電気電子計算工学及演習 課題 4

三軒家 佑將（さんげんや ゆうすけ）
3 回生
1026-26-5817
a0146089

1　プログラムの説明

1.1　概要

本レポートにおいては、プログラム言語として Haskell を採用した。行列演算ライブラリとしては、hmatrix(https://hackage.haskell.org/package/hmatrix) を用いた。

また、グラフを描画する際には、Jupyter notebook および IRuby kernel を用いた。

プログラムを実行する手順は、以下のとおりである。

```
> cd haskell
> stack build
> stack exec one-power # 課題4.1.1
> stack exec one-jacobi # 課題4.1.2
> stack exec two # 課題4.2
> stack exec three-trapezoidal # 課題4.3
> stack exec three-simpsons # 課題4.3
> stack exec two-extra # 考察
```

1.2　各機能・関数の説明

プログラムを作成するにあたって、見通しを良くするために、プログラムを複数のファイルに分割している。ここでは、各ファイルごとに、そのファイルの担う機能と、そのファイル内にある関数の機能などについて簡単に説明する。

各関数の詳しい使用方法などは、プログラム内のコメントにて示したので、そちらを参照されたい。

app/OnePower.hs

課題 4.1.1 を解くプログラム。固有ベクトルと、収束判定に用いたスカラ値の値を出力する。

app/OneJacobi.hs

課題 4.1.2 を解くプログラム。UU^{T} と $U\Lambda U^{T}$ を出力する。

app/Two.hs

課題 4.2.1、課題 4.2.2 を解くプログラム。

- k
- 複合台形公式による積分結果
- 複合シンプソン公式による積分結果

を CSV にして標準出力に出力する。

app/ThreeTrapezoidal.hs

課題 4.3 を解くプログラム。各定義式から π を求めるが、そのとき複合台形公式を用いる。

以下の内容が CSV 形式で標準出力に出力される。

- k
- 問の定義式 1 による π の計算結果
- 問の定義式 2 による π の計算結果
- 問の定義式 3 による π の計算結果
- 問の定義式 4 による π の計算結果

app/ThreeSimpsons.hs

課題 4.3 を解くプログラム。app/Threetrapezoidal.hs と同様のことを行なうが、その際、複合台形公式ではなく、複合シンプソン公式を用いる。

src/Types.hs

本プログラムにて使用する型シノニムを定義したファイル。

src/Integrator/Default.hs

複合台形公式によって積分計算を行なう関数 (compositTrapezoidalRule) と、複合シンプソン公式によって積分計算を行なう関数 (compositSimpsonsRule) を定義したファイル。

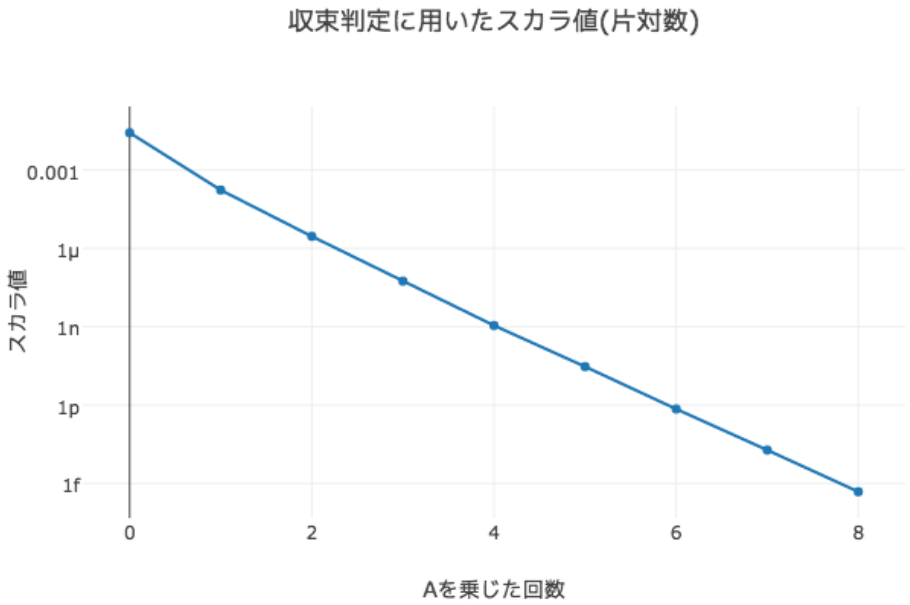

図 1　収束判定に用いたスカラ値

src/Three.hs

calc 関数を export しており、課題 4.3 を解くのに使われる。

src/Integrator/Recursive.hs

src/Integrator/Default.hs と同様の関数を export しているが、その際のアルゴリズムが、Default.hs のそれとは異なる。詳細は考察にて述べる。

app/TwoExtra.hs

考察に用いるプログラム。詳細は考察にて述べる。

2　課題 4.1.1

べき乗法によって、固有ベクトル u_1 を計算した。このとき、収束条件は、テキスト 6.3 節に記載されたものを使用し、また、$\epsilon = 10^{-15}$ とした。

2.1　結果

収束判定に用いたスカラ値 ($|\gamma_{max} - \gamma_{min}|$) と、行列 A を初期値 u_0 に乗じた回数の関係を片対数グラフに描いたのが図 1 である。

A を乗じるごとに、指数的に u_1 へと収束していることがわかる。A を乗じるたびに、ある一定の割合だけ u_1 へ近づくとすれば、直感的に納得できる結果である。

3 課題 4.1.2

ヤコビ法によって、固有値行列 Λ と、固有ベクトル行列 U を求めた。

3.1 結果

$\mathrm{UU^T}$ と $\mathrm{U\Lambda U^T}$ を出力した結果、以下のようになった。

$$\mathrm{UU^T} = \begin{pmatrix} 1.00e+00 & -2.43e-16 & 8.33e-17 & 2.50e-16 & 8.33e-17 & -4.30e-16 \\ -2.43e-16 & 1.00e+00 & 2.91e-16 & 3.33e-16 & -1.39e-16 & -2.22e-16 \\ 8.33e-17 & 2.91e-16 & 1.00e+00 & -1.94e-16 & -5.55e-17 & -1.39e-16 \\ 2.50e-16 & 3.33e-16 & -1.94e-16 & 1.00e+00 & -1.94e-16 & -2.78e-17 \\ 8.33e-17 & -1.39e-16 & -5.55e-17 & -1.94e-16 & 1.00e+00 & -5.55e-17 \\ -4.30e-16 & -2.22e-16 & -1.39e-16 & -2.78e-17 & -5.55e-17 & 1.00e+00 \end{pmatrix}$$

$$\mathrm{U\Lambda U^T} = \begin{pmatrix} 2.00 & 3.00 & 4.00 & 5.00 & 6.00 & 7.00 \\ 3.00 & 8.00 & 9.00 & 10.00 & 11.00 & 12.00 \\ 4.00 & 9.00 & 13.00 & 14.00 & 15.00 & 16.00 \\ 5.00 & 10.00 & 14.00 & 17.00 & 18.00 & 19.00 \\ 6.00 & 11.00 & 15.00 & 18.00 & 20.00 & 21.00 \\ 7.00 & 12.00 & 16.00 & 19.00 & 21.00 & 22.00 \end{pmatrix}$$

$\mathrm{UU^T}$ は単位行列と、$\mathrm{U\Lambda U^T}$ は A と、それぞれほとんど等しい値となっていることがわかる、

4 課題 4.2

積分 $\int_0^1 x^{19} dx$ を、分割数 $n = 2^k$ の複合台形公式と複合シンプソン公式を用いて求め、真の値との相対誤差の対数を求めた。

4.1 結果

k と相対誤差の対数の関係をグラフに描いたのが図 2 である。

複合台形公式でも複合シンプソン公式でも、k に対して指数的に相対誤差が減少している事がわかる。また、減少のスピードは、複合シンプソン公式の方が速いことがわかる。

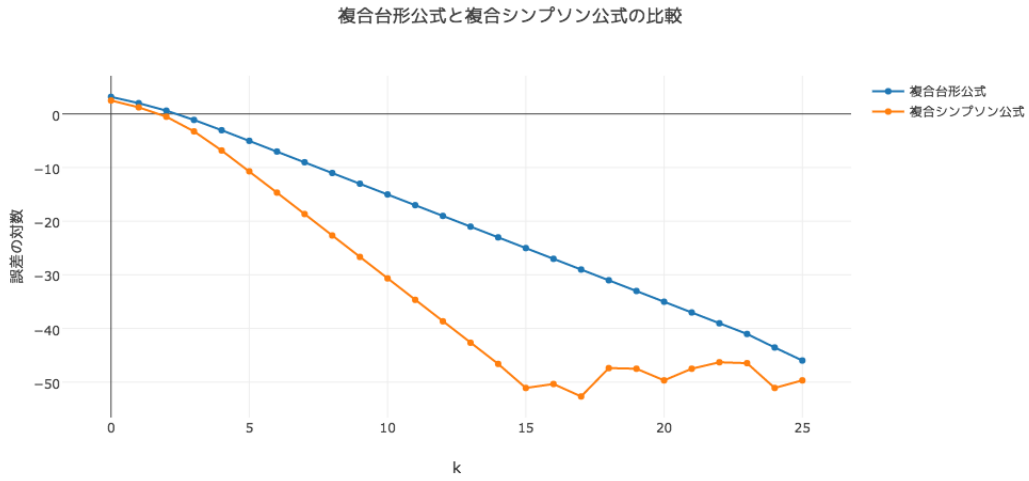

図 2　k と相対誤差の対数の関係

4.2　考察

4.2.1　指数的に誤差が減少していることについて

図 2 において、どちらの公式を用いても、誤差の対数は k に対して直線的に減少している。この直線の傾きを a とおくと、

$$\begin{aligned} log_2 E &= a \times k + c \\ \therefore E &= C \times (2^k)^a \\ &= Cn^a (\because n = 2^k) \end{aligned}$$

となり、誤差の大きさは、分割数 n の a 乗に従って減少していることがわかる。

ここで、複合台形公式の場合は、誤差の大きさ E が小区間の幅 h の平方に比例することから (参考文献 1)、E は n^{-2} に比例するとわかる (h は n の逆数に比例)。また、複合シンプソン公式の場合は、E は n^{-4} に比例する項で抑えられる (参考文献 2)。

そして、図 2 の直線部分 $5 \leq k \leq 15$ から最小二乗法にて傾きを求めると、複合台形公式の傾きは $a = -2.00$、複合シンプソン公式の傾きは $a = -4.02$ であり、これは上記に一致している。

4.2.2　複合公式の別の実装

複合台形公式は、積分範囲を細かく分け、それぞれに対して台形公式から積分値を求め、その和を取るというアルゴリズムである。複合シンプソン公式も同様である。このことから、台形公式やシンプソンの公式を、細かく分けた区間に対して適用し、その和を取ることでも、複合台形公式と複合シンプソン公式のアルゴリズムが実装できるのではないかと考えた。

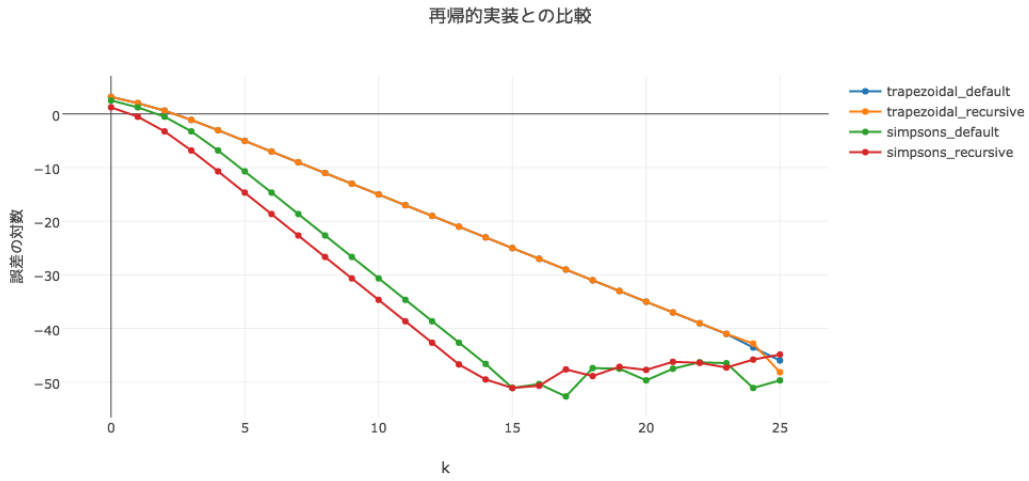

図 3　元の実装との比較

実際に、別の方法で実装したのが、src/Integrator/Recursive.hs である (再帰で実装しているわけではないが...)。compositTrapezoidalRule 関数は、section 関数によって積分区間を細かく分けた後、それぞれの区間に対して trapezoidalRule 関数を適用して、その計算結果の合計を取る、というふうに実装されている。compositSimpsonsRule 関数も同様である。

テキストの式を実装した複合公式と、別の方式で実装した複合公式のそれぞれを用いて、課題 4.2 と同様に積分 $\int_0^1 x^{19}dx$ を求め、k と誤差の対数の関係をグラフに描いたのが図 3 である。どちらの実装でも、概ね同様の精度で積分計算が行えていることがわかる (複合台形公式については、グラフの上ではほぼ完全に重なって見えている)。

5　課題 4.3

以下に示された定義式 (1)〜(4) を用いて、π を求めた。このとき、積分計算の手法として、分割数 $n = 2^k$ の複合台形公式と複合シンプソン公式をそれぞれ用いた。

$$\frac{\pi}{4} = \int_0^1 \frac{1}{1+x^2}dx \tag{1}$$

$$\frac{\pi}{4} = \int_0^1 \sqrt{1-x^2}dx \tag{2}$$

$$\frac{\pi}{12} + \frac{\sqrt{3}}{8} = \int_0^{1/2} \sqrt{1-x^2}dx \tag{3}$$

$$\frac{\pi}{6} - \frac{\sqrt{3}}{8} = \int_{1/2}^1 \sqrt{1-x^2}dx \tag{4}$$

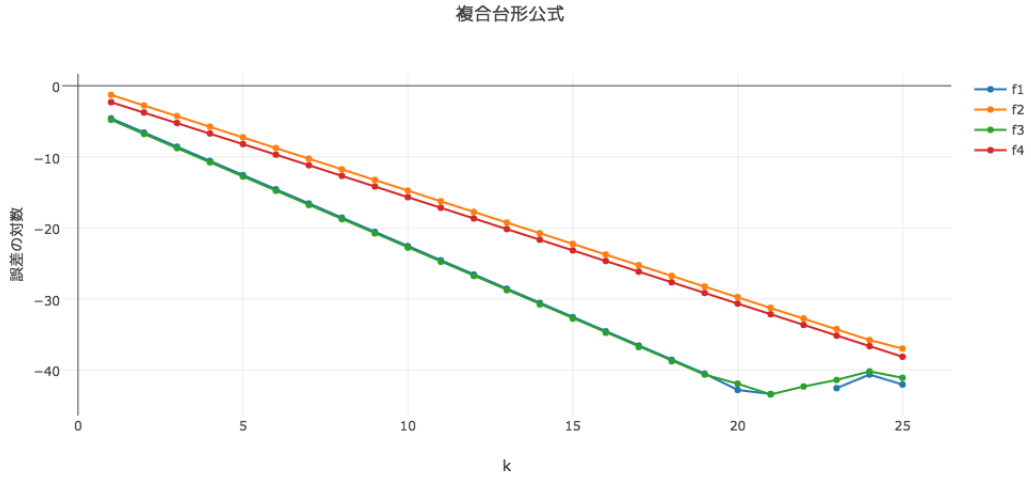

図 4　複合台形公式による計算

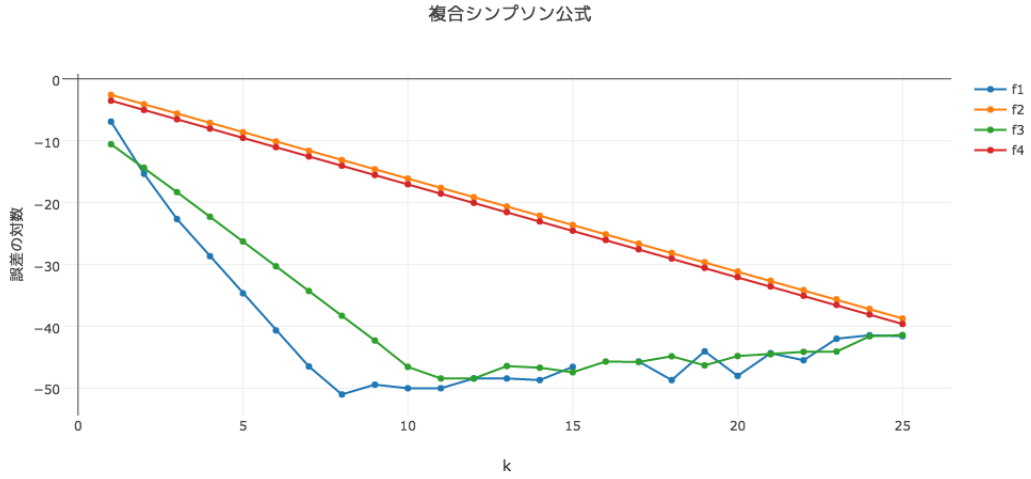

図 5　複合シンプソン公式による計算

5.1　結果

図 4 と図 5 はそれぞれ、複合台形公式と複合シンプソン公式によって、定義式 (1)〜(4) から π を求めた際の、真の値からの誤差の対数と k の関係をグラフに描いたものである。

いずれの場合も、定義式 (1) と (3) を用いた場合の方が、定義式 (2) や (4) を用いた場合に対して精度が高い。

5.2 考察

5.2.1 (1) と (3) の精度が高いことについて

図 4 と図 5 を見ると、積分計算の手法によらず、定義式 (1) と (3) によって π を求めたときのほうが、精度が高いことがわかる。

これは、積分対象関数の値の変動が大きい区間で積分を行なうと、微小な「短冊」を足し合わせる際に情報落ちが発生することが原因と考えられる。

実際、(2) や (3) や (4) の定義式の積分部分の積分対象関数は $\sqrt{1-x^2}$ であり、これを微分すると、$f'(x) = -\frac{x}{\sqrt{1-x^2}}$ となり、$x = 1$ 付近で関数値が大きく変動することがわかる。そして、(2) や (4) は積分区間に $x = 1$ を含む。一方、(3) は、積分区間に $x = 1$ を含まない。

また、(1) の積分対象関数は $\frac{1}{1+x^2}$ であり、これを微分すると、$f'(x) = -\frac{2x}{(x^2+1)^2}$ となり、積分区間の中では関数値が大きく変わらないことがわかる。

6 付録

6.1 ソースコード

ソースコードは別に添付したディレクトリ./haskell/以下にある。

6.2 出力結果

出力結果は、別に添付したディレクトリ./output/以下にある。

7 参考文献

1. 台形公式と誤差 (https://goo.gl/JpaCL9)
2. シンプソンの公式の誤差 (https://goo.gl/pU0qdx)