レポート課題 part 1 についてコメント

桂田 祐史

2016 年 7 月 6 日, 2016 年 7 月 7 日

別件で忙しくて、まだ、あまりレポートを読んでいません。何回か質問されたので、その辺を中心にとりあえず、ざっと、説明します。

課題 1

$$I = \int_0^1 f(x)\,dx, \quad f(x) = -\log\log\frac{1}{x} \quad (0 < x < 1).$$

$f(x)$ は $x = 0, 1$ で定義されず、

$$\lim_{x\to+0} f(x) = \infty, \quad \lim_{x\to1-0} f(x) = -\infty$$

である。従って、I は広義積分である。

Simpson 公式のような古典的な公式ではうまく計算出来ないことが想像できるので、DE 公式を使う。ただし積分区間が $(0, 1)$ なので、講義で主に説明した $\displaystyle\int_{-1}^{1} f(x)\,dx$ に対する公式をそのままでは使うことが出来ない。授業で説明したような変数変換を導入する。

I の値は Euler 定数 γ であるが、Euler 定数 γ は有名なので、ネットでも調べることが出来る。Mathematica で 20 桁の近似値を得るには

```
N[EulerGamma, 20]
```

(試しに WolframAlpha[1] でも実行できた。)

実行結果は次のようになった。

[1] https://www.wolframalpha.com/

```
$ gcc kadai1-1.c
$ ./a.out
test3 -log(log(1/x)) の積分)
h=1.000000, I_h=       0.5641031332883520, I_h-I=-1.311253e-02
h=0.500000, I_h=       0.5772109266922210, I_h-I=-4.738209e-06
h=0.250000, I_h=       0.5772156649014739, I_h-I=-5.895284e-14
h=0.125000, I_h=       0.5772156649014853, I_h-I=-4.751755e-14
h=0.062500, I_h=       0.5772156649013301, I_h-I=-2.027267e-13
h=0.031250, I_h=       0.5772156649011624, I_h-I=-3.704814e-13
h=0.015625, I_h=       0.5772156649010470, I_h-I=-4.858336e-13
h=0.007812, I_h=       0.5772156649009814, I_h-I=-5.514478e-13
h=0.003906, I_h=       0.5772156649009462, I_h-I=-5.866418e-13
h=0.001953, I_h=       0.5772156649009298, I_h-I=-6.030731e-13
$
```

$h = 1/4$ の段階で $|I_h - I| \fallingdotseq 6 \times 10^{-14}$ となっている。

example6.c で、$I_1 = \displaystyle\int_{-1}^{1} \sqrt{1-x^2}dx$ の誤差が 2×10^{-16}, $I_2 = \displaystyle\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}}dx$ の誤差が 2×10^{-8} であった。

I_1 ほど精度が高くないのは、広義積分であるからと推測出来る。

I_2 よりは精度が高いのは、$f(x)$ の $x = 0, 1$ での振る舞いが、$\dfrac{1}{\sqrt{1-x^2}}$ の $x = 0, 1$ での振る舞いより、緩やかであるからと推測できる。

実は、課題 4 のような工夫を施すと、I の値を高精度 (誤差が 10^{-16} 程度) に求めることが可能である。

課題 2

有名な Euler のガンマ関数

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}\,dt \quad (x > 0)$$

を数値積分で計算してみよう、という課題である。

(出題の意図: 関数の中にはこのガンマ関数のように、積分を使って定義される関数が多い。重要な関数は、その数値計算の方法も詳しく研究されていることが多いが、数値積分によって計算することもそれほど難しくない場合が多いことを知っておくのは有益であろう。)

積分範囲が ∞ まで伸びているので、広義積分である。幸い、e^{-t} という因子のおかげで、$t \to \infty$ のとき、被積分関数は十分速く減衰する。

「数値積分」の講義ノート (桂田 [1]) の §5.2.4 「半無限区間上の一重指数関数的な減衰をする関数の積分」に従い、

$$x = \varphi_4(t) := \exp\left(t - \exp(-t)\right) \quad (t \in \mathbb{R})$$

という変数変換を施してから、台形公式を適用するタイプの DE 公式を用いれば良い。

ところで、x が $0 < x < 1$ のときは、被積分関数が $t \to +0$ で発散するので、左端の方でも "広義積分である"。つまり I は

$$I = \lim_{\substack{\varepsilon \to +0 \\ R \to \infty}} \int_\varepsilon^R t^{x-1}e^{-t}\,dt$$

として計算する広義積分である。

このことから、x が小さくなり、0 に近いところでは、精度の良い数値積分が得にくくなることが予想される。その辺のことに注意して数値実験してもらえると良い。

ガンマ関数はあちこちで現れ、重要なので、非常に詳しく調べられている。課題文でも述べたように、UNIX の数学関数ライブラリィ(Mac OS X の C 言語のライブラリィには含まれている) にも、$\log\Gamma$ を計算する `lgamma()` という関数が用意されている。ゆえに $\Gamma(x)$ の値が知りたい場合は、C 言語のプログラムでは、`exp(lgamma(`x`))` とすれば良い。

余談になるが、標準正規分布の分布関数

$$F(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-t^2/2}\, dt$$

や

$$\phi(x) := \int_{0}^{x} \frac{1}{\sqrt{2\pi}} e^{-t^2/2}\, dt = F(x) - \frac{1}{2}$$

なども、初等関数では表せず、初等関数の積分で表せる関数である。UNIX の数学関数ライブラリィには、`erf()`, `erfc()` という関数が用意されている。

$$\mathrm{erf}(x) := \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2}\, dt, \quad \mathrm{erfc}(x) := 1 - \mathrm{erf}(x).$$

$$\phi(x) = \frac{1}{2}\mathrm{erf}(x/\sqrt{2})$$

という関係がある。

課題 3

$$T_{N,\,\text{補}} := T_N - \frac{h^2}{12}\left(f'(b) - f'(a)\right).$$

台形公式を補正したものなので、台形公式の結果と比べる実験をするのが良いだろう。例えば `example2.c` のようなプログラムをたたき台にしてみる。

例えば次のような関数を作る (`df()` は f' を計算する関数)。

```
/* 補正複合台形公式 */
double trapezoidal2(ddfunction f, ddfunction df, double a, double b, int N)
{
  double h;
  h = (b - a) / N;
  return trapezoidal(f, a, b, N) - h * h * (df(b) - df(a)) / 12.0;
}
```

中点公式の補正は分からなかった人も多いようだが、発見できた人もいて、

$$M_{N,\,\text{補}} := M_N + \frac{h^2}{24}\left(f'(b) - f'(a)\right)$$

とすれば良い。

Euler-Maclaurin 展開を用いれば、補正台形公式を更に補正した

$$T_{N,\,\text{補補}} := T_N - \frac{h^2}{12}\left(f'(b) - f'(a)\right) + \frac{h^4}{720}\left(f'''(b) - f'''(a)\right)$$

を得ることも出来る。この場合は、被積分関数 f の 3 階導関数が必要になる。これで試したレポートを書いた人もいた。

ここでは、$I = \int_1^2 \log x \; dx = \log 4 - 1$ について、複合台形公式、複合中点公式、複合 Simpson 公式、補正複合台形公式、2 回補正した複合台形公式の結果を見てみよう。

```
./kadai1-3
#    N       I-M_N          I-T_N          I-S_N         I-補 T_N        I-補補 T_N
     2   -5.085309e-03   1.027501e-02   4.597590e-04  -1.416547e-04   1.025498e-05
     4   -1.293949e-03   2.594852e-03   3.479831e-05  -9.314956e-06   1.794014e-07
     8   -3.250044e-04   6.504512e-04   2.317654e-06  -5.904989e-07   2.898481e-09
    16   -8.134780e-05   1.627234e-04   1.474441e-07  -3.704164e-08   4.569012e-11
    32   -2.034302e-05   4.068779e-05   9.257558e-09  -2.317243e-09   7.153722e-13
    64   -5.086136e-06   1.017238e-05   5.792660e-10  -1.448613e-10   1.110223e-14
   128   -1.271558e-06   2.543122e-06   3.621459e-11  -9.054424e-12   1.110223e-16
   256   -3.178909e-07   6.357823e-07   2.263467e-12  -5.658807e-13   0.000000e+00
   512   -7.947283e-08   1.589457e-07   1.414979e-13  -3.530509e-14   5.551115e-17
  1024   -1.986821e-08   3.973643e-08   8.881784e-15  -2.109424e-15   1.110223e-16
  2048   -4.967053e-09   9.934107e-09   6.106227e-16  -2.775558e-16  -1.665335e-16
  4096   -1.241763e-09   2.483527e-09   2.220446e-16  -2.775558e-16  -2.775558e-16
  8192   -3.104412e-10   6.208816e-10   5.551115e-17  -1.110223e-16  -1.110223e-16
 16384   -7.760981e-11   1.552213e-10  -2.775558e-16   8.881784e-16   8.881784e-16
 32768   -1.940148e-11   3.880235e-11   6.106227e-16  -2.775558e-15  -2.775558e-15
 65536   -4.851064e-12   9.704404e-12  -2.220446e-16   3.108624e-15   3.108624e-15
```

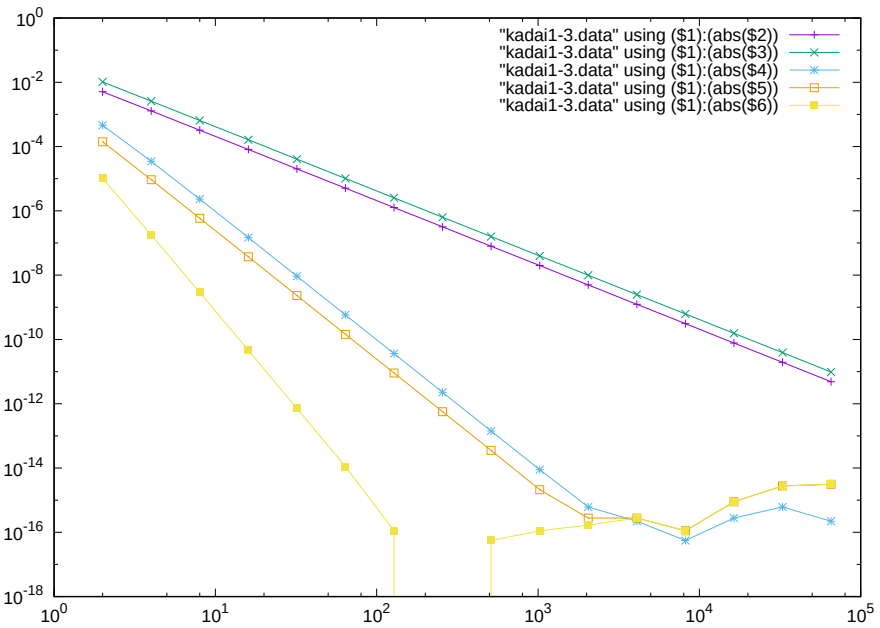

図 1: $\int_1^2 \log x \; dx$ を中点公式、台形公式、Simpson 公式、補正台形公式、補正を 2 回行った台形公式で計算したときの誤差

補正台形公式は、Simpson 公式と同様に、4 位の公式になっていることが分かる。

注意　N を大きくしたときに、誤差がこのように減衰するグラフは既に見せたはずだが、自分で計算すると、頭を使って考えてくれるらしくて、N が 10^3 を超えるあたりからグラフが直線にならず、ジグザグするのはなぜか？という質問を複数の人から受けた。

課題 4

$\displaystyle\int_{-1}^{1} \frac{dx}{\sqrt{1-x^2}}$ を素朴な DE 公式のプログラムで計算すると、8 桁程度の精度の値しか得られない。これを何とか改善してみよう、という話。

DE 公式では、tanh という関数を用いるが、

$$t \text{ が大きいときの } \tanh(t)-1, \quad t \text{ が小さいときの } \tanh(t)-(-1)=\tanh(t)+1$$

の精度の良い値が知りたい場合がある。$t \gg 1$ のとき、$\tanh(t)$ を浮動小数点数を用いて計算すると、1 になってしまい、$\tanh(t)-1$ はいわゆる桁落ちをして、0 になったり ($\dfrac{1}{\sqrt{1-x^2}}$ の分母が 0 になる !!)、0 にならないにしても精度が低くなる。

実際、Mac で使っている C 言語処理系の double (IEEE 754 規格に準拠) で素朴にやると、`tanh(20.0)-1` は 0 になる。

これは tanh に限った問題ではなく、C 言語の数学関数ライブラリィには、$x \fallingdotseq 0$ に対して、e^x-1 を桁落ちを避けて精度よく計算するために、`expm1()` という関数が用意されている。その真似をしてみよう。

$$(1) \qquad \mathrm{tanhm1}(t)=\tanh(t)-1=\frac{\sinh(t)}{\cosh(t)}-1=\frac{e^t-e^{-t}}{e^t+e^{-t}}-1=\frac{-2}{1+e^{2t}}=\frac{-2e^{-2t}}{1+e^{-2t}}.$$

特に難しいところのない計算ですが、こうすると桁落ちを避けられます。

```
//  tanh(t)-1 の計算 (t ≫ 1 の場合用)
double tanhm1(double t)
{
  if (t <= 354)
    return - 2.0 / (1.0 + exp(2.0 * t));
  else {
    printf("tahnm1(): %g は大きすぎる\n", t);
    return 0;
  }
}

//  tanh(t)+1 の計算 (t ≪-1 の場合用)
double tanhp1(double t)
{
  if (t >= - 354)
    return 2.0 / (1.0 + exp(- 2.0 * t));
  else {
    printf("tahnp1(): %g は小さ過ぎる\n", t);
    return 0;
  }
}
```

必要なところで、この `tanhm1()`, `tanhp1()` を使うようにすると、次のように結果が改善できる ($h=1/4$ で 1×10^{-15} 程度の誤差)。

```
$ cc kadai1-4.c
$ ./a.out
test (1/sqrt(1-x^2) の積分)
h=1.000000, I_h=       3.1435079789309328, I_h-I=1.915325e-03
h=0.500000, I_h=       3.1415926733057051, I_h-I=1.971591e-08
h=0.250000, I_h=       3.1415926535897940, I_h-I=8.881784e-16
h=0.125000, I_h=       3.1415926535897940, I_h-I=8.881784e-16
h=0.062500, I_h=       3.1415926535897936, I_h-I=4.440892e-16
h=0.031250, I_h=       3.1415926535897927, I_h-I=-4.440892e-16
h=0.015625, I_h=       3.1415926535897918, I_h-I=-1.332268e-15
h=0.007812, I_h=       3.1415926535897976, I_h-I=4.440892e-15
h=0.003906, I_h=       3.1415926535897958, I_h-I=2.664535e-15
h=0.001953, I_h=       3.1415926535897905, I_h-I=-2.664535e-15
$
```

参考文献

[1] 桂田祐史, 数値積分 講義ノート, http://nalab.mind.meiji.ac.jp/~mk/complex2/numerical-integration.pdf