数値解析レポート No.3

4 年 39 番 湯嶋 皓騎

2024 年 2 月 2 日

1 指数近似

式 (1) のデータを式 (2) の式を用いて指数近似を行い，結果のグラフを示せ.

$$(x, y) = (1.5, 8.96),\ (2.0, 14.78),\ (2.5, 24.36),\ (3.0, 40.17) \tag{1}$$

$$y = a_0 e^{a_1 x} \tag{2}$$

式 (2) の両辺の対数を取ると，式 (3) のようになる.

$$\ln y = \ln a_0 + a_1 x \tag{3}$$

右辺が線形となるため，課題 8 で用いた線形近似のアルゴリズムを用いて指数近似を行うことができる.

プログラムを List. 1 に示す.

Listing 1: 指数近似

```
Vector *exponentialApproximation(const Matrix *coordinates) {
  size_t param_num = 2; // a0, a1
  Vector *vector = allocVector(param_num);
  SquareMatrix *matrix = allocSquareMatrix(param_num);
  size_t i, j, k;

  for (i = 0; i < vector->length; i++) {
      for (j = 0; j < coordinates->row; j++) {
          // log(y_i) = log(a0_i) + a1_i x^i
          vector->data[i] += log(coordinates->data[j][1]) *
                             power(coordinates->data[j][0], i);
      }
  }

  for (i = 0; i < matrix->size; i++) {
      for (j = i; j < matrix->size; j++) {
          for (k = 0; k < coordinates->row; k++) {
```

```
18                 matrix->data[i][j] += power(coordinates->data[k][0], i + j);
19             }
20             matrix->data[j][i] = matrix->data[i][j];
21         }
22     }
23
24     return solveLinearEquation(matrix, vector);
25 }
```

ただし，`solveLinearEquation` は引数の行列とベクトルの線形方程式を解いた結果を返す関数であり，`power(double, size_t)` は引数の値を第二引数の値乗した値を返す関数である．また，解いた結果のベクトルの第一要素が $\log(a_0)$，第二要素が a_1 となる．第一要素が対数になるのは，はじめに対数を取ったためである．

これを実行した結果，$a_0 \approx 1.96, a_1 \approx 1.01$ が得られた．この結果を用いて指数近似を行った結果を Fig. 1 に示す．

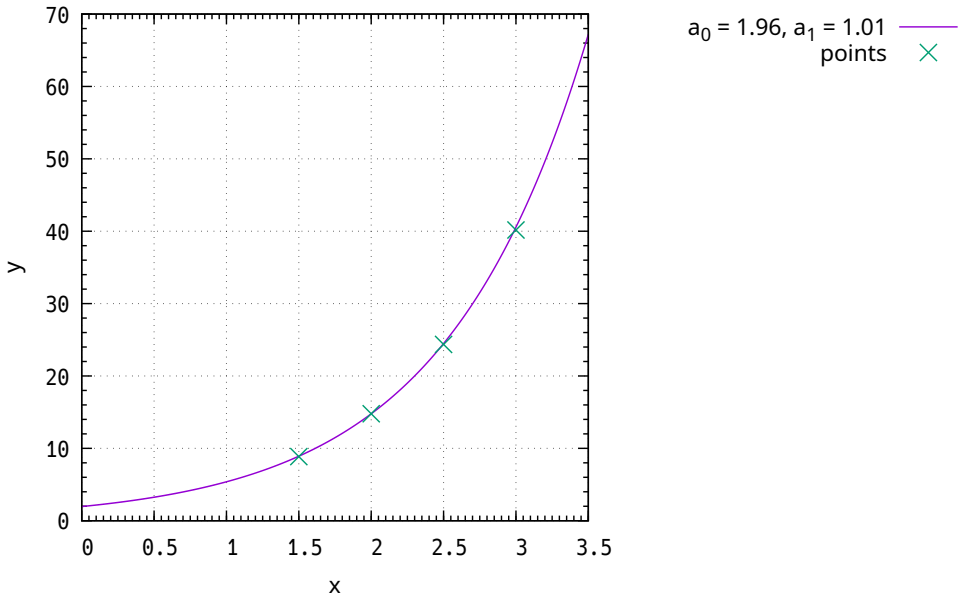

Fig. 1: 指数近似の結果

Fig. 1 より，正しく指数近似がなされていそうだとわかる．

2　これまでに行った相互評価について

これまでに行った相互評価について述べる．

2.1　良いと思うところ，参考になったところ

相互評価を行うことで，プログラムをどのように改善すればよいか，どのような点をアドバイスすればよいかがわかった．例えばプログラムの動作に問題がある場合，なぜ問題が起きているのか，どのように改善すれば解決するのかをフィードバックとして伝えることができた．

また，相互評価と直接関係することではないが，Windows とそれ以外の Unix 系 OS での動作の違いについても知ることができた．具体的には，`size_t` 型の変数を printf で出力する際に，Unix 系 OS では %lu もしくは %zu と指定するが，Windows では %Iu と指定しないと警告が表示されることがわかった．

2.2 改善すべきところ，改善案，バグ報告

2.2.1 プログラムの入力について

最初の講義で「入力は csv ファイルのみとする」とアナウンスされていたにも関わらず，ファイル名やパラメータを対話的に入力させるプログラムが散見された．あまり本質的な問題ではないが，入力方法が統一されていないと相互評価がしにくいため，統一できるようにしたかった．

2.2.2 プログラムの品質について

未使用の変数がそのまま残っている，変数名が不適切，動的に確保した領域を解放し忘れているプログラムが多く見られた．未使用の変数に関してはコンパイラの警告レベルを上げることで解決できるので，提出前にコンパイラの警告レベルを上げた状態でコンパイルが通ることを確認するようアナウンスして欲しい．gcc なら `-Wall -Wextra -Werror` オプションを，bcc32c なら `-Weverything -Werror` を付加することで多くの警告をエラーとして扱うことができ，プログラムの品質を上げることができる．さらに，gcc なら `-fanalyzer` オプションをつけると，簡単に静的解析ができるので，これもアナウンスして欲しい．

2.2.3 相互評価の相手について

同じ人と思われるプログラムが 5 回中 3 回の割合で回ってきたので，もう少し他の人のプログラムも評価したかった．

3 非線形関数に対する数値積分法

解析的に解ける非線形関数に対して台形・シンプソン・ロンバーグの 3 手法による数値積分し，それぞれの分割数と誤差，及び計算量・計算速度について調査する．

3.1 調査プログラム

調査プログラムを List. 2 に示す．

Listing 2: 調査プログラム

```
1 #include <math.h>
2 #include <stdbool.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 #include "NAbasic.h"
7
8 #define ROMBERG_TABLE_MAX 255
9 #define EPS 1e-7
```

```
10
11 typedef double (*func_t)(double);
12
13 void dump_romberg_table(size_t n, size_t p, Vector **T, size_t len) {
14     size_t i, j;
15     printf("division_num:%zu dimension:%zu\n", n, p);
16     puts("Romberg Table");
17     for (i = 0; i < len; i++) {
18         for (j = 0; j < T[i]->length; j++) {
19             printf("%.10f ", T[i]->data[j]);
20         }
21         putchar('\n');
22     }
23 }
24
25 double trapezoid(func_t f, double a, double b, size_t n) {
26     double h = (b - a) / n;
27     double sum = 0.0;
28     size_t i;
29
30     // sum = h*(f(a)/2 + f(a + h) + f(a + 2h) + ... + f(a + (n - 1)h) + f(b)/2)
31     // where (a + nh = b)
32     for (i = 1; i < n; i++) sum += f(a + i * h);
33     sum += (f(a) + f(b)) / 2.0;
34
35     return h * sum;
36 }
37
38 double simpson(func_t f, double a, double b, size_t n) {
39     double h = (b - a) / 2.0 / n;
40     double sum = 0.0;
41     size_t i;
42
43     // sum = h*(f(a) + 4*f(a + h) + 2*f(a + 2h) + ... + f(b)/6)/3.0
44     for (i = 1; i < 2 * n; i++) {
45         size_t power = (i % 2 == 0) ? 2 : 4;
46         sum += f(a + i * h) * power;
47     }
48     sum += f(a) + f(b);
49
50     return h * sum / 3.0;
51 }
52
53 double romberg(func_t f, double a1, double a2, size_t k_max) {
54     size_t i, k, m = 1;
```

```
55      size_t n = 1, p = 1; // n = 2^k, p = 4^m
56      double result;
57      Vector **T;
58      bool is_converged = false;
59
60      // T を長さk_max で確保し，T[0] は長さ1 で確保する
61
62      T[0]->data[0] = (a2 - a1) * (f(a1) + f(a2)) / 2.0;
63
64      for (k = 1; ((k < k_max) && !(is_converged)); k++) {
65          double h, sum = 0.0;
66
67          // T[k] を長さk+1 で確保
68
69          n *= 2; // n = 2^k
70          h = (a2 - a1) / n;
71          // sum = f(a + h) + f(a + 3h) + ... + f(a+(2n-1)h)
72          for (i = 1; i < n; i += 2) sum += f(a1 + i * h);
73          // T[k][0] = T[k-1][0]/2 + h*sum (k >= 1)
74          T[k]->data[0] = T[k - 1]->data[0] / 2 + h * sum;
75
76          for (m = p = 1; m <= k; m++) {
77              p *= 2; // p = 2^m
78              // T[k][m] = T[k][m-1] + (T[k][m-1] - T[k-1][m-1])/(p^2-1) (k >= 1)
79              T[k]->data[m] =
80                  T[k]->data[m - 1] +
81                  (T[k]->data[m - 1] - T[k - 1]->data[m - 1]) / (p * p - 1);
82              if (fabs(T[k]->data[m] - T[k - 1]->data[m - 1]) < EPS) {
83                  T[k]->length = m + 1;
84                  is_converged = true;
85                  break;
86              }
87          }
88      }
89
90      result = T[k - 1]->data[m - 1];
91      for (i = 0; i < k; i++) freeVector(T[i]);
92      free(T);
93
94      return result;
95  }
96
97  int main(void) {
98      double res;
99      double x_s = 0, x_e = 3;
```

```
100     size_t n;
101     func_t f = exp;
102     double exact_value;
103
104     printf("Trapezoid\n");
105     for (n = 1; n <= 1000; n++) {
106         res = trapezoid(f, x_s, x_e, n);
107         printf("%zu,%.10f,%.10f\n", n, res, (res - exact_value));
108     }
109
110     printf("Simpson\n");
111     for (n = 1; n <= 1000; n++) {
112         res = simpson(f, x_s, x_e, n);
113         printf("%zu,%.10f,%.10f\n", n, res, (res - exact_value));
114     }
115
116     printf("Romberg\n");
117     for (n = 1; n <= 10; n++) {
118         res = romberg(f, x_s, x_e, n);
119         printf("%zu,%.10f,%.10f\n", n, res, (res - exact_value));
120     }
121     return EXIT_SUCCESS;
122 }
```

変数 `x_s`, `x_e` に積分範囲の始点と終点，`exact_value` に解析解の値を，`f` に積分する関数を指定する．

3.2 分割数と誤差

典型的な非線形関数である指数関数と周期関数である三角関数に対して台形・シンプソン・ロンバーグの 3 手法による数値積分を行い，分割数と誤差の関係を調査する．

3.2.1 指数関数

指数関数 $f(x) = e^x$ に対して，台形・シンプソン・ロンバーグの 3 手法による数値積分を行った．積分範囲は $[0, 3]$ とする．

まず，解析解は

$$\int_0^3 e^x dx = e^3 - 1 \approx 19.08553692 \tag{4}$$

である．

次に数値積分の分割数対誤差のグラフを Fig. 2 に示す．両対数グラフであることに注意すること．

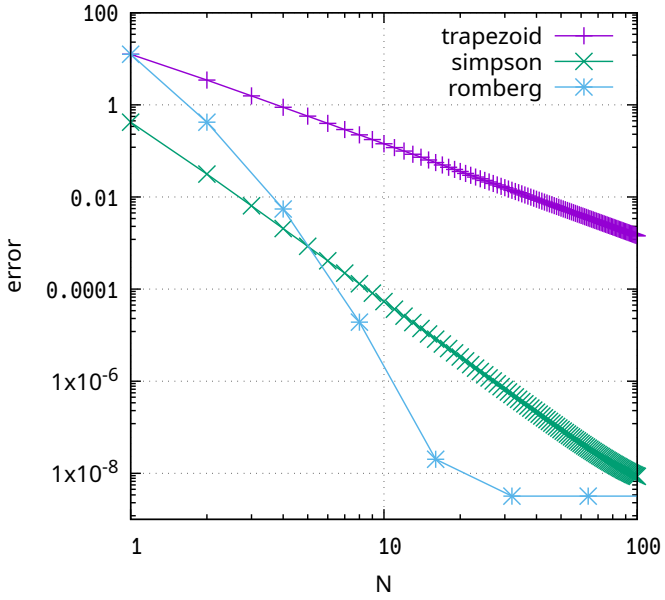

Fig. 2: 指数関数に対する数値積分の分割数対誤差

ただし，ロンバーグ法では分割数 32，補間次数 16 でステップ間誤差が 10^{-7} 以下になるため，実際には分割数 32 以降は計算を行っていない．

Fig. 2 を見ると，台形法とシンプソン法は傾きがほぼ一定だから分割数と誤差の関係は冪関数的であることがわかる．台形法の傾きは -2，シンプソン法の傾きは -4 だから，台形法の誤差は $O(N^2)$，シンプソン法の誤差は $O(N^4)$ の速さで小さくなることがわかる．一方，ロンバーグ法は冪関数的ではなく，他の二つの手法と比べて $N = 8$ 以上で小さい誤差が得られており，$N = 32$ でステップ間誤差が 10^{-7} 以下になるため，計算を打ち切り，その時点での絶対誤差は 10^{-9} オーダーである．

3.2.2 三角関数

三角関数 $f(x) = \sin x$ に対して，台形・シンプソン・ロンバーグの 3 手法による数値積分を行った．積分範囲は $[0, \pi/2]$ とする．ここで，`PI`$= 3.14159265358979323846$ とした．

まず，解析解は

$$\int_0^{\pi/2} \sin x dx = 1 \tag{5}$$

である．

次に数値積分の分割数対誤差のグラフを Fig. 3 に示す．両対数グラフであることに注意すること．

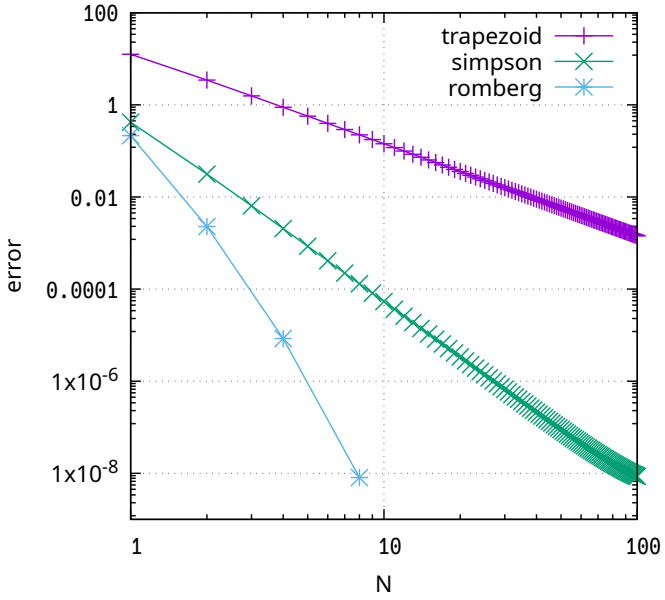

Fig. 3: 三角関数に対する数値積分の分割数対誤差

Fig. 3 より，台形法の誤差は $O(h^2)$，シンプソン法は $O(h^4)$ であることが再確認できる．また，ロンバーグ法は $N = 16$ で誤差が 0 になってしまったため，それ以降はグラフ上に描画されていない．

3.2.3 まとめ

台形法の誤差は $O(N^2)$，シンプソン法は $O(N^4)$ の速さで小さくなり，またいずれの場合においても，ロンバーグ法は他の二つの手法と比べてより早く誤差の小さな値を得ることができることがわかった．

3.3 計算量・計算速度

台形・シンプソン・ロンバーグの 3 手法における計算量と計算速度について調査する．

3.3.1 計算量

分割数を N として，台形法とシンプソン法の計算量は $O(N)$ であり，ロンバーグ法の計算量は $O(N^2)$ である．これは関数の入れ子になっている for ループの最大値を数えることでわかる．

3.3.2 計算速度

一見すると計算量が $O(N^2)$ であるロンバーグ法が遅いように見えるが，前述の通りロンバーグ法は収束が速いため，実際には台形法やシンプソン法よりも早く収束した．そのため，今回のケースではロンバーグ法を用いることが最も効率的であると言える．しかし，ロンバーグ法は補外を用いて漸化的に精度を上げていくため，処理の並列化が困難である．一方，台形法やシンプソン法は互いの計算が独立であるため，並列化が容易である．そのため，非常に広い範囲の数値積分を行う場合には，ロンバーグ法よりも台形法やシンプソン法を用いることが望ましいだろう．

4 感想

今回のレポート執筆を通して講義で学んだ内容を復習することで，数値積分法についての理解が深まった．今後は演習を行った数値積分法以外のアルゴリズムについても調査していきたい．

参考文献

[1] Embarcadero C++ Compiler Warnings: https://docwiki.embarcadero.com/RADStudio/Sydney/en/C%2B%2B_Compiler_Warnings