数値解析レポート No.3

32 番 平田 蓮

1 指数近似

指数近似 $y = a_0 e^{a_1 x}$ について、両辺対数を取ると、$\ln y = \ln a_0 + a_1 x$ となる。この式は多項式近似と同じ形であるので、a_0, a_1 を求めることができる。これを元に指数近似を行う関数 `exp_approximation()` をリスト 1 に示す。

リスト 1　exp_approximation

```
#define A_NUM 2    / / coefficient number
#define DATA_NUM 4 / / sample number

double *exp_approxmation(
    double *x, / / sample x coordinates
    double *y  / / sample y coordinates
) {
    double
        *a = allocVector(A_NUM),
        *b = allocVector(A_NUM),
        **A = allocMatrix(A_NUM, A_NUM);

    for (i = 0; i < A_NUM; i++) {
        for (j = 0; j < DATA_NUM; j++) {
            b[i] += log(y[j]) * pow(x[j], i);
        }
    }

    for (i = 0; i < A_NUM; i++) {
        for (j = i; j < A_NUM; j++) {
            for (k = 0; k < DATA_NUM; k++) {
                A[i][j] += pow(x[k], i + j);
            }
            A[j][i] = A[i][j];
        }
    }

    gauss(A, b, A_NUM, A_NUM); / / gaussian elimination
    backward_substitution(A, b, a, A_NUM); / / backward substitution
```

```
30      a[0] = exp(a[0]);
31
32      return a;
33  }
```

与えられたデータを近似すると、$a_0 \approx 2, a_1 \approx 1$ が得られる。与えられたデータと $y = 2e^x$ をプロットしたものを図 1 に示す。

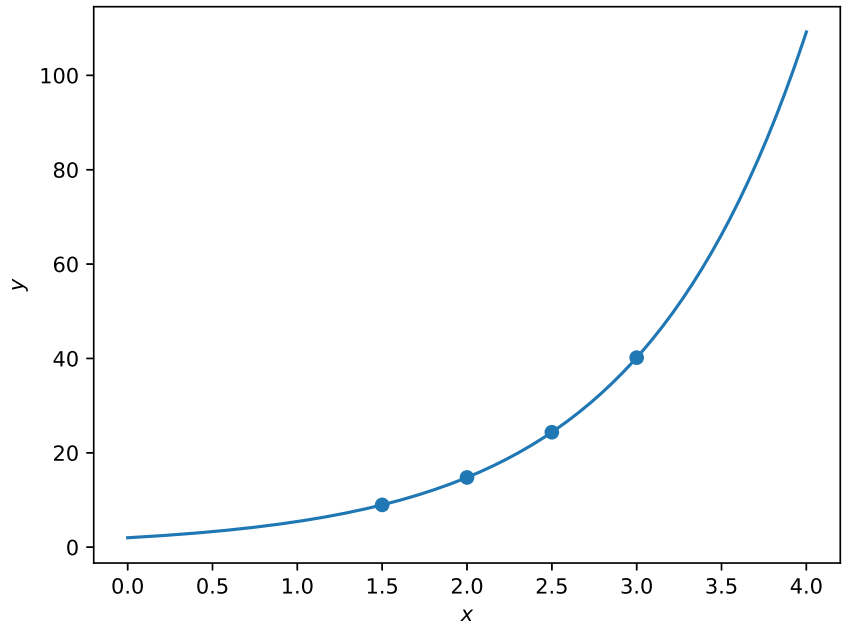

図 1　指数近似

2　サービス問題

2.1　良いところ · 参考になったところ

他人のコードを読むにあたって、特にコメントの書き方は興味深かった。自分は普段あまりコメントを書かないので、他人に伝わりやすいコメントという観点でとても参考になったところもあった。

また、他人のコードを読むことで同じアルゴリズムでも少し違った実装の仕方を発見できたり、今後のプログラミングに活かせそうな学びを得ることができた。

2.2　感想

特に、プログラムの汎用性という部分であまり上手にプログラムを書けない人が多い印象であった。自作関数、構造体など、汎用性を高めるためのパーツはこれまでの授業で扱ったが、実際にそれらを利用して汎用性のないプログラムにどのように汎用性を持たせることができるかといった内容の講義があればよりクラスのプ

ログラミング力が向上するのではないかと思った。

3　数値積分の比較

台形公式、シンプソン公式、ロンバーグ積分の三つの手法で定積分を行う関数をリスト 2 に示す。

リスト 2　integral.c

```
double trapezoid(
    double (*f)(double), / / function
    double a, / / start
    double b, / / end
    int n / / partition number
) {
    if (a > b) return -trapezoid(f, b, a, n);
    double sum = 0, dx = (b - a) / n, x;
    for (x = a; x <= b; x += dx) {
        sum += f(x);
    }
    sum -= (f(a) + f(b)) / 2.0;
    return sum * dx;
}

double simpson(
    double (*f)(double), / / function
    double a, / / start
    double b, / / end
    int n / / partition number
) {
    if (a > b) return -simpson(f, b, a, n);
    double sum = 0, dx = (b - a) / n, x;
    sum = f(a) + f(b);
    for (x = a + dx; x <= b; x += dx) {
        sum += 4.0 * f(x - dx / 2.0) + 2.0 * f(x);
    }
    sum -= 2.0 * f(b);

    return sum * dx / 6.0;
}

double romberg(
    double (*f)(double), / / function
    double a, / / start
    double b, / / end
    int n / / partition number
    double **T / / 2d array for calculation
) {
```

```
40      if (a > b) return -romberg(f, a, b, n, T);
41      int k, m;
42      double h, x;
43
44      T[0][0] = (b - a) * (f(a) + f(b)) / 2.0;
45
46      for (k = 1; k < n; k++) {
47          h = (b - a) / pow(2, k);
48
49          T[k][0] = T[k - 1][0] / 2.0;
50          for (x = a + h; x <= b - h; x += 2.0 * h) {
51              T[k][0] += h * f(x);
52          }
53
54          for (m = 1; m <= k; m++) {
55              T[k][m] = T[k][m - 1] + (T[k][m - 1] - T[k - 1][m - 1]) / (pow(4.0, m)
        - 1.0);
56          }
57      }
58
59      return T[n - 1][n - 1];
60  }
```

3.1 分割数の増加による誤差の推移

このプログラムを使って $y = \sin x$ $(0 \leq x \leq \frac{\pi}{2})$、$y = x \ln x$ $(0 \leq x \leq e)$ の定積分を求める。

まず理論値を求める。$y = \sin x$ について、$\int_0^{\frac{\pi}{2}} \sin x dx = 1$ となる。$y = x \ln x$ については広義積分を行い、$\int_0^e x \ln x dx = \left[\frac{x^2 \ln x}{2} - \frac{x^2}{4}\right]_0^e = \frac{e^2}{4} \approx 1.847264$ となる。

図 2、3 に各分割数について定積分の計算結果と理論値との差をプロットしたものを示す。

図より、ロンバーグ積分、シンプソン公式、台形公式の順に収束が早いことがわかる。また、分割数が大きくなっても台形公式とシンプソン公式は誤差が出ることがあるが、ロンバーグ積分は完全に収束する。関数と分割数の相性によっては台形公式やシンプソン公式の方が小さい分割数で誤差がロンバーグ積分より少なくなる場合もあることがわかる。

3.2 各手法の計算量

リスト 2 を見ればわかるように、台形公式とシンプソン公式の計算量は分割数 N に対して $O(N)$ である。対してロンバーグ積分は $O\left(\sum_{k=0}^{N-1} 2^k\right) = O\left(2^N\right)$ であるので、現実的に計算可能なロンバーグ積分は $N = 30$ 前後である。

しかし、誤差の小ささをみると、台形公式やシンプソン公式よりロンバーグ積分を使う方が良いと考えられる。台形公式とシンプソン公式は関数と相性の良い分割数についてはほぼ誤差がない値を導けるが、そうでな

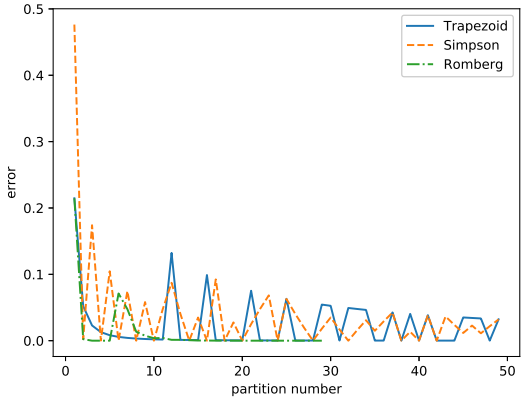

図 2　$y = \sin x$ の定積分の誤差の推移

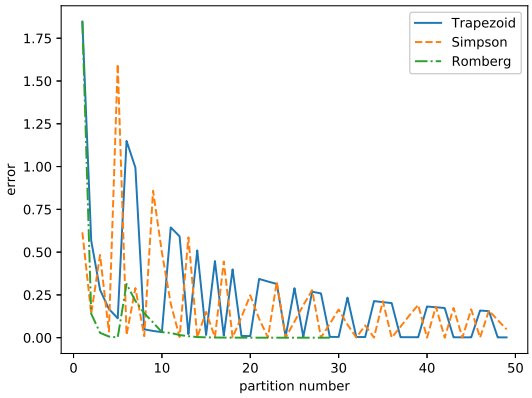

図 3　$y = x \ln x$ の定積分の誤差の推移

い分割数についてはある程度の誤差を出してしまう。対してロンバーグ積分は分割数を増やしたときに値の変化がほぼなくなるまで計算をすると理論値にとても近い値を導くことができる。