.0) {
  xleft = xcentre;
}
else {
  xright = xcentre;
}
```

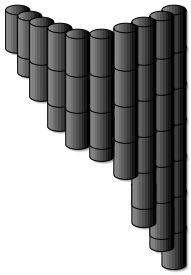
二分法のプログラム(2)

```
#include <stdio.h>
#include <math.h>
#define N 10000
#define ERR 1.0e-7
double funcv(double x);
int main(void){
  int i;
  double xcentre, xleft = 0.0, xright = 2.0;
  double fcentre, fleft, fright;
  for (i=0; i<N; i++) {
        二分法のプログラム(1)
  }
  printf("SOLUTION NOT FOUND\n");
  return 1;
}
```

例題1 (二分法)

- 2の12乗根を求める。

$$f(x) = x^{12} - 2$$

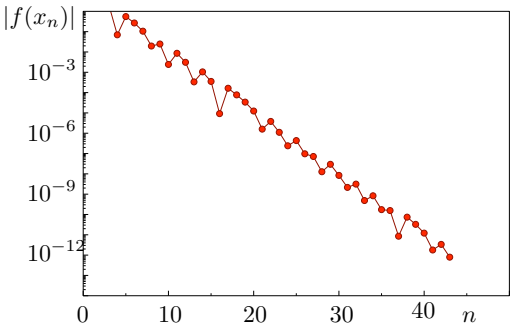


```
double funcv(double x){
  double x3, x6;
  x3 = x*x*x;
  x6 = x3*x3;
  return (x6*x6 - 2.0);
}
```

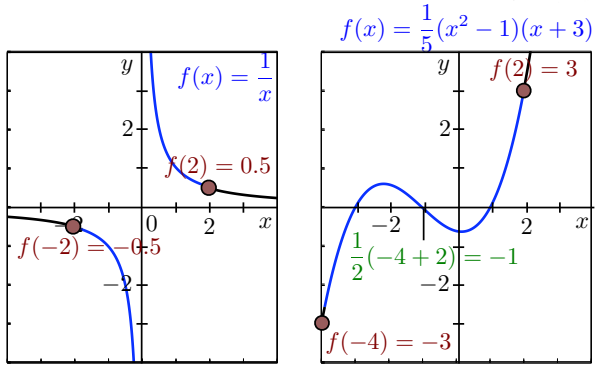
例題1(二分法による計算結果)

	x_L	x_C	x_R	$f(x_C)$
1	1	1.5	2	127.746338
2	1	1.25	1.5	12.551915
3	1	1.125	1.25	2.109891
4	1	1.0625	1.125	0.06989
5	1	1.03125	1.0625	0.553336
6	1.03125	1.046875	1.0625	0.267242
7	1.046875	1.054688	1.0625	0.105539
8	1.054688	1.058594	1.0625	0.019605
9	1.058594	1.060547	1.0625	0.024689
10	1.058594	1.05957	1.060547	0.00243
11	1.058594	1.059082	1.05957	0.008615
12	1.059082	1.059326	1.05957	0.003099

例題1(二分法の解への漸近)



二分法の注意

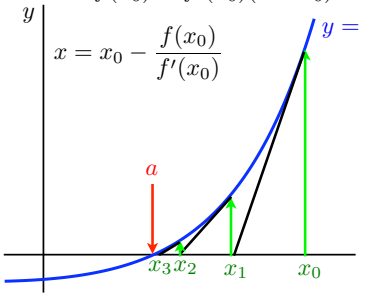


ニュートン法

$$y - f(x_0) = f'(x_0)(x - x_0)$$

$$0 - f(x_0) = f'(x_0)(x - x_0)$$

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$



$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

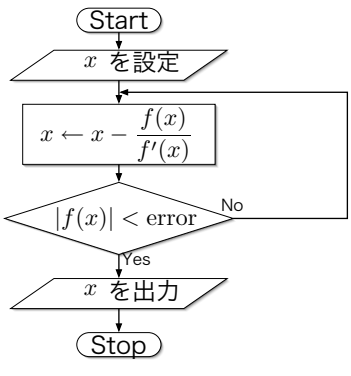
$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}$$

$$\vdots$$

$$x_n \longrightarrow a \ (n \to \infty)$$

ニュートン法のアルゴリズム



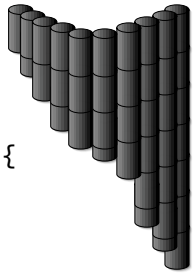
ニュートン法のプログラム(1)

```
for (i=0; i<N; i++) {
  f = funcv(x);
  if (fabs(f) < ERR) {
    printf("x = %f\n", x);
    return 0;
  }
  df = funcdv(x);
  x = x - f/df;
}
```

ニュートン法のプログラム(2)

```
#include <stdio.h>
#include <math.h>
#define N 10000
#define ERR 1.0e-7
double funcv(double x);
double funcdv(double x);
int main(void){
  int i;
  double x = 2.0, f, df;
  ニュートン法のプログラム(1)
  printf("SOLUTION NOT FOUND\n");
  return 1;
}
```

例題1 (ニュートン法)



- 2の12乗根を求める。

$f(x) = x^{12} - 2$

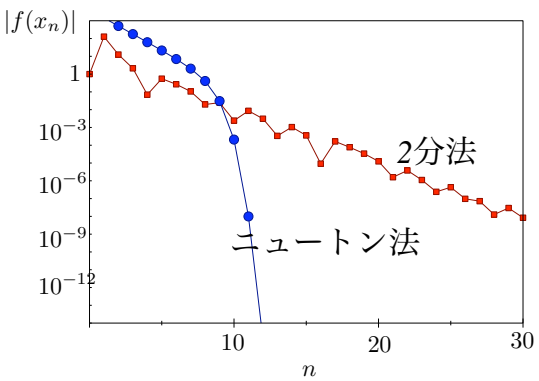
$f'(x) = 12x^{11}$

```
double funcdv(double x){
  double x2, x3, x4;
  x2 = x*x;
  x3 = x2*x;
  x4 = x2*x2;
  return 12.0*x4*x4*x3;
}
```

ニュートン法による計算結果

	x	$f(x)$
0	2	4.094E+03
1	1.833415	1.440542E+03
2	1.680842	5.065371E+02
3	1.541323	1.777723E+02
4	1.414308	6.205135E+01
5	1.300129	2.132589E+01
6	1.201075	7.012397E+00
7	1.123197	2.031529E+00
8	1.076031	4.093315e–01
9	1.060797	3.041653e–02
10	1.059472	2.079743e–04
11	1.059463	9.910903e–09
12	1.059463	1.332268e–15

ニュートン法の解への漸近



ニュートン法の注意

$\tanh x + 0.2x + 0.3 = 0$ を解く　$f(x) = \tanh x + 0.2x + 0.3$

$f'(x) = \dfrac{1}{\cosh^2 x} + 0.2$

```
double funcv(double x){
  return (tanh(x) + 0.2*x + 0.3);
}
double funcdv(double x){
  double ch;
  ch = cosh(x);
  return (1.0/(ch*ch) + 0.2);
}
```

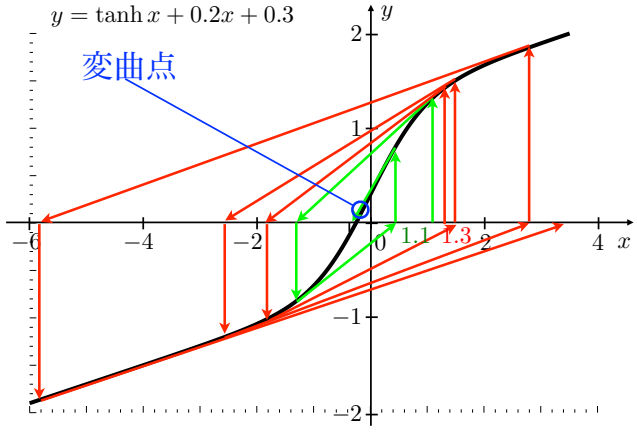
ニュートン法の注意

$f(x) = \tanh x + 0.2x + 0.3$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

n	x_n	$f(x_n)$	x_n	$f(x_n)$
0	1.1	1.320499	1.3	1.421723
1	–1.261402	–0.803730	–1.808045	–1.009242
2	0.430546	0.791887	1.533904	1.517871
3	–0.334307	–0.089247	–2.569401	–1.202219
4	–0.252882	0.001797	2.817221	1.856324
5	–0.254461	0.000001	–5.849533	–1.869890
6	–0.254461	0	3.498365	1.997845

ニュートン法の注意



2分法の特徴

- $f(x_\mathrm{L})f(x_\mathrm{R}) < 0$かつ領域 $x_\mathrm{L} < x < x_\mathrm{R}$で $f(x)$が連続になる x_Lと x_Rさえ見つければ、確実に解が求まる。
- 解の誤差は N回繰り返すと 2^{-N}で小さくなる。
- 関数値のゼロに対する誤差、収束性は(ニュートン法に比べると)あまり良くない。

ニュートン法の特徴

- 一つの入力値から解を求めることができる。
- (二分法に比べて)非常に高速に(少ない繰り返し回数で)精度の高い解を見つけることができる。
- 関数の微分値を必要とする。
- 初期値と解の間に変曲点があると解が求まらない場合がある。

解の収束をどう決定するか

- 最初に設定した誤差 ε に対して

(1) $|x_n - x_{n-1}| < \varepsilon$
  - 解が任意の精度で決定できる
  - 二分法では繰返回数を設定するだけでよい

(2) $|f(x_n)| < \varepsilon$
  - $f(x)$の計算精度内で意味のある解を求める
  - 二分法の問題点を回避できる