数値解析
Numerical Analysis

関数，方程式

Function, Equation

電気･電子情報工学系　　市川　周一

Shuichi Ichikawa, Dept. EEIE

本日の内容
Index

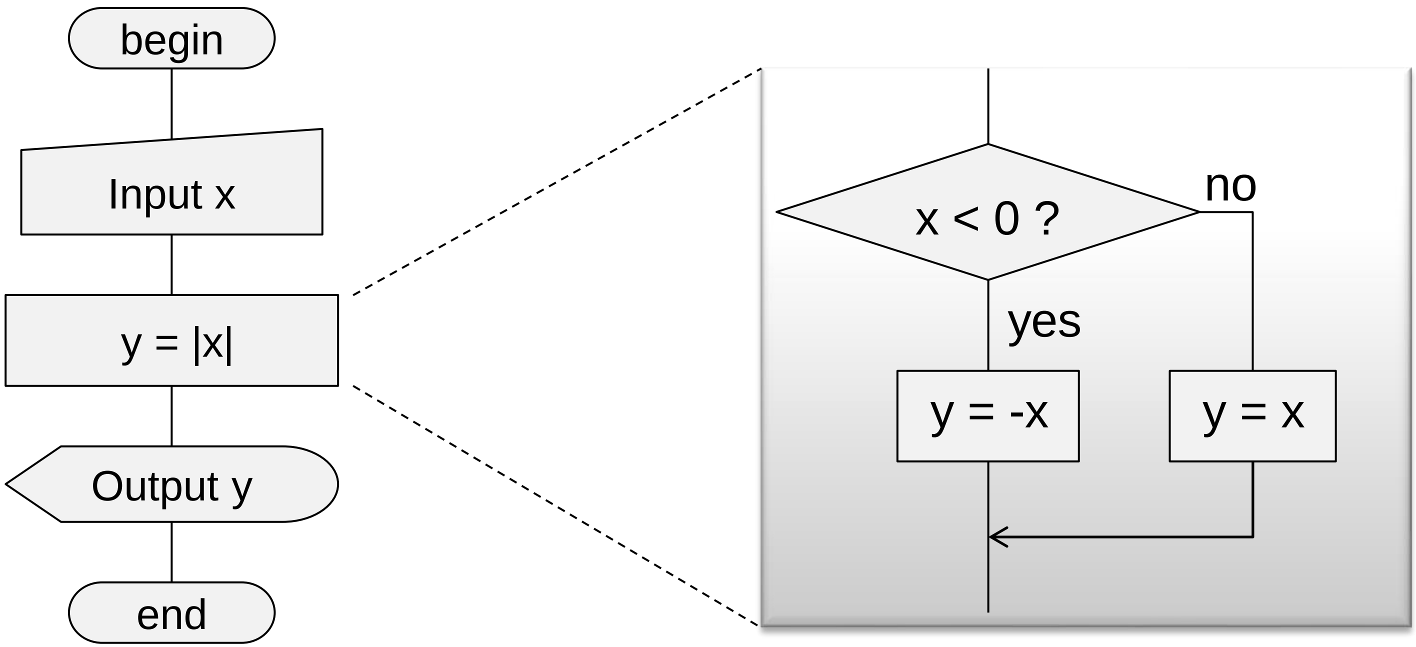
- 関数(function)とは
  - 関数宣言 (function declaration)
  - 関数定義 (function definition
- 非線型方程式の求解法
  How to solve a non-linear equation
  - 二分法 (bisection)
  - Newton法 (Newton method)

関数は何故必要か？
Why do you need functions?

- 同じ処理を色々な場所で繰返すことは多い
  It is very popular to repeat the same process in many places.
  - 同じコードを何度も書きたくない
  - 似て非なるコードをたくさん作ると，無駄な労力が増える
  - 1箇所で処理すれば，動作確認や修正も1箇所だけで済む
- プログラムが構造化される
  Refer to “structured programming”
  - 理解しやすい，修正しやすい，改良しやすい
  - キーワード 『プログラムの構造化』，『段階的詳細化』
- 他人のプログラム（部品）を借りることができる
  You can borrow the functions (parts) from other persons.
  - 便利なソフトウェア部品を集めたもの → ライブラリ Library
  - ライブラリに含まれる，便利な部品 → ライブラリ関数 Library functions
    - 関数printfやscanfもライブラリ関数です！（standard C library = libc）

復習：　段階的詳細化
Stepwise refinement

- 処理の流れを大まかに記述してから，各部を詳細化
- トップダウン設計
  - プログラムを，上位（機能）→下位（実装）へと詳細化する
- それに対応した構造をプログラムに与えるには？



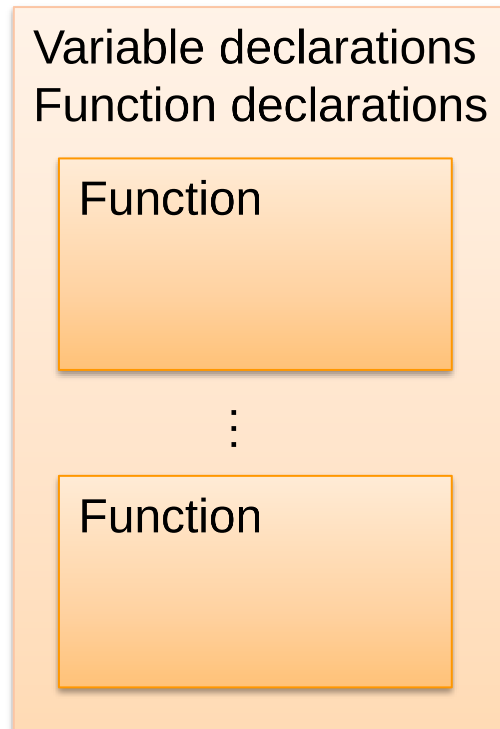
C言語プログラムの構造
Structure of a C file

- 変数宣言 Variable declaration
  - データ型　変数名
  - 例: `int count;`

変数が複数あってもよい

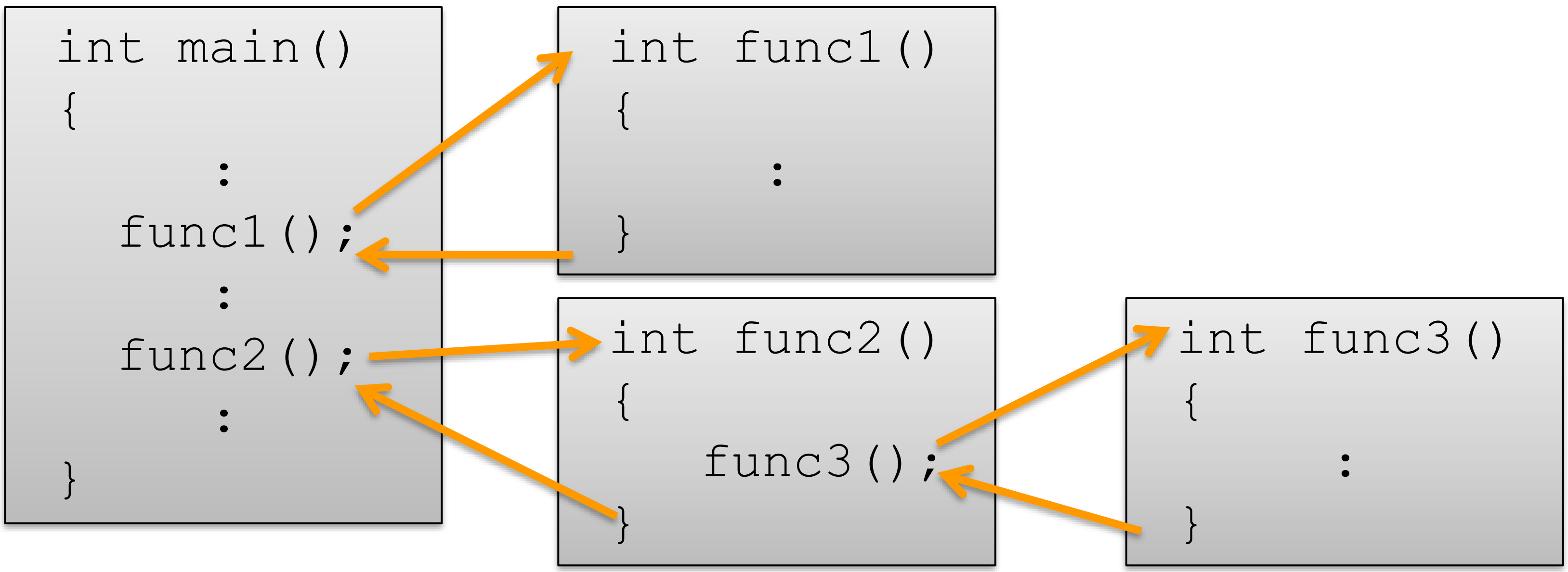
- 関数宣言 Function declaration
  - 関数の型　関数名(引数の型　引数名)
  - 例: `double exp(double x);`
  - 本体は別に定義
- 関数 Functions
  - 少なくとも`main()`は必要
  - 各関数の構造は後ほど説明する

File:　xxx.c



関数の呼び出し関係
Calling functions

- プログラムの構造化 Structured programming
  - 機能を関数に分けて実現する Write functions for each purposes
- 関数から関数を呼んでよい Function call
  - 実際，関数main()から関数を呼ぶ
  - ポイント：printfも関数です！



関数を呼び出す

- 関数名(実引数, …)
  - 例: `printf("Hello¥n");` ← 引数は文字列1個
  - 例: `initialize();` ← 引数が無くてもよい
- 関数の値を使う場合
  - 他の値と同様に，代入，演算などに利用できる
  - 例: `x = sin(t);` ← 関数sinの値をxに代入
  - 例: `x = sin(t) + z;` ← 関数sinの値を演算に利用
  - 例: `x = exp(sin(t));` ← 関数の値を関数の引数に
- 関数の値を使わない場合
  - 例: `scanf("%d", &x);` ← 関数scanfの値を無視

関数の構造
Structure of a function

- 関数の型　＝　関数の返す値の型
  - 省略するとint型が推定されるが，int型でも明示すべきである
  - 値を返さない関数では，voidと宣言する
- 引数　＝　引数の型　引数名
  - 引数は複数あってもよい
  - 引数が2つの例： `int func(int arg1, int arg2)`

```
関数の型　関数名（引数の型　引数名）
{
    変数宣言
    実行文
}
```

```
Type　FuncName（ArgType Argument）
{
    Variable declarations
    Sentences
}
```

関数から戻る：　return文

- 関数から呼び出し側へ戻る方法
  1. 関数末端に到達する
  2. “return”文を実行する
- Return文の使い方
  - 関数から値を返す場合
    - 文法：　`return 値;　// “値”は，定数，変数，式などで指定`
    - 例：　`return 0;`
    - 例：　`return x;`
    - 例：　`return (x%y);`
  - 値を返さずに戻る場合
    - 例：　`return;`
    - 例：　`if (x < 0) return;`

参考： main()も関数の一つである

- 関数mainの型は int である
  - 今までは関数の型を省略していた
    - 省略すればintを推定するので問題なかった
  - 今後は明示的にintと書くとよい
- 関数mainは，どんな値を返すべきか？
  - プログラム毎に自由に決められるが…
  - 以下のような慣習がある
    - 正常終了 → 0を返す
    - 異常終了 → エラーコード（0以外）を返す
- プログラムが値を返して，誰が見るのか？
  - プログラムを呼び出したプログラムが見る
  - 例えば，Linuxのシェル

なぜ関数宣言が必要なのか
Why function declarations are necessary

- 関数の型と引数（数と型）を，使用前に宣言する
  Declare the type and name of each function
  - 宣言しないとint型が仮定される
    Type int is assumed as a default
  - 後で関数定義と不一致が生じれば，コンパイルエラーが生じる
    If the assumed type mismatches the actual type, compiler terminates with an error.

コンパイルエラーが出る例

```
#include <stdio.h>

int main()
{
    printf("%e¥n", sum(1.5, 2.0));
    return 0;
}

float sum(float a, float b)
{
    return a+b;
}
```

int型を推定

実はfloat型

コンパイルエラー
（sumの型が不一致）

```
$ cc example1.c
example1.c:10: error: conflicting types for 'sum'
example1.c:5: error: previous implicit declaration of 'sum' was here
```

関数宣言の例
Function declaration

- 前頁のプログラム例に，関数宣言を入れてみる
  - コンパイルエラーは無くなり，正常に動作する

```
#include <stdio.h>

float sum(float a, float b);

int main()
{
    printf("%e¥n", sum(1.5, 2.0));
    return 0;
}

float sum(float a, float b)
{
    return a+b;
}
```

関数sumの宣言

関数sumを利用

関数sumの定義

正常動作の例

```
$ cc example2.c

$ ./a.out
3.500000e+00
```

エラーなし！

正しい

別の解決法
Another solution

- 関数sumを先に定義してから，関数mainで使用する

```
#include <stdio.h>

float sum(float a, float b)
{
     return a+b;
}

int main()
{
     printf("%e¥n", sum(1.5, 2.0));
     return 0;
}
```

使用前に関数宣言する代わり，
使用前に関数を定義した

正しい型でコンパイルされる

```
$ cc example3.c

$ ./a.out
3.500000e+00
```

エラーなし！

正しい

一見これで万事解決のようだが…

関数宣言が必要な理由

- 関数定義は，いつでも先にできるとは限らない
  - Sometimes, declarations are essential!

どちらを先にしても，問題が生じる

```
関数A(...)
{
    関数B(...);
}

関数B(...)
{
    関数A(...);
}
```

→

関数Aの定義が先なら，関数Bの関数宣言は必要

```
関数B(...); // 関数定義

関数A(...)
{
    関数B(...);
}

関数B(...)
{
    関数A(...);
}
```

→

一般的には全て関数宣言

```
関数A(...); // 関数定義
関数B(...); // 関数定義

関数A(...)
{
    関数B(...);
}

関数B(...)
{
    関数A(...);
}
```

関数宣言なしで動く場合もあるが…

- 関数宣言を省略するとint型が仮定される
  - …ので，関数がint型なら省略可能ではあるが…
  - You can omit the declaration if the function type is int.
- 型がintでも，**省略せずに関数宣言することを強く勧める**
  - **It is strongly recommended to declare the functions explicitly.**

```
#include <stdio.h>

int main()
{
    printf("%d¥n", sum(5, 2));
    return 0;
}

int sum(int a, int b)
{
    return a+b;
}
```

int型を推定

実際もint型

```
$ cc example4.c

$ ./a.out
7
```

エラーなし！

正しい

復習：　libmの指数関数 e^x = exp(x)

- 関数exp(x)の詳細は，マニュアルで確認すること

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

main()
{
    double x;

    x = 1.0;
    printf("exp(%e) = %e¥n", x, exp(x));
}
```

printfのために必要

expのために必要

expの関数宣言がない?
double exp(double x);

ヘッダファイルmath.hには
数学関数の型宣言などが
含まれている

#include <math.h>　で
Expなど数学関数の
宣言も行われるので，
別途宣言する必要はない

```
$ cc test.c -lm
$ ./a.out
exp(1.000000e+00) = 2.718282e+00
```

数学ライブラリを使うときは –lm が必要

先週の例題：　exp

- 指数関数の多項式展開をホーナー法で計算
- 関数として書き換えてみよう

```
#include <stdio.h>

main()
{
    double x = 1.0;
    double v = 1.0;
    int i;

    for (i = 10; i > 0; i = i-1) v = v*x/i + 1;
    printf("exp(%e) = %e¥n", x, v);
}
```

```
$ cc exp0.c
$ ./a.out
exp(1.000000e+00) = 2.718282e+00
```

関数expを定義

```
$ cc exp1.c
$ ./a.out
exp(1.000000e+00) = 2.718282e+00
```

```
#include <stdio.h>

double exp(double x);

int main()
{
    double x = 1.0;

    printf("exp(%e) = %e¥n", x, exp(x));
}

double exp(double x)
{
    double v = 1.0;
    int i;

    for (i = 10; i > 0; i = i-1) v = v*x/i + 1;
    return v;
}
```

関数宣言

関数本体

関数expを定義した例

数学ライブラリのexpを
使うのと類似の記述になる

本日の内容
Index

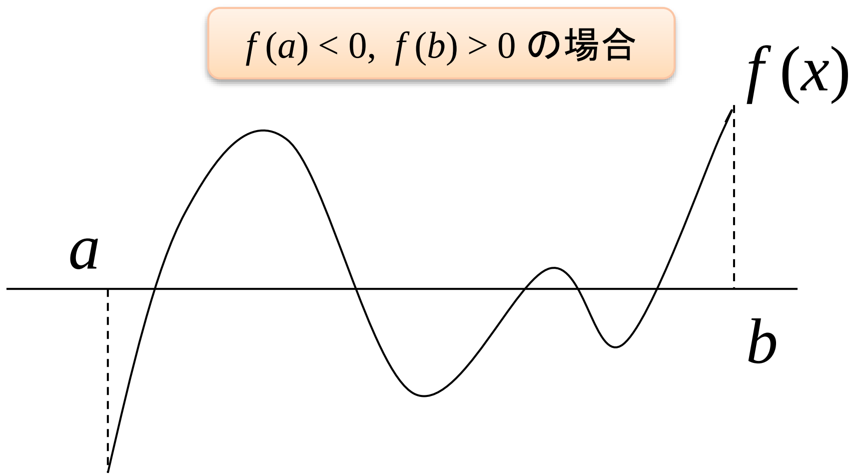
- 関数(function)とは
  - 関数宣言 (function declaration)
  - 関数定義 (function definition
- 非線型方程式の求解法
  How to solve a non-linear equation
  - 二分法 (bisection)
  - Newton法 (Newton method)

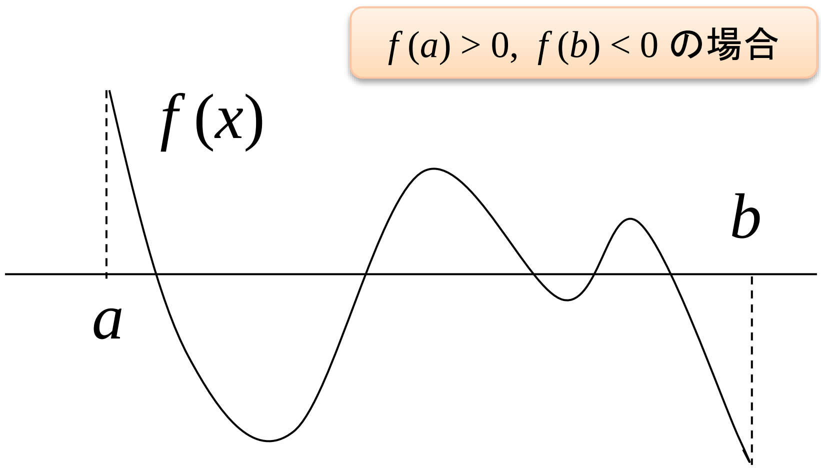
非線形方程式 $f(x)=0$ の実数解を求める
Solve a non-linear equation $f(x)=0$

- 【注意】 方程式の形や性質に応じて色々な解法がある
  Various methods exist according to the equation
  - 本講義では，反復法の基礎(だけ)を学ぶ
    This lecture introduces the basics of iterative methods
  - 線形方程式については，別に各種の解法がある
- 反復法 Iterative method
  - 初期値(initial value) x_0 を与える
  - 漸化式 $x_{n+1} = g(x_n)$ により，逐次 $x_1, x_2, \cdots$ を求める
  - 数列が(ほぼ)収束したら終了する
- 代表的な例
  - 二分法 bisection
  - ニュートン(Newton)法 ← 別名 Newton-Raphson法

【数学】 定理 theorem

- 関数 $f(x)$ が閉区間 $[a, b]$ で連続で, $f(a)\,f(b) < 0$ ならば, 方程式 $f(x) = 0$ は開区間 (a, b) で少なくとも一つの解を持つ
  - 証明: 中間値の定理から自明

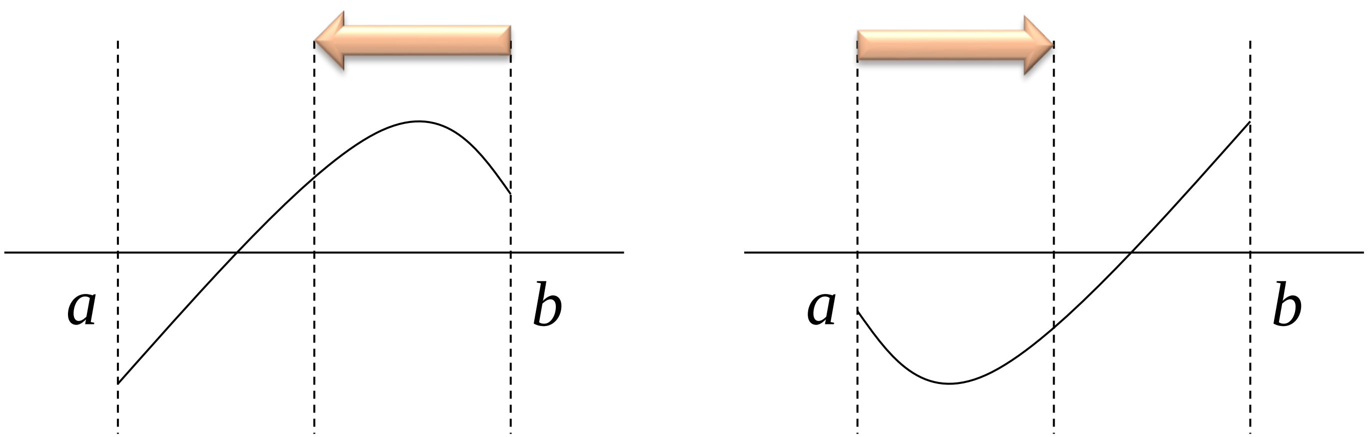




二分法 bisection

- 区間縮小法の一種　reduces the intervals
  - $f(a)\,f(b) < 0$ を守りながら区間を縮小してゆく
  - $a \approx b$ になったら，それを解（近似値）とする
- 二分法　＝　区間を1/2に縮小する
  - 以下の説明では $f(a) < 0,\ f(b) > 0$ とする
  - $f((a+b)/2) < 0$　ならば　$a \leftarrow (a+b)/2$
  - $f((a+b)/2) > 0$　ならば　$b \leftarrow (a+b)/2$

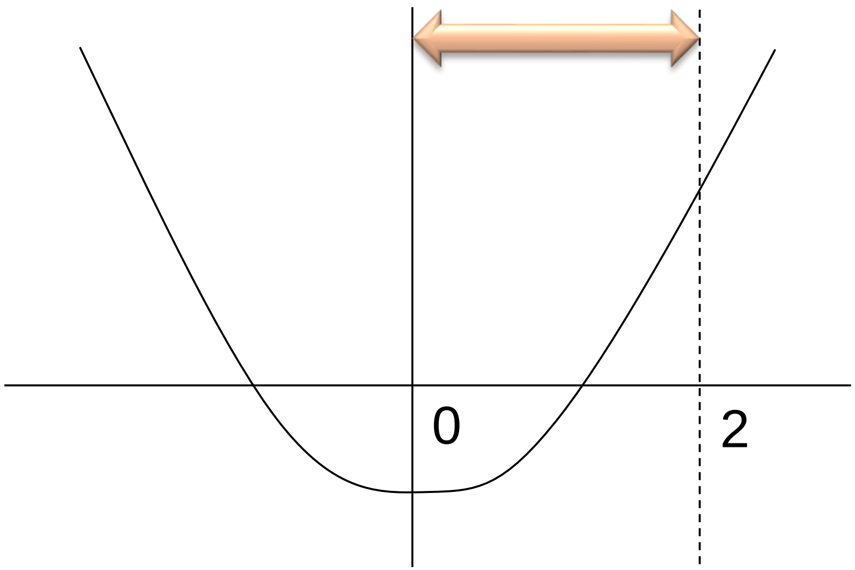
$f(a) > 0,\ f(b) < 0$ の場合は
自分で考えてみよ.
場合分けせずに判定する
条件を書けるか？



少なくとも
解の1つが
求まる

例題： 方程式 $f(t) = t^2 - x = 0$

- 別の言い方： xの平方根を求める $(0 < x \leq 1)$
  - 正の平方根を求めるとする $(t > 0)$



$$f(0) = -x < 0$$

$$f(2) = 4 - x > 0$$

プログラム例： 二分法による開平
Example: Square root by bisection mothod

- 仮定： $0 < x \leq 1,\ f(lo) < 0,\ \ f(hi) > 0$

```
while (fabs(hi-lo) > 1e-10) {
    md = (lo+hi)/2.0;
    mv = md*md - x;
    if (mv < 0) {
        lo = md;
        lv = mv;
    } else {
        hi = md;
        hv = mv;
    }
}
return (hi+lo)/2.0;
```

変数

- `lo`： 区間下限
- `lv`： $f(lo) < 0$
- `hi`： 区間上限
- `hv`： $f(hi) > 0$
- `md`： 区間中点
- `mv`： $f(md)$

実行例：　二分法　$\sqrt{0.7}$

```
    (lo, hi) = (0.000000e+00, 2.000000e+00)
    (lo, hi) = (0.000000e+00, 1.000000e+00)
    (lo, hi) = (5.000000e-01, 1.000000e+00)
    (lo, hi) = (7.500000e-01, 1.000000e+00)
    (lo, hi) = (7.500000e-01, 8.750000e-01)
    (lo, hi) = (8.125000e-01, 8.750000e-01)
    (lo, hi) = (8.125000e-01, 8.437500e-01)
    (lo, hi) = (8.281250e-01, 8.437500e-01)
    (lo, hi) = (8.359375e-01, 8.437500e-01)
```

途中20回　省略

```
    (lo, hi) = (8.366600e-01, 8.366600e-01)
    (lo, hi) = (8.366600e-01, 8.366600e-01)
    (lo, hi) = (8.366600e-01, 8.366600e-01)
    (lo, hi) = (8.366600e-01, 8.366600e-01)
    (lo, hi) = (8.366600e-01, 8.366600e-01)
    (lo, hi) = (8.366600e-01, 8.366600e-01)
7.000000e-01 -> 8.366600e-01  (err +1.131322e-11)
```

ループ　約3.3回
→ 有効数字1桁

有効数字10桁
→ ループ約33回

Newton法
Newton–Raphson method

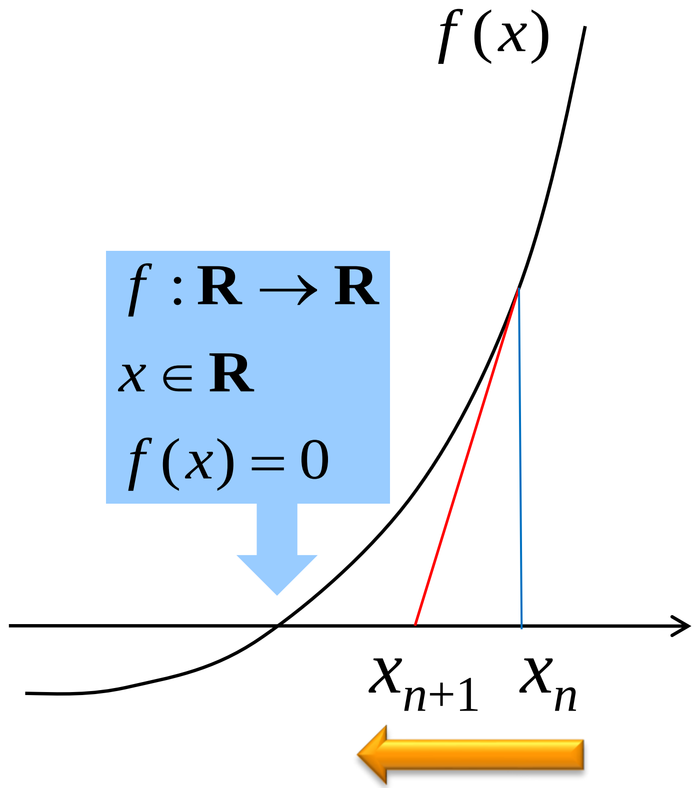
- 方程式の解を求める，反復解法の一種
  - 漸化式で解に収束させる
  - 収束条件の吟味が必要

$x = x_n$における接線の式

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

$y = 0$のとき$x = x_{n+1}$とする

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



参考： Newton法の収束条件 Convergence conditions

- 詳細・証明などは，数値解析の参考書参照のこと
  - Refer to the textbook for more details
- $\alpha\ (a < \alpha < b)$は方程式$f(x) = 0$の解であるとする.
- $f(x)$とx_0が下の条件のいずれか一つを満たす時，Newton法の反復列$\{x_n\}$は単調にαに収束する.

$$(1)\quad f(a) < 0,\ f(b) > 0,\ f''(x) > 0\ (a \le x \le b),\ \alpha < x_0 \le b$$

$$(2)\quad f(a) < 0,\ f(b) > 0,\ f''(x) < 0\ (a \le x \le b),\ a \le x_0 < \alpha$$

$$(3)\quad f(a) > 0,\ f(b) < 0,\ f''(x) > 0\ (a \le x \le b),\ a \le x_0 < \alpha$$

$$(4)\quad f(a) > 0,\ f(b) < 0,\ f''(x) < 0\ (a \le x \le b),\ \alpha < x_0 \le b$$

例：　Newton法による逆数近似
Calculate reciprocal by Newton method

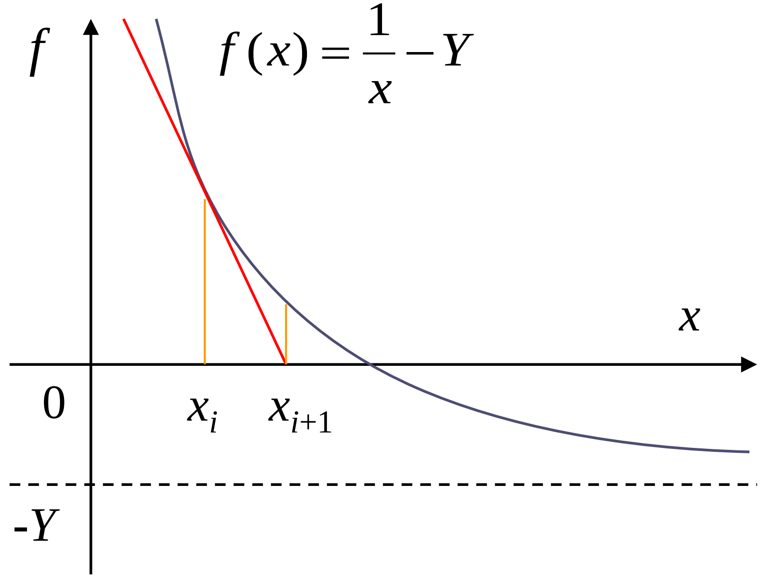
- 初期値x_0には，適切な近似値を用いる
  - 適切でないと収束条件を満たさない可能性がある

$$f(x)=\frac{1}{x}-Y$$

$$f'(x)=-\frac{1}{x^2}$$

$$x_{i+1}=x_i-\left(\frac{1}{x_i}-Y\right)(-x_i{}^2)$$

$$=x_i(2-x_iY)$$



乗算と減算の繰返しで除算が実現できる

Newton法による逆数近似の収束

- 二次の収束
  - 誤差の絶対値→0
  - 有効数字2倍／回
- 計算したい桁数　n bit
- 初期値の有効数字がm bit

$$約\left(\log_2\frac{n}{m}\right)回$$

$$x_{i+1}=x_i(2-x_iY)$$

$$x_i\equiv\frac{1}{Y}(1+\varepsilon_i),\quad |\varepsilon_i|<1$$

$$x_{i+1}=\frac{1}{Y}(1+\varepsilon_i)\left(2-\frac{1}{Y}(1+\varepsilon_i)Y\right)$$

$$=\frac{1}{Y}(1+\varepsilon_i)(1-\varepsilon_i)$$

$$=\frac{1}{Y}(1-\varepsilon_i{}^2)$$

$$\varepsilon_{i+1}=-\varepsilon_i{}^2$$

2次の収束

Newton法による開平
Square root by Newton method

- 方程式は一意に決まらないので，漸化式も色々作れる
  - You can use various equations to calculate square root
- 下の方法は典型的な一例
  - 漸化式を繰り返して適用する
  - $|u_{n+1} - u_n| \leq \varepsilon$ になったら停止する

$\sqrt{x}$を求める方程式

$$f(u) = u^2 - x = 0$$

$$u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}$$

⇒

$$f(u) = u^2 - x$$

$$f'(u) = 2u$$

$$u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)} = \frac{u_n^2 + x}{2u_n}$$

プログラム例： Newton法による開平

- $0 < x < 1$ と仮定して初期値を与えている

```
double newton(double x)
{
    double u, uu;

    u = 2.0;
    uu = u + 1.0;
    while (fabs(uu-u) > 1e-10) {
        uu = u;
        u = (u*u+x)/(2*u);
    }
    return u;
}
```

初期値2.0に固定

$|u_{n+1} - u_n| \leq \varepsilon$ になったら停止

$$u_{n+1} = \frac{u_n^2 + x}{2u_n}$$

実行例：Newton法 $\sqrt{0.7}$

- 二分法と同じ初期値2.0から始めている
  - 二分法（三十数回）より少ないループで求まった
- 条件を満たせばNewton法は二次収束する
  - 有効数字10桁（10進）≒30桁（2進）
  - 1 → 2 → 4 → 8 → 16 → 32　で概ね5～6回で達成するはず

```
     v = 2.000000e+00
     v = 1.175000e+00
     v = 8.853723e-01
     v = 8.380001e-01
     v = 8.366611e-01
     v = 8.366600e-01
7.000000e-01 -> 8.366600e-01  (err +0.000000e+00)
```

別解：　Newton法による開平

- Newton法で$x^{-1/2}$を求めてから，xを掛ける
  - 漸化式に除算がない → ハードウェアで用いられる方法

$\dfrac{1}{\sqrt{x}}$を求める方程式

$$f(u) = \frac{1}{xu^2} - 1 = 0$$

$$u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}$$

→

$$f(u) = \frac{1}{xu^2} - 1$$

$$f'(u) = -\frac{2}{xu^3}$$

$$u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)} = \frac{u_n}{2}\left(3 - xu_n{}^2\right)$$

注意点
Cautions

- いきなり反復法を使うことはできない
  Iterative methods are not all-mighty
  - 方程式の性質を事前に良く検討しておく
  - 最初に $f(x)$ の振る舞いを確認しておく
  - 適切な解法を選んで用いる
- 実数解が複数ある場合は？
  If the equation has two or more solutions?
  - それぞれの解の近傍で反復法を使う
    Find the solutions one by one
  - 収束性，初期値の検討が必要
- 実数解がない場合もある
  If the equation has no real solutions?
  - 反復法が収束しない，等
  - プログラムが停止するように工夫しておく
    Your program must stop after some iterations.