数値解析
Numerical Analysis

数値積分
Numerical Integration

電気･電子情報工学系　　市川　周一
Shuichi Ichikawa, Dept. EEIE

本日の内容
Index

- 数値積分 Numerical Integration
- 区分求積法 rectangular rule
- 台形公式 trapezoidal rule
- シンプソン法 Simpson’s rule

今回の目的
Numerical Integration

- 関数$f(x)$が閉区間$[a,b]$で積分可能であるとき，その定積分の近似値を（数値的に）求める
  - 実際には，まず「積分の近似値を求める公式を求める」
- 前提
  - 不定積分$F(x)$が求まるなら，数値積分は不要！
  - 関数$f(x)$の不定積分$F(x)$が，簡単に求まらない → 数値積分
  - サンプリング結果（離散的な実測値）から積分を計算したい

$$S=\int_a^b f(x)dx=\left[F(x)\right]_a^b=F(b)-F(a)$$

【数学】 積分
Math: Integration

- 以下のような公式は知っているものとする

$$\int x^p dx = \frac{x^{p+1}}{p+1} \quad (p \neq -1)$$

$$\int \frac{1}{x} dx = \int x^{-1} dx = \log x$$

$$\int e^x dx = e^x$$

$$\int \sin x \, dx = -\cos x$$

$$\int \cos x \, dx = \sin x$$

積分定数省略

$$\int \left(\alpha \cdot f(x) + \beta \cdot g(x)\right) dx = \alpha \int f(x) dx + \beta \int g(x) dx$$

$$\int f'(x) g(x) dx = f(x) g(x) - \int f(x) g'(x) dx$$ （部分積分）

$$\int f(x) dx = \int f(g(t)) g'(t) dt$$ （$x = g(t)$とおく 置換積分）

コラム： MATLABって便利です Column: Try MATLAB in IMC

- 式の微分や不定積分もできます.

```
wlinux1:~$ matlab⏎
（省略）
                    < M A T L A B (R) >
            Copyright 1984-2013 The MathWorks, Inc.
             R2013a (8.1.0.604) 64-bit (glnxa64)
                     February 15, 2013
（省略）
>> syms x
>> int(1/(1+x*x),x)

ans =

atan(x)

>>
```

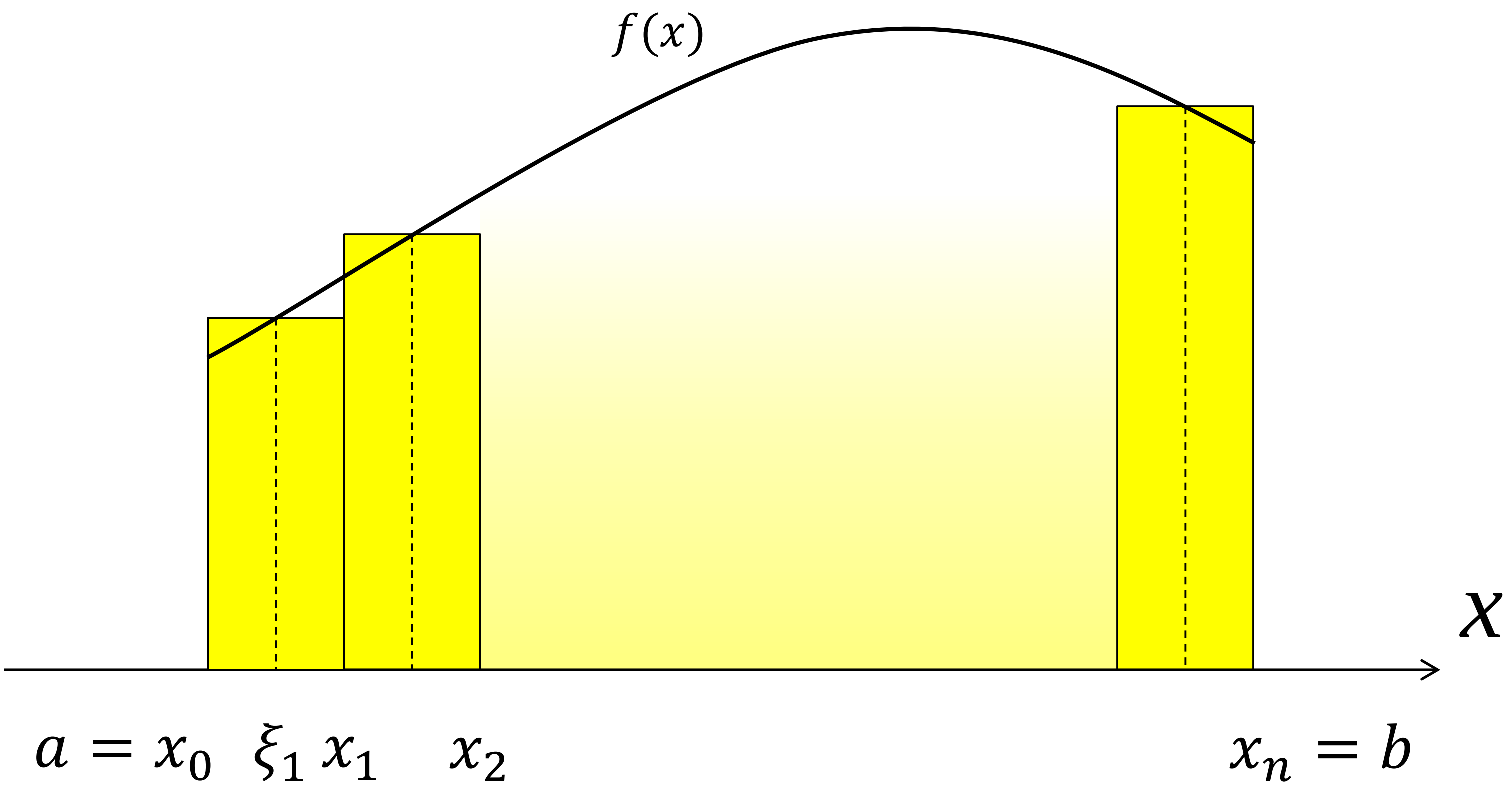
$$\int \frac{1}{1+x^2} dx = \arctan x$$

【数学】 定積分
Math: Definite Integral

- 閉区間$[a, b]$で定義された有界関数$f(x)$を考える.
  - 有界 → $\exists K, |f(x)| < K \ (a \leq x \leq b)$
- 区間$[a, b]$をn個の小区間に分割する
  - $\Delta: a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$
  - ここで, $\delta(\Delta) \equiv \max_j (x_j - x_{j-1})$とする. $(j = 1,2, \ldots, n)$
  - 任意の$\xi_j \ (x_{j-1} < \xi_j < x_j) \ (j = 1,2, \ldots, n)$に対して近似和 $S(\Delta; \xi) = \sum_{j=1}^{n} f(\xi_j)(x_j - x_{j-1})$ をつくる.
- $\delta(\Delta) \to 0$ となるように分割を細かくしたとき, Δとξによらず$S(\Delta; \xi)$が一定値に収束するとき, この値を$f(x)$の$[a, b]$における定積分と呼ぶ.

定積分
Definite Integral

- Sが収束 → $f(x)$とx軸で囲む図形の面積に対応する



数値積分
Numerical Integration

- 数値的に積分したい場合
  - 関数の不定積分が簡単に求まらない
    It is difficult to calculate the indefinite integral
  - 関数の値が離散的にしかわかっていない（例：観測値）
    You have the observed values

区間$[a,b]$をn等分する．

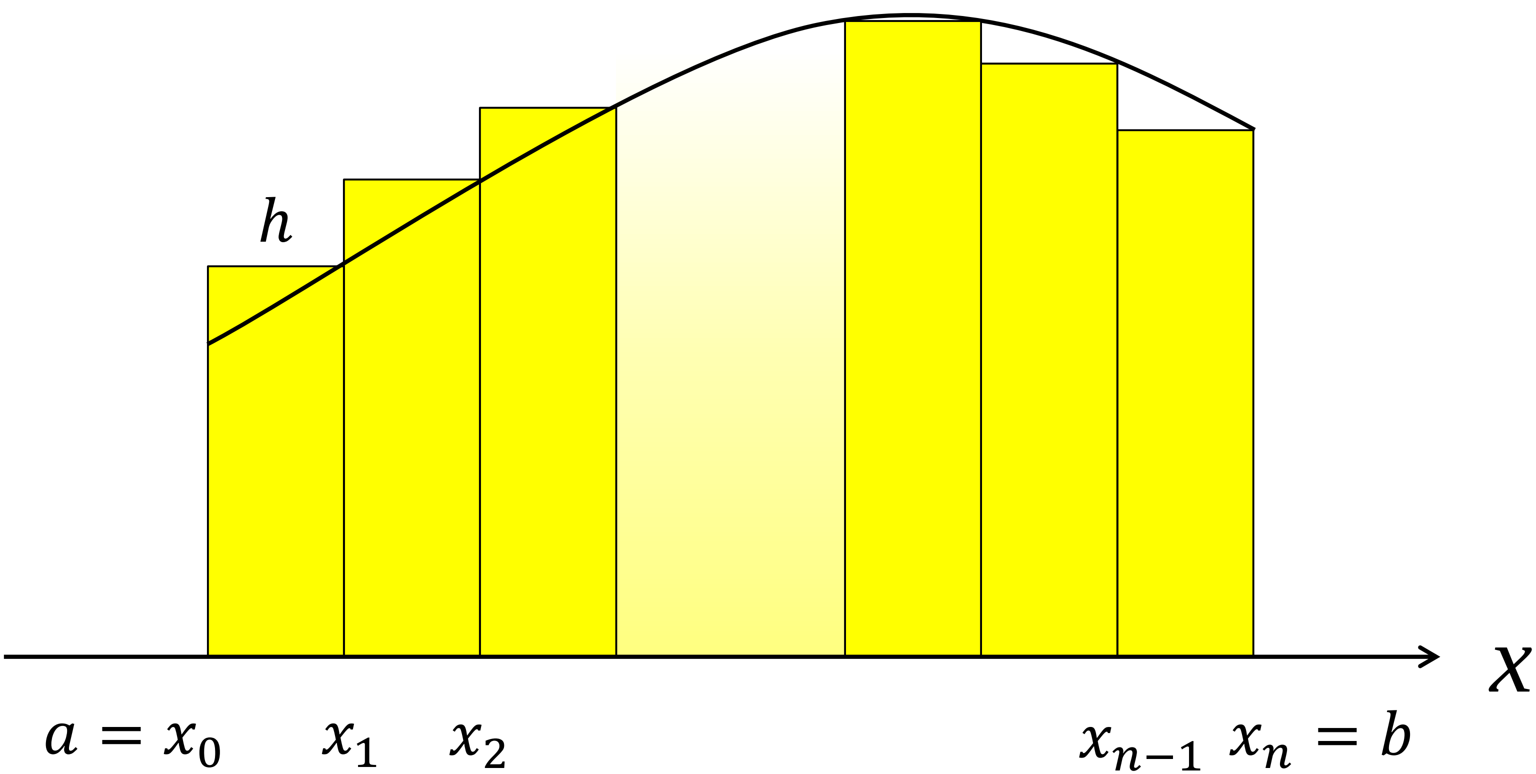
$$h=(b-a)/n,$$

$$x_0=a,\quad x_j=x_0+jh\ (j=0,1,...,n),\quad x_n=b$$

$$\int_a^b f(x)dx \approx h\{A_0 f(x_0)+A_1 f(x_1)+\cdots+A_n f(x_n)\}\quad A_j\text{は定数}$$

区分求積法
Rectangular Rule

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} h\left\{ f(x_1) + \cdots + f(x_n) \right\}$$



プログラム例： 区分求積法
Program: Rectangular Rule

- 仮定： $0 \le x \le 1$ で関数funcを積分する．Stepは区間数.

```
double func(double x);

double RectInt(int step)
{
    double value;
    int i;

    value = 0.0;
    for (i = 1; i <= step; i++) {
        value += func((double)i/step);
    }
    return value/step;
}
```

キャスト演算子

$$x_i = \frac{i}{step}$$

$$h = \frac{1}{step}$$

復習：　整数型と浮動小数点型の演算
Revisited: mixed arithmetic

- 型の異なる変数・定数を混ぜて演算することができる
  - 適切な演算が選択され，それに合わせて「型変換」が起こる
    - 最上位の型に変換してから演算

```
double d;
int i;

d = i;
```

整数 → 実数
に暗黙の型変換

整数の加算

整数	＋	整数	→	整数
整数	＋	実数	→	実数
実数	＋	整数	→	実数
実数	＋	実数	→	実数

浮動小数点数の加算

整数を実数に型変換したのち，
実数＋実数の演算が行われる

復習： 間違いやすい例
Revisited: examples

- 下の4つの演算は，すべて同じではない

```
$ cat t.c
#include <stdio.h>

main()
{
    int i;
    float f, g;

    i = 2;
    f = 11;
    g = 11 / 2;  printf("%e¥n", g);
    g = 11 / i;  printf("%e¥n", g);
    g = f  / 2;  printf("%e¥n", g);
    g = f  / i;  printf("%e¥n", g);
}
```

整数／整数 なので，
整数除算が行われる

実数／整数 なので，実数除算

```
$ cc t.c
$ ./a.out
5.000000e+00
5.000000e+00
5.500000e+00
5.500000e+00
```

復習： 型の変換
Revisited: cast operation

- 演算する前に実数の型に変換してやればよい

```
$ cat t.c
#include <stdio.h>

main()
{
    int i;
    float f, g;

    i = 2;
    f = 11;
    g = 11.0 / 2;   printf("%e¥n", g);
    g = 11.0 / i;   printf("%e¥n", g);
    g = (float)11 / i;   printf("%e¥n", g);
    g = 11 / (float)i;   printf("%e¥n", g);
}
```

キャスト演算子 (型)変数
例 (float)i　整数変数iの値をfloat型に

11 は整数
11.0 は実数

```
$ cc t.c
$ ./a.out
5.500000e+00
5.500000e+00
5.500000e+00
5.500000e+00
```

区分求積法の実行例
Example: rectangular rule

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

$$\int_0^1 f(x)dx = [\arctan(x)]_0^1 = \frac{\pi}{4}$$

```
int main()
{
    int i;
    double rv, uv;

    // real value
    rv = ifunc(0.0, 1.0);
    // check for various steps
    for (i = 10; i <= 1000000; i *= 10) {
        uv = RectInt(i);
        printf("Step = %d¥n", i);
        printf("Value = %+e (err. %+.2e)¥n", uv, (uv/rv)-1);
        printf("¥n");
    }
    return 0;
}
```

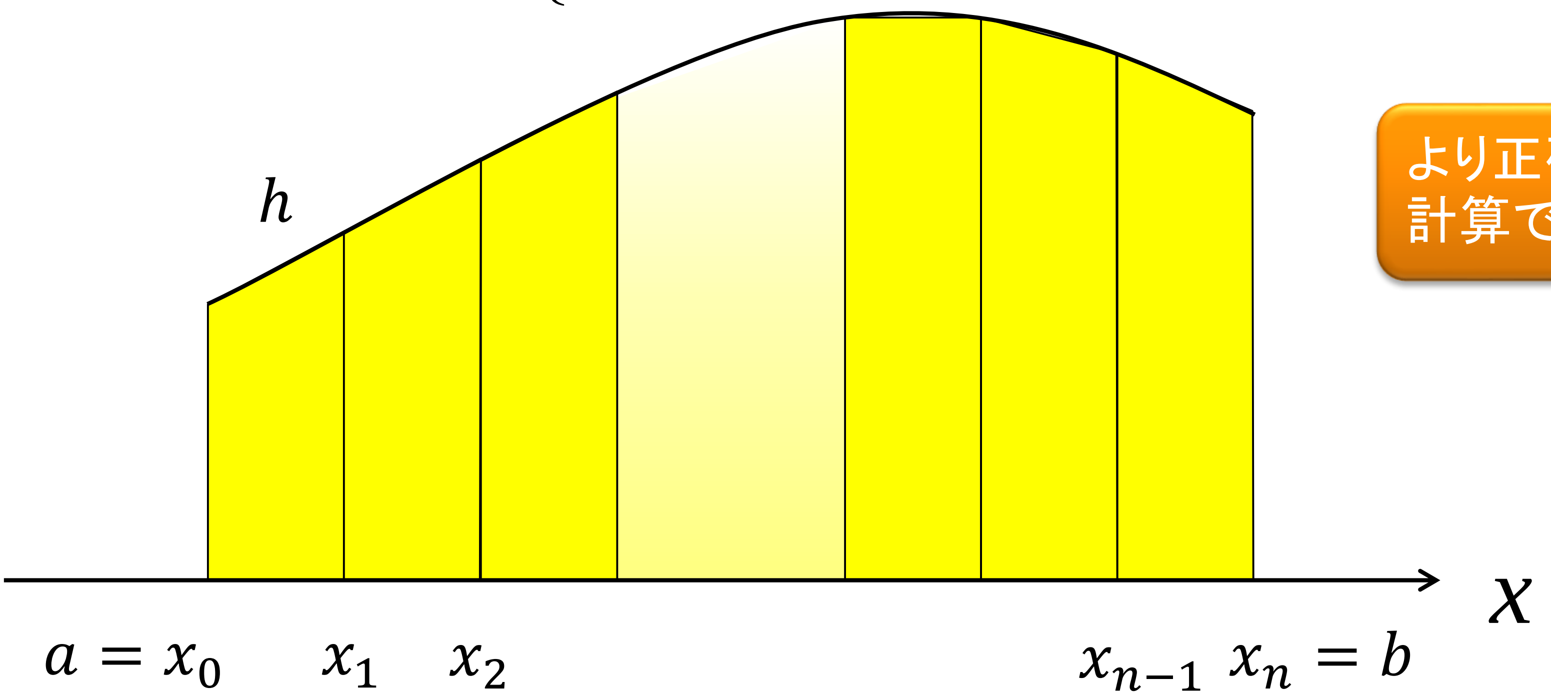
関数ifunc
関数funcの定積分

```
Step = 10
Value = +7.599815e-01 (err. -3.24e-02)
Step = 100
Value = +7.828940e-01 (err. -3.19e-03)
Step = 1000
Value = +7.851481e-01 (err. -3.18e-04)
Step = 10000
Value = +7.853732e-01 (err. -3.18e-05)
Step = 100000
Value = +7.853957e-01 (err. -3.18e-06)
Step = 1000000
Value = +7.853979e-01 (err. -3.18e-07)
```

台形法
Trapezoidal rule

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} \frac{h}{2} \sum_{i=0}^{n-1} \left\{ f(x_i) + f(x_{i+1}) \right\}$$

$$= \lim_{n\to\infty} h \left\{ \frac{1}{2} f(x_0) + f(x_1) + \cdots + f(x_{n-1}) + \frac{1}{2} f(x_n) \right\}$$



より正確に
計算できる

プログラム例： 台形法
Program: Trapezoidal Rule

- 仮定： $0 \le x \le 1$ で関数funcを積分する．Stepは区間数．

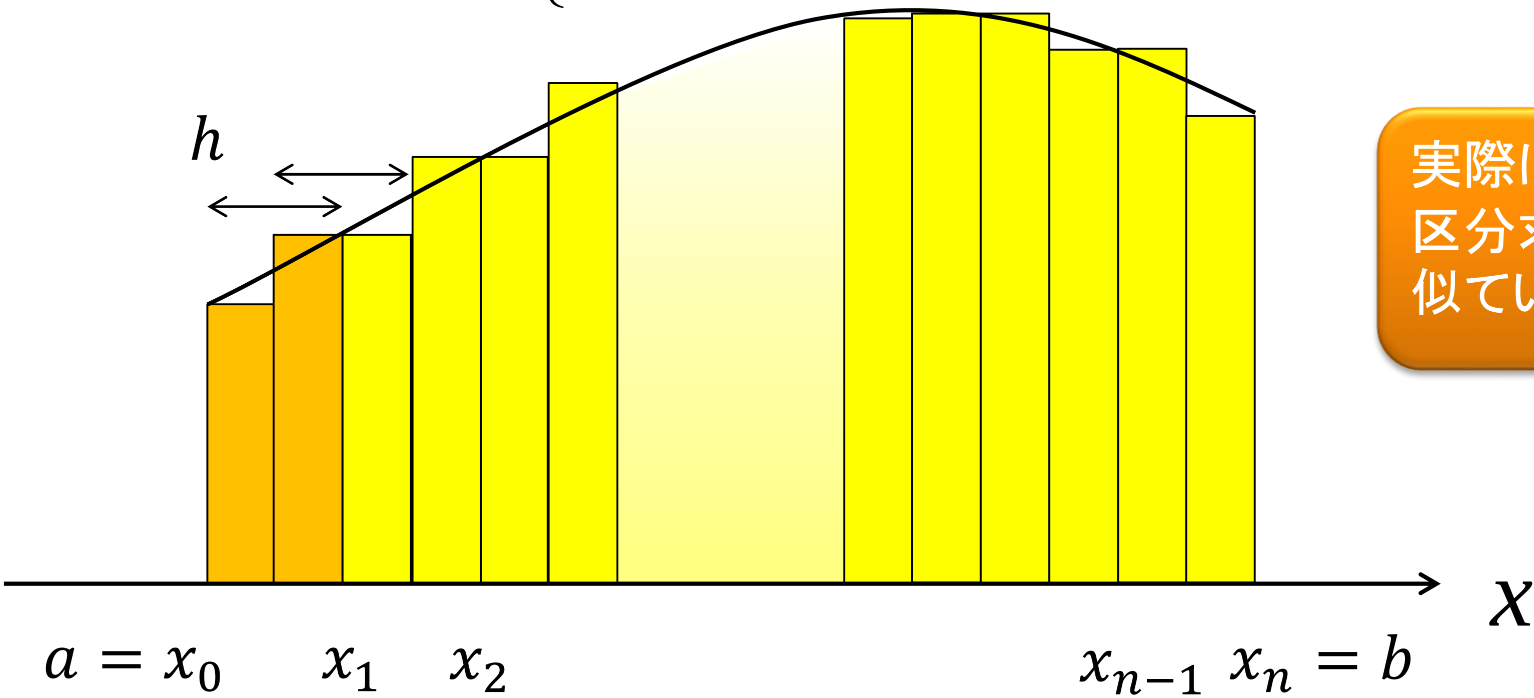
関数宣言
実体は問題に応じて定義

```
double func(double x);

// trapezoid integration
double TrapInt(int step)
{
    double value;
    int i;

    value = 0.0;
    for (i = 0; i < step; i++) {
        value += (func((double)i/step) + func((double)(i+1)/step));
    }
    return value/(2.0*step);
}
```

台形法
Trapezoidal Rule

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} \frac{h}{2} \sum_{i=0}^{n-1} \{ f(x_i) + f(x_{i+1}) \}$$

$$= \lim_{n\to\infty} h \left\{ \frac{1}{2} f(x_0) + f(x_1) + \cdots + f(x_{n-1}) + \frac{1}{2} f(x_n) \right\}$$



実際には
区分求積法と
似ている

プログラム例：　台形法
Program: Trapezoidal Rule

- 仮定：　$0 \le x \le 1$ で関数funcを積分する．Stepは区間数.

関数宣言
実体は問題に応じて定義

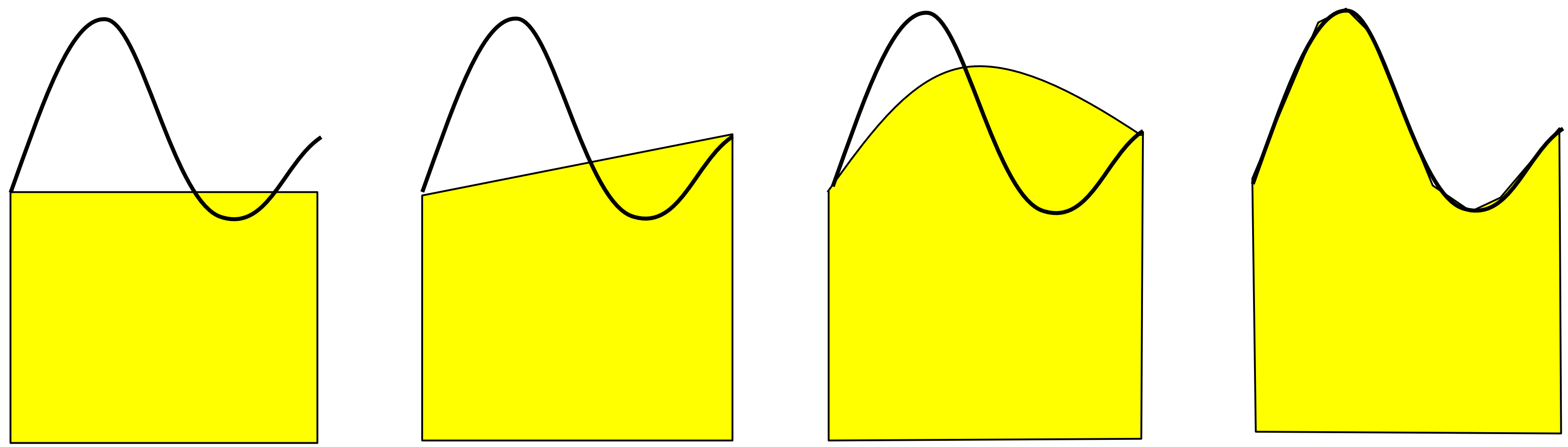
```
double func(double x);

// trapezoid integration
double TrapInt(int step)
{
 double value;
    int i;

    value = 0.5 * func(0.0);
    for (i = 1; i < step; i++) {
        value += func((double)i/step);
    }
    value += 0.5 * func(1.0);
    return value/step;
}
```

$$h = \frac{1}{step}$$

$$+\frac{1}{2} f(x_0)$$
$$+\{f(x_1)+\cdots+f(x_{n-1})\}$$
$$+\frac{1}{2} f(x_n)$$

補間関数で関数を近似する
Approximate functions with interpolation

- 複雑な関数$f(x)$を簡単な関数$g(x)$で置き換えて積分
  - (例えば)多項式で置き換えれば簡単に積分できる
    - Typically, g(x) is a polynomial.
  - 閉区間全体は無理でも，各小区間で近似することは可能



ラグランジュの補間公式 Lagrange Interpolation Formula

- ラグランジュ(J.L.Lagrange)による補間
  - $L_i(x)$はn次式で，$L_i(x_i) = 1.\ x_j\,(i \neq j)$ のとき$L_i(x_j) = 0.$
  - $L_i(x)$を重ね合わせることで，求める補間式を得る．

n次多項式　$p_n(x_j) = f_j \quad (j = 0, 1, ..., n)$

は下のように与えられる．

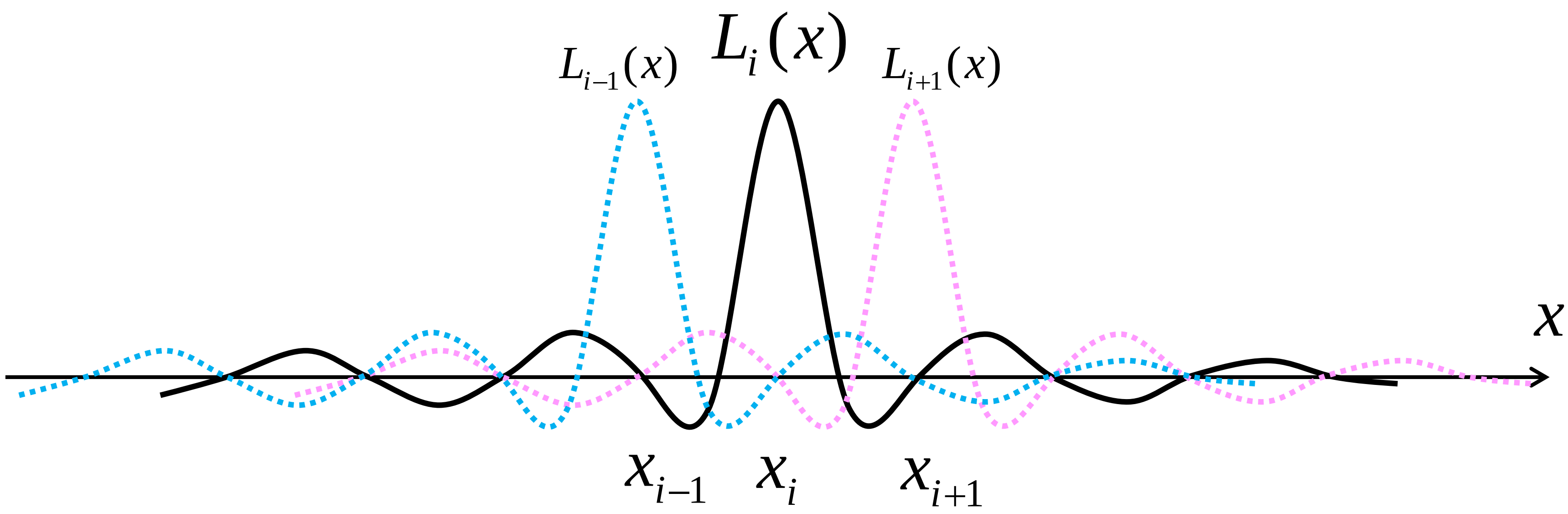
$$L_i(x) = \frac{(x - x_0)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_n)}{(x_i - x_0)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)}$$

$$p_n(x) = \sum_{i=0}^{n} L_i(x) f_i$$

ラグランジュ補間
Lagrange Interpolation

- $L_i(x)$はn次式で，$L_i(x_i) = 1.$ $x_j\,(i \neq j)$ のとき$L_i(x_j) = 0.$

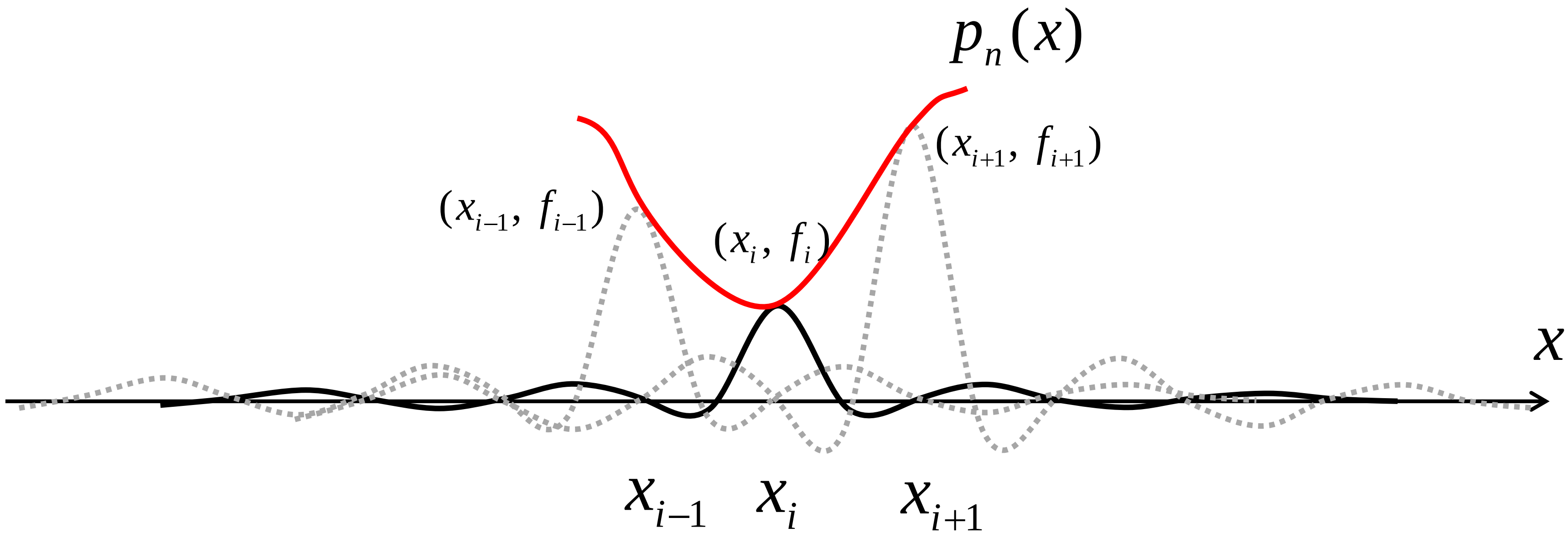
$$\boxed{L_i(x) = \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}}$$



ラグランジュ補間
Lagrange Interpolation

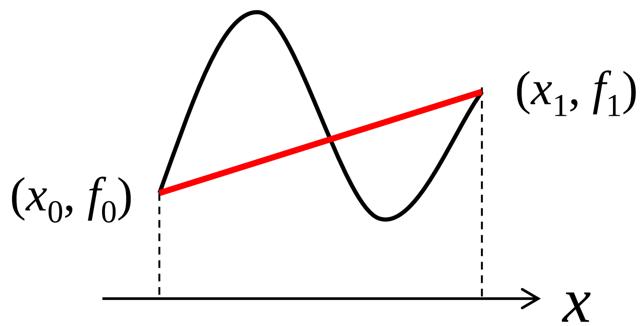
- $L_i(x)$を重ね合わせることで，求める補間式を得る．

$$p_n(x) = \sum_{i=0}^{n} L_i(x) f_i$$



例：　１次と2次のラグランジュ補間
Example: 1st order and 2nd order

- １次のラグランジュ補間関数とは，２点(x_0, f_0), (x_1, f_1)を通る直線
- ２次のラグランジュ補間関数とは，３点(x_0, f_0), (x_1, f_1) , (x_2, f_2)を通る二次関数



$$p_1(x) = \frac{x - x_1}{x_0 - x_1} f_0 + \frac{x - x_0}{x_1 - x_0} f_1$$

$$p_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f_0 + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f_1 + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f_2$$

ニュートン・コーツ(Newton–Cotes)の公式
Newton–Cotes Quadrature Formula

$$\int_a^b f(x)dx \approx \int_a^b p_n(x)dx$$

$$= \int_a^b \left(\sum_{j=0}^{n} L_j(x) f(x_j) \right) dx$$

$$= \sum_{j=0}^{n} \left(\int_a^b L_j(x)dx \right) f(x_j)$$

$$\int_a^b f(x)dx \approx h \sum_{j=0}^{n} A_j f(x_j)$$

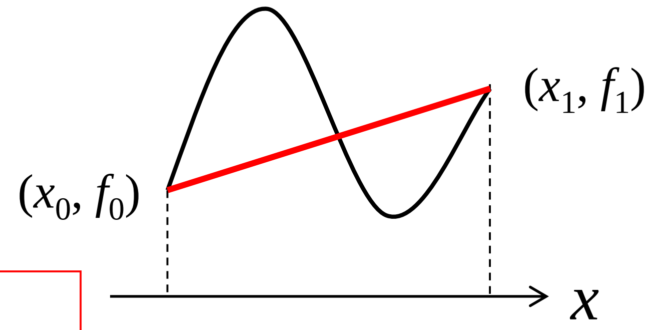
$L_j(x)$は n次関数

次数nが与えられれば，
事前に計算できる

事前に計算した係数

例：　1次のNewton-Cotes公式 Example: 1st order

- 台形公式になる！
  Trapezoidal rule



$$p_1(x) = \frac{x - x_1}{x_0 - x_1} f_0 + \frac{x - x_0}{x_1 - x_0} f_1$$

$$\int_{x_0}^{x_1} f(x)dx \approx \int_{x_0}^{x_1} p_n(x)dx$$

$$= \left[\frac{f_0}{x_0 - x_1} \frac{(x - x_1)^2}{2} \right]_{x_0}^{x_1} + \left[\frac{f_1}{x_1 - x_0} \frac{(x - x_0)^2}{2} \right]_{x_0}^{x_1}$$

$$= \frac{1}{2}(f_0 + f_1)(x_1 - x_0)$$

台形の面積

例：　シンプソンの公式
Example: Simpson’s Rule

- ２次のNewton-Cotes公式

$$\int_a^b f(x)dx \approx h\left(A_0 f_0 + A_1 f_1 + A_2 f_2\right)$$

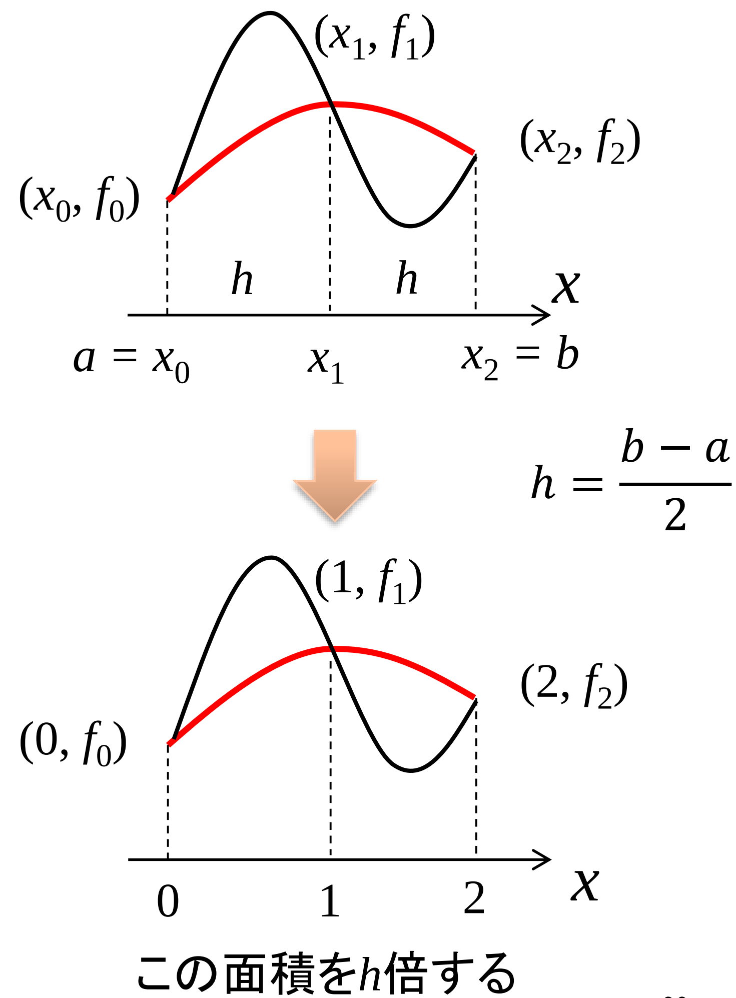
$$A_0 = \int_0^2 \frac{(u-1)(u-2)}{2}du = \frac{1}{3}$$

$$A_1 = \int_0^2 u(2-u)du = \frac{4}{3}$$

$$A_2 = \int_0^2 \frac{u(u-1)}{2}du = \frac{1}{3}$$

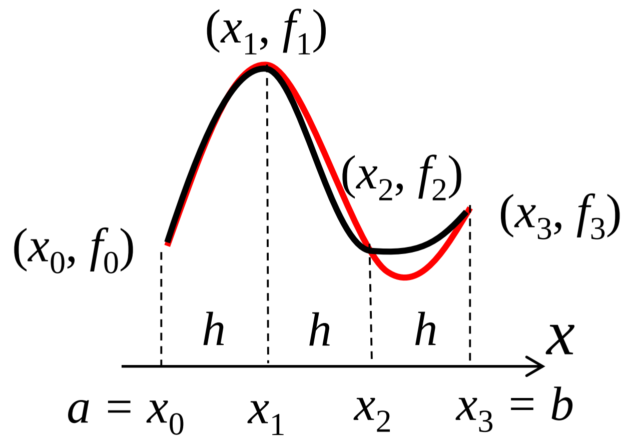
$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f_0 + 4f_1 + f_2\right)$$

$$\approx \frac{(b-a)}{6}\left(f_0 + 4f_1 + f_2\right)$$



3次のNewton−Cotes
3^{rd} order

- シンプソンの3/8公式
  Simpson's 3/8 Rule



$$\int_a^b f(x)dx \approx h\left(A_0 f_0 + A_1 f_1 + A_2 f_2 + A_3 f_3\right)$$

$$\int_a^b f(x)dx \approx \frac{(b-a)}{8}\left(f_0 + 3f_1 + 3f_2 + f_3\right)$$

プログラム例： シンプソン法
Program: Simpson’s Rule

- 仮定： $0 \le x \le 1$ で関数funcを積分する．Stepは区間数.

```
double SimpInt(int step)
{
    double value;
    int i;

    value = 0.0;
    for (i = 0; i < step; i++) {
        value += (func((double)i/step)
                 + 4.0*func((i+0.5)/step)
                 + func((i+1.0)/step));
    }
    return value/(6.0*step);
}
```

実行例
Example

```
int main()
{
    int i;
    double rv, sv, tv, uv;

    // real value
    rv = ifunc(0.0, 1.0);
    // check for various steps
    for (i = 10; i <= 1000000; i *= 10) {
        sv = RectInt(i);
        tv = TrapInt(i);
        uv = SimpInt(i);
        printf("Step = %d¥n", i);
        printf("Rectangle = %+e (err. %+.2e)¥n", sv, (sv/rv)-1);
        printf("Trapezoid = %+e (err. %+.2e)¥n", tv, (tv/rv)-1);
        printf("Simpson's = %+e (err. %+.2e)¥n", uv, (uv/rv)-1);
        printf("¥n");
    }

    return 0;
}
```

定積分を計算する
arctan(1.0)-arctan(0.0)

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

$$\int_0^1 f(x)dx = [\arctan(x)]_0^1 = \frac{\pi}{4}$$

原始関数の求め方

- ここでは，数値積分の誤差を評価するため，原始関数が簡単に求まるような例を用いている．
- 実用上は，原始関数が求まるなら数値積分する必要はない！

$$\int \frac{1}{1+x^2}dx = \int \frac{1}{1+\tan^2\theta}(\tan\theta)' \, d\theta$$

$$\frac{1}{1+\tan^2\theta} = \frac{\cos^2\theta}{\cos^2\theta+\sin^2\theta} = \cos^2\theta$$

$$\frac{d}{d\theta}\tan\theta = \frac{d}{d\theta}\frac{\sin\theta}{\cos\theta} = \frac{\cos\theta}{\cos\theta} + \frac{\sin^2\theta}{\cos^2\theta} = \frac{1}{\cos^2\theta}$$

$$\int \frac{1}{1+x^2}dx = \int d\theta = \theta = \arctan x$$

実行例 Example

- この3種の比較では，シンプソン法が一番精度が高い
- シンプソン法であればステップ数は小さくて良い
  - シンプソン法は計算が速い
- ステップを細かくしすぎると，計算誤差の累積で有効数字が減少している

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

$$\int_0^1 f(x)dx = [\arctan(x)]_0^1 = \frac{\pi}{4}$$

```
Step = 10
Rectangle = +7.599815e-01 (err. -3.24e-02)
Trapezoid = +7.849815e-01 (err. -5.31e-04)
Simpson's = +7.853982e-01 (err. -1.97e-10)

Step = 100
Rectangle = +7.828940e-01 (err. -3.19e-03)
Trapezoid = +7.853940e-01 (err. -5.31e-06)
Simpson's = +7.853982e-01 (err. +2.22e-16)

Step = 1000
Rectangle = +7.851481e-01 (err. -3.18e-04)
Trapezoid = +7.853981e-01 (err. -5.31e-08)
Simpson's = +7.853982e-01 (err. +1.33e-15)

Step = 10000
Rectangle = +7.853732e-01 (err. -3.18e-05)
Trapezoid = +7.853982e-01 (err. -5.31e-10)
Simpson's = +7.853982e-01 (err. +4.44e-16)

Step = 100000
Rectangle = +7.853957e-01 (err. -3.18e-06)
Trapezoid = +7.853982e-01 (err. -5.30e-12)
Simpson's = +7.853982e-01 (err. -1.75e-14)

Step = 1000000
Rectangle = +7.853979e-01 (err. -3.18e-07)
Trapezoid = +7.853982e-01 (err. -8.70e-14)
Simpson's = +7.853982e-01 (err. -1.90e-14)
```