# Simpson's rule.

## stage 1：Simpson則とは。

名前ぐらいは聞いた事もあろう、今回はSimpson則を紹介する。点 $(x_n, f(x_n))$ と点 $(x_n + \Delta x, f(x_n + \Delta x))$ の2点を直線で結び、その直線とx軸とが囲む台形領域を積分範囲に渡って足し合わせるのが台形則であった。それに対してSimpson則は3点を通るような2次関数を用いる。積分したい関数の小部分を直線 $y = ax + b$ で近似するのと曲線 $y = px^2 + qx + r$ で近似するのではどちらが精度よく積分が求まるのか、それを見ていこう。
積分範囲を $[a, b]$ として、その区間を $n$ 等分する。そのときの曲線上の分割点を $P_0(x_0, y_0)$, $P_1(x_1, y_1)$, $P_2(x_2, y_2), \cdots$, $P_{n-1}(x_{n-1}, y_{n-1})$, $P_n(x_n, y_n)$ とする。ここで $x_0 = a, x_n = b$ であり、**$n$ は0より大きい偶数でなければならない。**その理由は簡単で、そうしなければ $(P_0P_1P_2), (P_2P_3P_4), \cdots, (P_{n-2}P_{n-1}P_n)$ のように3点で1つの組を作れない、あぶれモノが出てきてしまうからである。これらの組を用いて、**曲線を $n/2$ 個の放物線の弧で近似して積分しようというのがSimpson則**である。すなわち $(P_0P_1P_2), (P_2P_3P_4), \cdots, (P_{n-2}P_{n-1}P_n)$ をそれぞれ $y$ 軸に平行な軸を持つ放物線の一部と見て、これらと $x$ 軸との間の面積を順に $S_1$, $S_2, \cdots$, $S_{n/2}$ とすると、求める面積を

$$S \simeq \sum_{i=1}^{n/2} S_i = S_1 + S_2 + \cdots + S_{n/2} \tag{1}$$

と近似するものである。

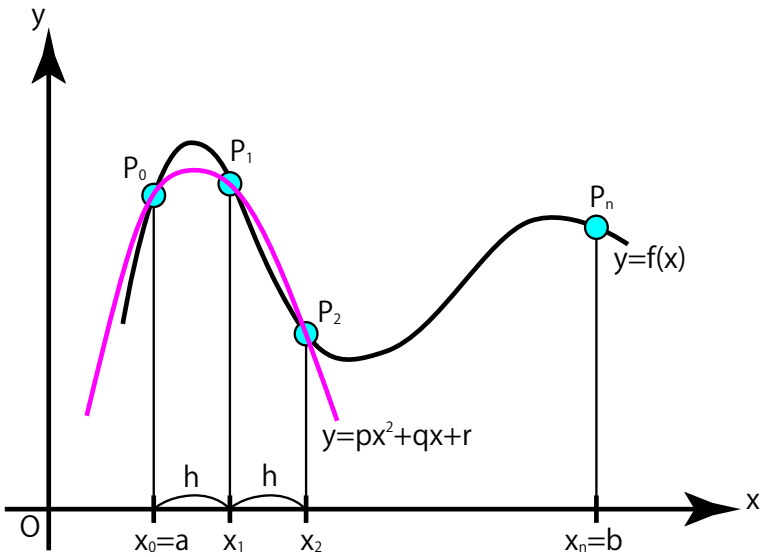


図 1: Simpson則の概念図。積分区間 [a, b] を偶数 n で等分する。添字順に3点で1組を作り、その3点を通るような放物線で積分計算を近似する。

## stage 2：公式の導出。

まずは $S_1$ を求めよう。点 $P_0$, $P_1$, $P_2$ を通る放物線の式を $y = px^2 + qx + r$ とする。ポイントは、計算が簡単になるよう $x_1 = 0$ に平行移動することである。分割幅を $h$ とすると、面積は平行移動しても変わらないので、

$$S_1 = \int_{x_0}^{x_2} y \, dx = \int_{-h}^{h} (px^2 + qx + r) \, dx = \left[ \frac{1}{3}px^3 + \frac{1}{2}qx^2 + rx \right]_{-h}^{h} = \frac{h}{3}(2ph^2 + 6r) \tag{2}$$

のように書ける。この放物線は $P_0(-h, y_0),\ P_1(0, y_1),\ P_2(h, y_2)$ を通るので

$$\begin{cases} y_0 = ph^2 - qh + r \\ 4y_1 = 4r \\ y_2 = ph^2 + qh + r \end{cases} \implies y_0 + 4y_1 + y_2 = 2ph^2 + 6r \tag{3}$$

$$\therefore (2), (3) \implies S_1 = \frac{h}{3}(y_0 + 4y_1 + y_2) \tag{4}$$

$S_2,\ S_3 \cdots S_{n/2}$ に関しても同様に

$$\begin{aligned} S_2 &= \frac{h}{3}(y_2 + 4y_3 + y_4), \\ S_3 &= \frac{h}{3}(y_4 + 4y_5 + y_6), \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ S_{n/2} &= \frac{h}{3}(y_{n-2} + 4y_{n-1} + y_n) \end{aligned} \tag{5}$$

と求まる。これらを (1) 式に代入することにより

$$S \simeq \frac{h}{3}(y_0 + y_n + 4\underbrace{(y_1 + y_3 + \cdots + y_{n-1})}_{\text{奇数項}\frac{n-2}{2}+1\text{ 個の和}} + 2\underbrace{(y_2 + y_4 + \cdots + y_{n-2})}_{\text{偶数項}\frac{n-2}{2}\text{個の和}}) \tag{6}$$

と求まる。この形にまで持ってくれば、プログラムを書くのは非常に簡単である。

# stage 3 : 誤差の見積もり。

点 $(P_0P_1P_2)$ を通る放物線が作る面積 (4) 式と、$x_0 \sim x_2$ までの真の積分値との誤差を見積もろう。この計算は頭の劣化した私には少々苦しい。しかしやる。

$$\varepsilon(S_1) \equiv \frac{h}{3}(y_0 + 4y_1 + y_2) - \int_{x_0}^{x_2} f(x)\ dx \tag{7}$$

を考える。$y_0 = f(x_0),\ y_1 = f(x_1) = f(x_0 + h),\ y_2 = f(x_2) = f(x_0 + 2h)$ より

$$\begin{aligned} \text{(第一項)} &= \frac{h}{3}(y_0 + 4y_1 + y_2) = \frac{h}{3}\left( f(x_0) + 4\sum_{n=0}^{\infty} \frac{h^n}{n!} \left.\frac{d^n f}{dx^n}\right|_{x_0} + \sum_{n=0}^{\infty} \frac{(2h)^n}{n!} \left.\frac{d^n f}{dx^n}\right|_{x_0} \right) \\ &= 2hf(x_0) + \sum_{n=1}^{\infty} \frac{h^{n+1}}{n!} \frac{4 + 2^n}{3} \left.\frac{d^n f}{dx^n}\right|_{x_0} \end{aligned} \tag{8}$$

$$\text{(第二項)} = \int_{x_0}^{x_0+2h} f(x)\ dx = \int_{x_0}^{x_0+h} f(x)\ dx + \int_{x_0+h}^{x_0+2h} f(x)\ dx$$

$$
\begin{aligned}
&\underbrace{=}_{\text{Taylor 展開}} && \int_{x_0}^{x_0+h} \sum_{n=0}^{\infty} \frac{(x-x_0)^n}{n!} \left.\frac{d^n f}{dx^n}\right|_{x_0} dx + \int_{x_0+h}^{x_0+2h} \sum_{n=0}^{\infty} \frac{(x-(x_0+h))^n}{n!} \left.\frac{d^n f}{dx^n}\right|_{x_0+h} dx \\
&\underbrace{=}_{\text{変数変換}} && \int_0^h \sum_{n=0}^{\infty} \frac{\alpha^n}{n!} \left.\frac{d^n f}{dx^n}\right|_{x_0} d\alpha + \int_0^h \sum_{n=0}^{\infty} \frac{\beta^n}{n!} \left.\frac{d^n f}{dx^n}\right|_{x_0+h} d\beta \\
&= && \sum_{n=0}^{\infty} \frac{1}{n!}\frac{h^{n+1}}{n+1} \left.\frac{d^n f}{dx^n}\right|_{x_0} + \sum_{n=0}^{\infty} \frac{1}{n!}\frac{h^{n+1}}{n+1} \left.\frac{d^n f}{dx^n}\right|_{x_0+h}
\end{aligned} \tag{9}
$$

誤差計算の常套手段、Taylor 展開を使用して計算してきたが、最後の微係数をさらに Taylor 展開する。

$$
\frac{d^n}{dx^n} f(x_0+h) = \frac{d^n}{dx^n} \left( \sum_{m=0}^{\infty} \frac{h^m}{m!} \frac{d^m f}{dx^m} \right)_{x_0} \tag{10}
$$

$$
(9) = \sum_{n=0}^{\infty} \frac{1}{n!}\frac{h^{n+1}}{n+1} \left.\frac{d^n f}{dx^n}\right|_{x_0} + \sum_{n=0}^{\infty} \frac{1}{n!}\frac{h^{n+1}}{n+1} \frac{d^n}{dx^n} \left( \sum_{m=0}^{\infty} \frac{h^m}{m!} \frac{d^m f}{dx^m} \right)_{x_0} \tag{11}
$$

$$
\begin{aligned}
\therefore (7),(8),(11) \implies \varepsilon(S_1) \quad &= \quad 2hf(x_0) + \sum_{n=1}^{\infty} \frac{h^{n+1}}{n!} \frac{4+2^n}{3} \left.\frac{d^n f}{dx^n}\right|_{x_0} \\
&\quad - \left[ \sum_{n=0}^{\infty} \frac{1}{n!}\frac{h^{n+1}}{n+1} \left.\frac{d^n f}{dx^n}\right|_{x_0} + \sum_{n=0}^{\infty} \frac{1}{n!}\frac{h^{n+1}}{n+1} \frac{d^n}{dx^n} \left( \sum_{m=0}^{\infty} \frac{h^m}{m!} \frac{d^m f}{dx^m} \right)_{x_0} \right]
\end{aligned} \tag{12}
$$

これをイッペンに計算することは私は不可能だったので、$h$ の次数の比較から攻めて行く。すると

$$
\varepsilon(S_1) = \frac{h^5}{90} \frac{d^4 f}{dx^4}(x_0) + O(h^6) \tag{13}
$$

と求まる。これは $S_1$ のみの誤差評価であり、実際に有限区間の積分を実行したときの誤差は大雑把に考えて $n/2$ 倍と考えてよいだろう (今積分区間は $2h$)。

$$
\varepsilon \simeq \frac{h^4}{180} \frac{d^4 f}{dx^4}(\alpha) \ (a \le \alpha \le b) \tag{14}
$$

$h = (b-a)/n$ なので、誤差は $n^{-4}$ に比例している。すなわち積分の精度は $n^{-3}$ に比例することがわかる。

# Program example in C & etc...

## stage 1 : compile method.

Simpson フォルダを適当な場所に置いて、そのフォルダをカレントディレクトリにします。デスクトップにこのフォルダがある場合には以下のようにコマンドを打ちます。

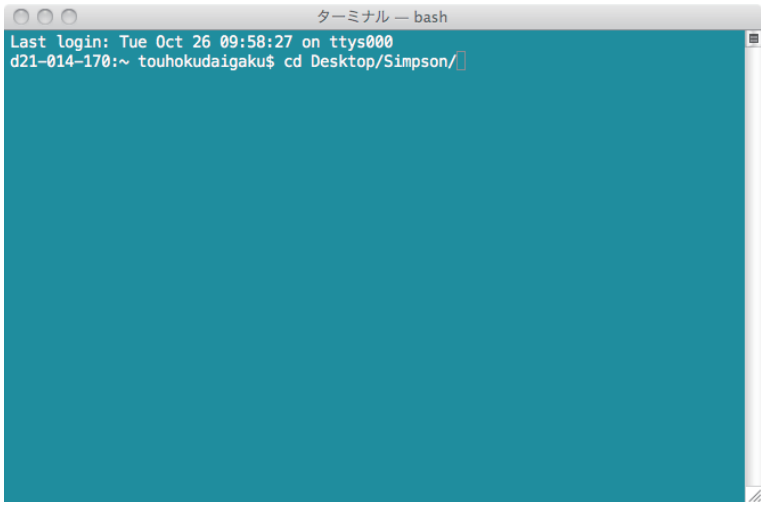


図 2: コマンド。ちなみに Linux では”端末”と名付けられたこのウィンドウは Mac では”ターミナル”である。

Simpson フォルダの中には answer.h, common.h, integrand.h, main.c の 4 つのファイルが入っています。

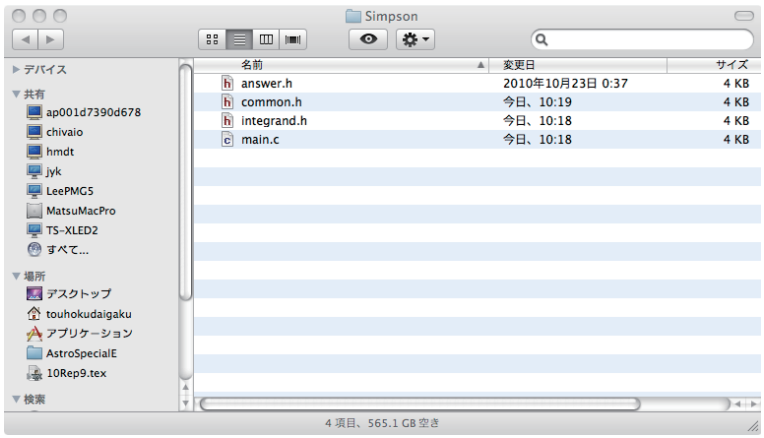


図 3: Mac の洗練されたファイル画面。ここに掲載したのはただの自己満足か？

それぞれどのような構造になっているかは各自で確認するとして、ここではコンパイル方法とこのプログラムの使い方のみご紹介します。
コンパイルには gcc を使います。とはいえ、gcc があらかじめ入っている Linux 等をすでにお持ちの皆さんであれば、以下の様に打ち込んで Enter を押すだけです。C 言語の場合は Fortran と違って、main 関数のファイルのみをコンパイルすればそれで OK です。ちなみに-o はコンパイルオプションで、コンパイル後に生成される実行ファイルの名前を変更することができるオプションです。このオプションをつけなければ実行ファイルは皆さんおなじみの a.out になります。ここでは

simpson としておきました。

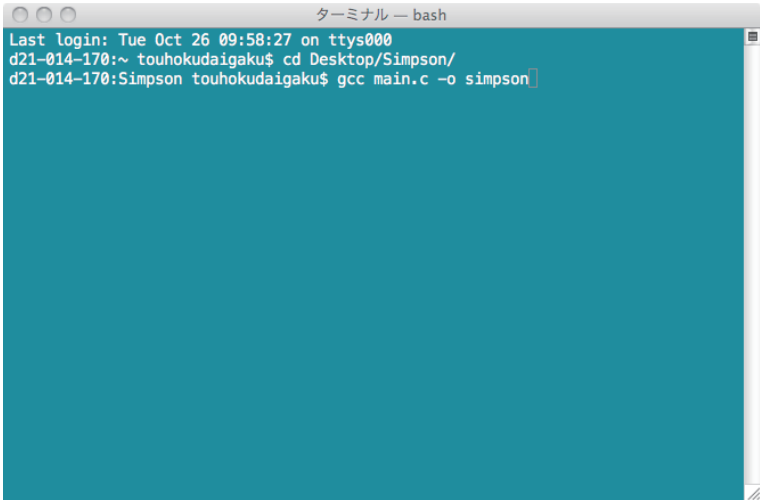


図 4: gcc コンパイル。疲労困憊る。

ひょっとしたらコンパイルが通らないよぉ〜、という方もいらっしゃるかもしれまんせん。その場合にはコンパイルオプションに-lm を足してみてください。これはこのプログラムのなかで ⟨math.h⟩ というのを使っているためで、このコンパイルオプションがないせいで ⟨math.h⟩ を認識しないようになっているかもしれないからです。
./simpson と打ち込んで実行してみましょう。

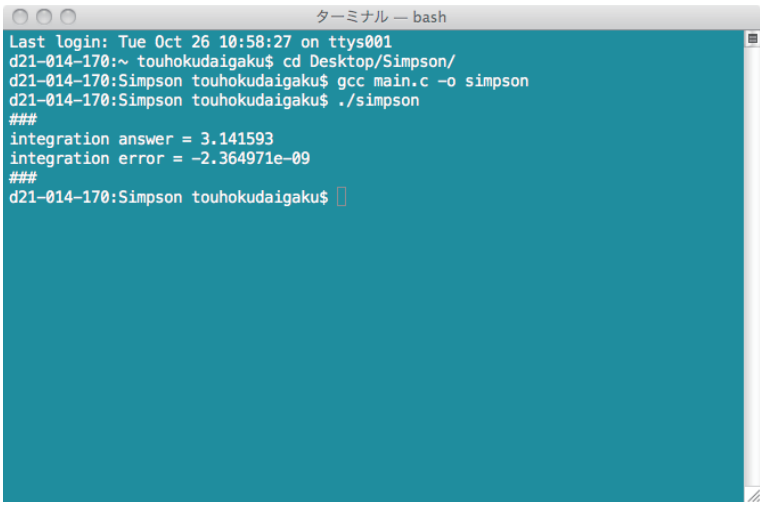


図 5: 実行。

次の行に積分を実行した結果と真値からの差 (error) が表示されています。今計算したのは

$$\int_0^1 \frac{4}{1+x\times x}dx = \pi$$

という、Simpson 公式の例題としてはメジャーなものです。

## stage 2 : grid number change.

せっかく error の誤差の評価も行ったので、積分の分割数を変更してみたいと思った方がいらっしゃるかもしれません。その場合には common.h を開いてください。

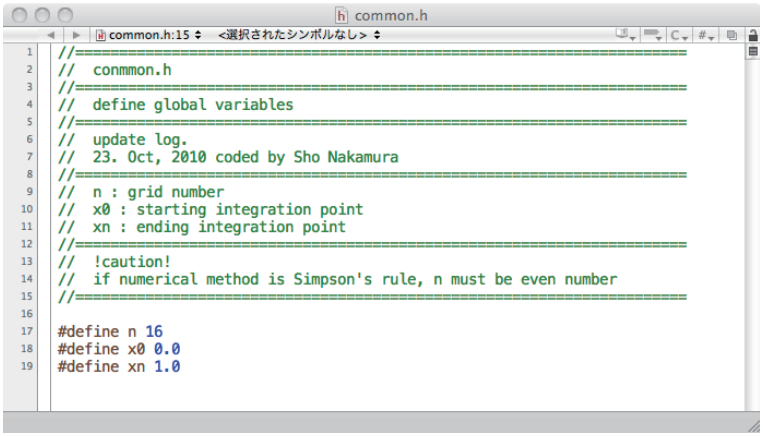


図 6: common.h の中身。プログラムオタクの間では common.h に global 変数をひとまとめに書いてしまうのが通例のようである。とすると、common の由来は Fortran の common 文から来たものだろう。

上から順に分割数、積分下限値、積分上限値が#define で書かれています。分割数を適当に変更し、保存して先ほどと同様にコンパイルして実行してみましょう。本当は error が分割数に対してどのような振る舞いをするか、グラフにできれば良かったのですが、時間が足りず職務怠慢で終わってしまいました。

## stage 3 : integrand change.

被積分関数 (英語で integrand と呼ぶそうです) を変えることもできます。このプログラムでは 4 種類選べるようにしました。

1. $\int_0^1 4\sqrt{1-x\times x}dx$　　2. $\int_0^1 \frac{4}{1+x\times x}dx$　　3. $\int_0^1 8\times x\times x\sqrt{2-x\times x}dx$　　4. $\int_0^1 8x^2\sqrt{2-x^2}dx$

答えはいずれも $\pi$ ですが、数値的に積分計算すると驚くべき違いが出てきます。これらを切り替えるには integrand.h の先頭に書かれている sw の値を変更します。1 ～ 4 の好きなものを確かめてみてください。

## stage $\infty$ : hint.

· 1, 2, 3 は積分した結果は同じですが、被積分関数が違います。グラフを手書きでも構いませんので書いてみてください。致命的に違う点があるはずです。その相違点が数値計算誤差に大きく反映してきます。実は、1, 3 は同じ関数を変数変換して書き換えただけです。
· 3, 4 は被積分関数は同じですが、プログラムの表記の仕方が違います。3 は $x$ の 2 乗を $x\times x$、4 は pow$(x, 2.0)$ としています。ちょっとの違いですが、分割数が大きくなると (すなわち繰り返し計算の回数が大きくなると) 違いが見えてきます。ここでいう違いとは精度の問題だけでないことに注意して考察してみてください。

# 参考文献

·C 言語によるはじめてのアルゴリズム入門 (河西朝雄)
·NUMERICAL RECIPES in C (和訳:丹慶勝市、奥村晴彦、佐藤俊郎、小林誠)
· 詳解 FORTRAN 演習 (中村明子、伊藤文子)
· 数値計算の常識 (伊里正夫、藤野和建)