工学基礎実験実習
～C言語の基礎～
ニュートン法

岡山大学工学部情報系学科
後藤　佑介

前回までのまとめ

- C言語の基礎を学習
  - 変数と定数
  - 算術
  - 制御の流れ
  - 基本的な入出力
  - 関数
- LaTeXによるレポート作成，提出
  - 数値計算方法である Newton 法を例にして学習

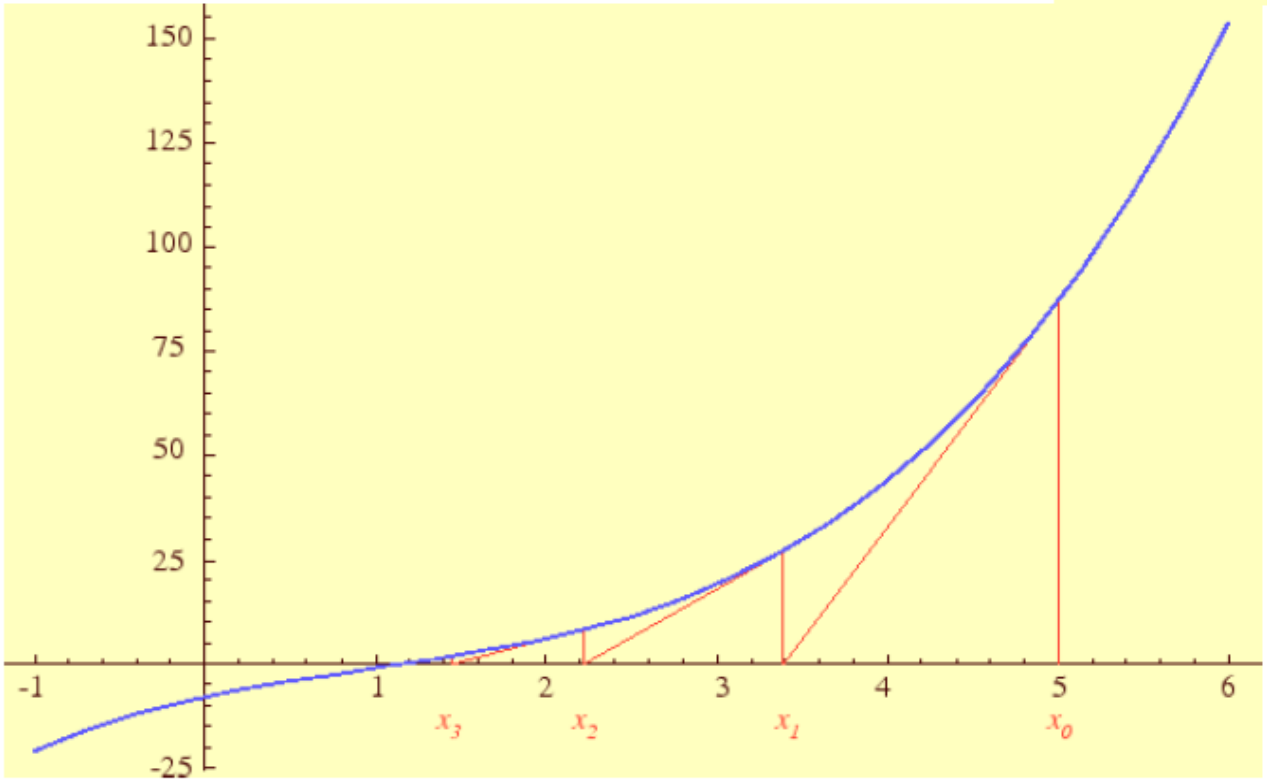
今日やること

- ニュートン法のプログラム作成
  - プログラムを一から作成
  - フローチャート，C言語の文法を学習しよう．
- 注意点
  - プログラミングは考えて失敗しながら完成させるのが上達への近道
  - どうしても分からないときはサンプルプログラムを見て勉強してもよいが，慎重に．
    ⇨ 今は良いが，今後の授業で結局困るのは自分．
  - この後のスライドは空白が多いので理解して読むこと．
    どこかに書き留めておくとよい．

ニュートン法

- $f(x) = 0$ となる x を求める.
- 曲線を直線で近似して計算する.

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



↓

y＝0とするには？

↓

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

ニュートン法の導出手順

（手順）$y = f(x)$のグラフ:

(1) 初期値: $y = f(x) = 0$ の解の候補として, 適当な数を $x0$ として与える.

(2) 点$(x0, f(x0))$ における$y = f(x)$の接線を以下の式で求める.

$$y = f'(x0)\,(x - x0) + f(x0)$$

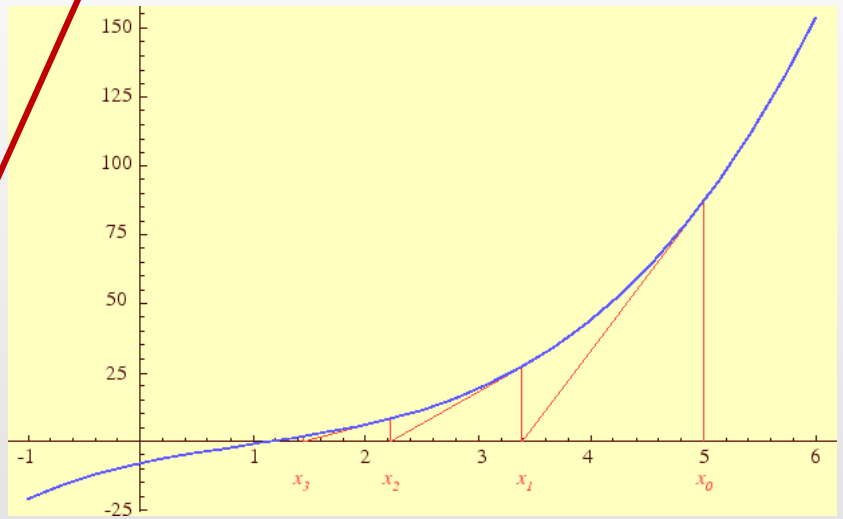
(3) この接線が x 軸と交わる場所を $(x1, 0)$ とすると,

$$0 = f'(x0)\,(x1 - x0) + f(x0)$$

$$\Leftrightarrow x1 = x0 - f(x0) / f'(x0)$$

(4) この操作を何回か繰り返して, 解を導出する.

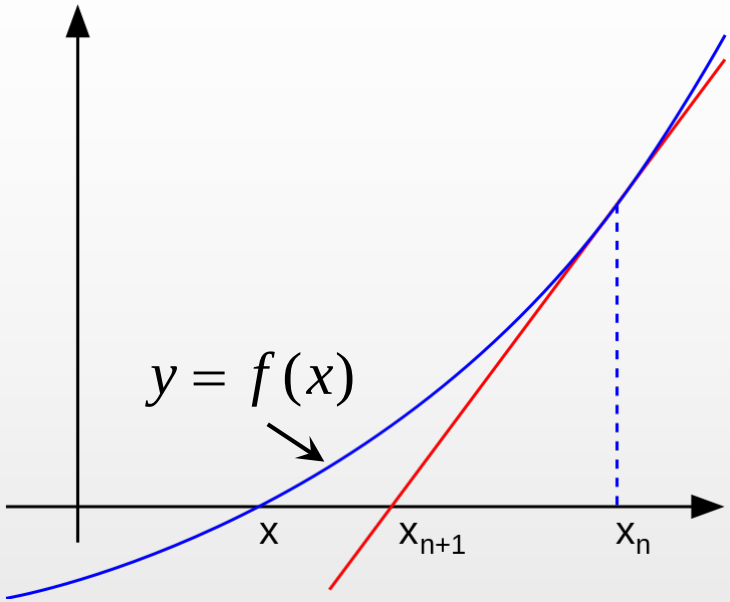
$$y - f(x_i) = f'(x_i)(x - x_i)$$



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

Cプログラミング：ニュートン法（レポート課題）

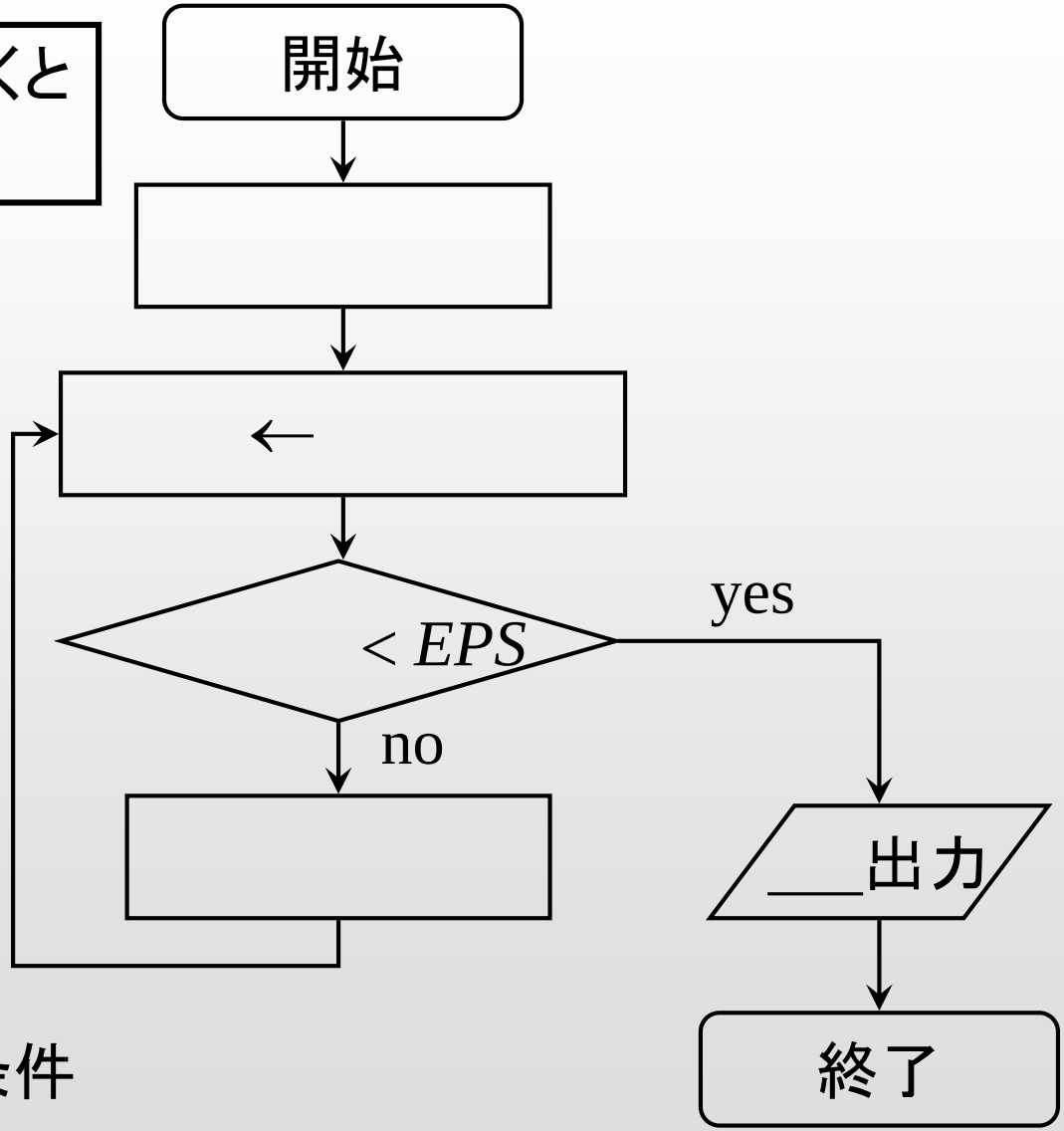
- ニュートン法とは
  - 方程式の解を数値計算で解くための＿＿＿＿による求根アルゴリズムの1つ



- 手順
  1. 予想される真の解に近いと思われる値 x_0 を一つ取る.
  2. グラフ上の点 ＿＿＿＿ から接線を引き，その x 切片 ＿＿ を計算する.
  3. 以後，x_n に対してグラフの接線を考えて，x 切片 ＿＿＿ を求める操作を繰り返す.

ニュートン法のフローチャート（レポート課題）

フローチャートを書くと
理解しやすくなる.



使用する変数

$n, x_n, x_{n+1}, f_{x_n}, f_{x_n}'$

EPS: 収束の判定条件
(例 : 10^{-16})

ニュートン法のプログラミング

- 考え方
  - 微分の計算をユーザ定義関数として追加

(1) 元の関数値を計算するユーザ定義関数: f( )

- $f(x) = 0$ の x を求める「問題となる関数」を設定

（例）: $\sin e^x = 0$ ， $x^3 - 3x - 2 = 0$

(2) 微分係数を計算するユーザ定義関数: f1()

- $f(x)$ の一次導関数 : $f'(x)$

(3) ニュートン法を計算する関数: newton()

- ニュートン法の繰り返し関数

(4) main()

- 初期値設定，printf() で結果表示
- 主な処理（今回の場合，ニュートン法の収束判定処理）

(1) 元の関数値を計算するユーザ定義関数

```
#include <stdio.h> /* printf を利用するのに必要 */
#include <math.h> /* 各種算術関数のために必要 */


/* f(x) = 0 の x を求める問題となる関数 */
double f(double x)        // 64ビット長の浮動小数点数型
{                         ← この変数 x は関数 f() 内でのみ有効
  return sin(exp(x));     // 問題1(a)の関数
}                         →(b), (c)はこの部分を入れ替えて使用
```

(2) 微分係数を計算するユーザ定義関数

```
/* f の一次導関数*/
double f1(double x)
{
  return exp(x) * cos(exp(x)); // 問題1(a)の関数を微分
}
```

この変数 *x* は関数 *f1()* 内でのみ有効

→(b), (c)はこの部分を入れ替えて使用

(3)ニュートン法の繰り返し関数

```
/* ニュートン法の繰り返し関数 */
double newton(double xk)
{
  return xk - (f(xk) / f1(xk));   //  x_k − f_{x_k} / f_{x_k}'
}
```

反復式（算出法を参照）

変数 *xk* を関数 *f()* に代入した
出力結果の値: exp(x) * cos(exp(x))

変数 *xk* を関数 *f()* に代入した
出力結果の値: sin(exp(xk))

(4) main関数

- プログラミング構成
  - (a) 各種初期値設定
  - (b) 開始メッセージを表示（プログラム開始）
  - (c) ニュートン法の繰り返し部分
    - ＜収束の条件判定＞
      - (1) $f(x)$ が許容誤差以下になったら終了
      - (2) $f(x_{k-1})$ と $f(x_k)$ の差がなくなったら終了
      - (3) n が ある回数を越えたら終了，など.
    - →初期値の決め方とその理由を考察
  - (d) x の絶対値を求めて収束条件と比較，出力
  - (e) 終了メッセージを表示（プログラム終了）

(4) main関数のプログラム構成(1/3)

```
main()
{

 /* 各種初期値設定 */
 int n = 0;                          /* 繰り返し回数 */
 double ans = log(3.14159265358979323846);  /* 真の x (本来不明) */
 double delta = 3E-16;       /* 許容誤差 */
                             /* 3E-16 .. 3.0x10の-16乗 */
 double xk    = 1;           /* 初期値 */

 /* 開始メッセージを表示 */
 printf("Newton method program start.¥n");


…
```

(4) main関数のプログラム構成(2/3)

/* ニュートン法の繰り返し部分 */
/* xxx: 本来は n =0 の初期状態から表示すべき
　xxx:「収束の条件判定」には工夫が必要
　(例1) f(x) が delta 以下になったら終了
　(例2) f(xk-1) と f(xk) の差がなくなったら終了
　(例3) n が ある回数を越えたら終了
　等々…
　xxx: 初期値の決め方とその理由
*/

(4) main関数のプログラム構成(3/3)

```
/* fabs(x) … x の絶対値を求める関数*/
while (fabs(f(xk)) > delta){ /* 収束条件 : f(x) が delta 以下 */
  n = n + 1;                    // 繰り返し回数を1増やす
  xk = newton(xk); //ニュートン法の繰り返し関数でxkに出力
  printf("n:%d xk:%f f(xk):%f ans-xk:%f¥n", n, xk, f(xk),
 ans - xk);
}
```

繰り返し回数　現時点の x　sin(exp(xk))（現時点のy）　真の値から x がどれだけ離れているか

```
/* 終了メッセージを表示 */
printf("done.¥n");
}
```

printf 文の表示部分を分かりやすくするのも大事

今日作ったmain関数のプログラム

```
main()
{
 /* 各種初期値設定 */
 int    n     = 0;                        /* 繰り返し回数 */
 double ans   = log(3.14159265358979323846);  /* 真の x (本来は不明) */
 double delta = 3E-16;                    /* 許容誤差. 3E-16 .. 3.0x10の-16乗 */
 double xk    = 1;                        /* 初期値 */

 /* 開始メッセージを表示 */
 printf("Newton method program start¥n");

 /* ニュートン法の繰り返し部分 */
 /* xxx: 本来は n =0 の初期状態から表示すべき
    xxx: 収束の条件判定には工夫が必要
      (1) f(x) が delta 以下になったら終了
      (2) f(xk-1) と f(xk) の差がなくなったら終了
      (3) n が ある回数を越えたら終了
      等々
    xxx: 初期値の決め方とその理由
 */

 /* fabs(x) … x の絶対値を求める関数*/
 while (fabs(f(xk)) > delta){ /* 収束条件 : f(x) が delta 以下になる */
   n = n + 1;
   xk = newton(xk);
   printf("n:%d xk:%f f(xk):%f ans-xk:%f¥n", n, xk, f(xk), ans - xk);
 }

 /* 終了メッセージを表示 */
 printf("done.¥n");
}
```

コンパイル方法

- 通常のコンパイルではうまくいかない.
- 今回のプログラムでは数学ライブラリを利用
  - fabs関数（絶対値を算出）
- 数学ライブラリをリンクさせる必要あり
  - -lmオプション（mathライブラリを使用するオプション）を付与してコンパイル

- コンパイル方法: gcc -lm newton-method.c
  （あなたが作ったプログラム名）