数値積分法

定積分の近似値

定積分の値は、不定積分がわかれば簡単に計算できる[*1]。しかし、不定積分が求められない場合も多い。たとえば、

$$\int e^{-x^2}dx$$

を求めようと思っても、初等関数では表せないのだ (初等関数以外では表せる場合もあるが)。

ここでは、近似公式や計算機を使って定積分の近似値を求める方法 (**数値積分法**) を簡単に紹介する。詳しく知りたい場合は専門書等を見てくだせえ。

定積分 $\int_a^b f(x)dx$ の近似値は、積和 $S_n = \sum_{k=1}^{n} f(\zeta_k)\Delta_k$ によって計算できる。n 個の' 帯' に分けるとき、その帯をどのように近似するかによって、種々の公式を得られる。

0.1 台形公式

曲線 $y = f(x), x = a, x = b$ x 軸で囲まれる面積を、幅 $h = (b-a)/n$ の n 個の帯に分割する。i 番目の帯の上部は弧 $P_{i-1}P_i$ であるが、これを直線で結び、面積を

$$\frac{1}{2}hf(a+(i-1)h) + f(a+ih) \tag{1}$$

で近似する。各々の帯の面積をこのように近似して、**台形公式**

$$\int_a^b f(x)dx \fallingdotseq \frac{h}{2}(f(a)+f(a+h)) + \frac{h}{2}(f(a+h)+f(a+2h)) + \cdots + \frac{h}{2}(f(a+(n-1)h)+f(b)) \tag{2}$$

$$\fallingdotseq \frac{h}{2}(f(a)+2f(a+h)+2f(a+2h)+\cdots+2f(a+(n-1)h)+f(b)) \tag{3}$$

を得る。

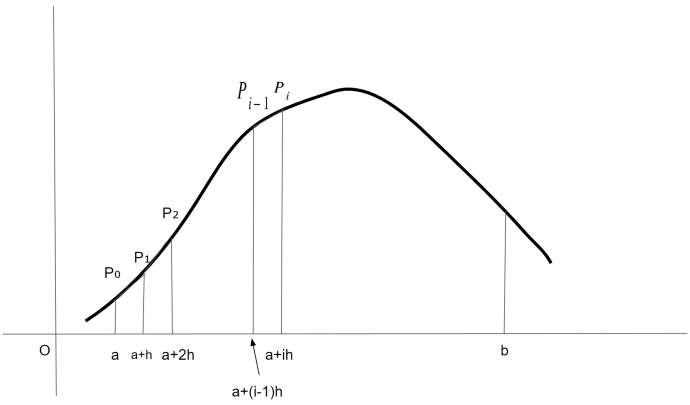

図 1　台形公式

[*1] 微積分学の基本定理より $F(b) - F(a)$

0.2　シンプソンの公式

今度は、面積を幅 $h = (b-a)/2m$ の $n = 2m$ 個の帯に分割する。その帯を 2 個ずつまとめて、弧 $P_0P_1P_2, P_2P_3P_4, \cdots, P_{2m-2}P_{2m-1}P_{2m}$ を持つ m 個の帯を作る[*2]。i 番目の帯の上部は弧 $P_{2i-1}P_{2i-1}P_{2i}$ であるが、これを通る放物線 $y = Ax^2 + Bx + C$ で近似すると、面積は

$$\frac{h}{3}(f(a+2(i-2)h) + 4f(a+(2i-1)h) + f(a+2ih)) \tag{4}$$

となる (下の例題)。m 個の帯の面積をこのように近似して、**シンプソンの公式**

$$\begin{aligned}\int_a^b f(x)dx \fallingdotseq \frac{h}{3}&(f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + 2f(a+4h) \\ &+ \cdots + 2f(a+(2m-2)h) + 4f(a+(2m-1)h) + f(b))\end{aligned} \tag{5}$$

を得る。

例題 1

3 点 $P_0(a, y_0), P_1(\frac{a+b}{2}, y_1), P_2(b, y_2)$ を通る放物線 $y = Ax^2 + Bx + C$ に対して、

$$\int_a^b ydx = \frac{b-a}{6}(y_0 + 4y_1 + y_2)$$

を示せ。

解答

定積分の値は

$$\int_a^b ydx = \int_a^b (Ax^2 + Bx + C)dx = \frac{b-a}{3}(A(a^2 + ab + b^2) + \frac{3}{2}B(a+b) + 3C)$$

$y = Ax^2 + Bx + C$ は 3 点 P_0, P_1, P_2 を通るから、

$$\begin{aligned}y_0 &= Aa^2 + Ba + C \\ y_1 &= A(\frac{a+b}{2})^2 + B\frac{a+b}{2} + C \\ y_2 &= Ab^2 + Bb + C\end{aligned}$$

よって、

$$y_0 + 4y_1 + y_2 = 2(A(a^2 + ab + b^2) + \frac{3}{2}B(a+b) + 3C)$$

だから、

$$\int_a^b ydx = \frac{b-a}{6}(y_0 + 4y_1 + y_2)$$

[*2] 残念ながら図はない。作るのが面倒だった。興味があったらインターネットで調べて。

電卓を用いて、次の例題を解いてみよう。

例題 2

定積分 $\int_0^1 e^{-x^2}dx$ を $n = 10$ の、(a) 台形公式、(b) シンプソンの公式、を用いて計算せよ。

解

$h = 0.1$ である。あらかじめ、$f(x) = e^{-x^2}$ の値を計算しておく。(表でまとめると便利[*3])

表 1　e^{-x^2} の値

x	0	0.1	0.2	0.3	0.4
f(x)	1	0.990049834	0.960789439	0.913931185	0.852143789

x	0.5	0.6	0.7	0.8	0.9	1
f(x)	0.778800783	0.697676326	0.612626394	0.527292424	0.444858066	0.367879441

(a) 台形公式 (3) より

$$\begin{aligned}\int_0^1 e^{-x^2}dx &= \frac{0.1}{2}(f(0)+2f(0.1)+2f(0.2)+2f(0.3)+2f(0.4)+2f(0.5)+2f(0.6)+2f(0.7)+2f(0.8)+2f(0.9)+f(1))\\ &= 0.746211\end{aligned}$$

(b) シンプソンの公式 (5) より

$$\begin{aligned}\int_0^1 e^{-x^2}dx &= \frac{0.1}{3}(f(0)+4f(0.1)+2f(0.2)+4f(0.3)+2f(0.4)+4f(0.5)+2f(0.6)+4f(0.7)+2f(0.8)+4f(0.9)+f(1))\\ &= 0.746825\end{aligned}$$

なお、正確な値を小数第六桁まで書くと 0.746825 である。

問題

以下それぞれ (a) 台形公式と (b) シンプソンの公式を用いて計算し、正しい積分値と比べよ。

1. $\int_2^6 \frac{dx}{x}$ $\cdots((a)n = 4, (b)n = 4)$
2. $\int_0^{\frac{1}{2}} \frac{dx}{1+x^2}$ $\cdots((a)n = 5, (b)n = 4)$

[*3] 余談だが、表 1 を描画するのにすごく苦労した

最後に

今回は数値積分法について軽く触れた。不定積分が求められなくても、定積分の値なら求められることがあることがわかっただろう。

次回は関数の展開とオイラーの公式について触れる。先にネタバレをしてしまうと、関数の展開はその名前の通り関数を有理関数の和に展開することである。例えば $\sin x$ を展開すると

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots + \frac{(-1)^{n-1}x^{2n-1}}{(2n-1)!} + R_{2n+1}, R_{2n+1} = (-1)^n \frac{x^{2n-1}}{(2n-1)!}\cos\theta x \cdots (0 < \theta < 1)$$

となる。なぜこのようになるのかは次回の楽しみにしておいてほしい。
オイラーの公式はこの関数の展開を利用して導くもので、式は

$$e^{ix} = \cos x + i \sin x$$

と、三角関数の和が指数関数になってしまうのである。アメリカの物理学者リチャード・ファインマンはこの式を**人類の至宝**といったらしい。ちなみに $x = \pi$ とすると直ちに

$$e^{i\pi} + 1 = 0$$

が成り立つ[*4]。またこの公式を用いれば、有名なド・モアブルの公式も簡単に示せてしまうのだが、それは次の機会に取っておこう。

話がそれてしまったが、残すところ積分も最後である (まぁ二年生になったら多変数の微分積分をやるのだが)。気を引き締めていきたい。[*5]

[*4] $\cos\pi = -1, \sin\pi = 0$ より

[*5] メモ：(制作時間 約 3 時間) 正月早々なにやってんだ

おまけ

数値積分のソースコード。問題 (1) の計算を Java(OpenJDK17) で実装した。

Listing 1　Main.java

```
1  package javas.lab;
2
3  import javas.maths.*;
4
5  public class Main {
6      public static void main(String[] args) {
7          System.out.println(integral.simpson_rule(new Function() {
8              @Override
9              public double func(double x) {
10                 return 1/x;
11             }
12             @Override
13             public boolean isEvenFunction() {
14                 return false;
15             }
16             @Override
17             public boolean isOddFunction() {
18                 return true;
19             }
20         }, 2, 6, 4));
21     }
22
23 }
```

Listing 2　Function.java

```
1  package javas.maths;
2
3  public interface Function {
4      public double func(double x);
5      // 偶関数かどうか
6      public boolean isEvenFunction();
7      // 奇関数かどうか
8      public boolean isOddFunction();
9  }
```

Listing 3　integral.java

```
1     package javas.maths;
2
3  public class integral {
4     /**
5      *
6      * @param fx 被積分関数　:
7      * @param begin 積分下限:
8      * @param end 積分上限　:
9      * @param n 帯の数　多ければ多いほどよい近似値が得られる:
10     * @return 台形公式による定積分の近似値
11     */
12    public static double trapezoidal_rule(Function fx,
13                                          double begin,double end,int n){
14        if(begin==end)return 0;
15        if(fx.isOddFunction()&&begin==-end)return 0;
16
17        double sum=0;
18        double width=(end-begin)/n;
19
20        for(double i=begin;i<=end;i+=width){
21            sum+=fx.func(i);
22            if(!(i==begin||i==end))sum+=fx.func(i);
23        }
24        sum*=width/2;
25        return sum;
26    }
27    /**
28     *
29     * @param fx 被積分関数　:
30     * @param begin 積分下限:
31     * @param end 積分上限　:
32     * @param n 帯の数　多ければ多いほどよい近似値が得られる:
33     * @return シンプソンの公式による定積分の近似値
34     */
35    public static double simpson_rule(Function fx,
36                                      double begin,double end,int n) {
37        if(begin==end)return 0;
38        if(fx.isOddFunction()&&begin==-end)return 0;
```

```
39
40          double sum=0;
41          double width=(end-begin)/n;
42
43          for (int i = 0; i < n+1; i++) {
44              double temp;
45              temp=fx.func(begin+i*width);
46              if(!(i==0||i==n))temp*= i%2==0?2:4;
47              sum+=temp;
48          }
49          sum*=width/3;
50          return sum;
51      }
52  }
```

シンプソンの公式を使っているが、台形公式でも計算できる。その場合は 12 行目の「trapezoidal」を「simpson」に変えるだけだ。

Main クラスの main メゾット内にある匿名クラス宣言しているコードを書き換えれば、別の関数でも計算することができる。

例えば、e^x を計算したいとしたら

```
    ...
    public static void main(String[] args) {
        System.out.println(integral.simpson_rule(new Function() {
            @Override
            public double func(double x) {
                return Math.exp(x);
            }
            @Override
            public boolean isEvenFunction() {
                return false;
            }
            @Override
            public boolean isOddFunction() {
                return false;
            }
        }, 2, 6, 4));
    }
    ...
```

とすればよい。このとき isEvenFunction() と isOddFunction() を書き換えることを忘れないようにしよう。

正直に言って、Java は数値計算するための言語ではないが、残念なことに私は Java 以外の言語はあまり触れたことがないのだ。[*6]

このソースコードは自由に改変してもらって構わない。もしコードに間違いがあったら指摘してほしい。[*7] ソースコードは src フォルダにおいておく。

[*6] Python を使えば簡単にできるだろうが、Python のほうが Java より動作が遅い！

[*7] コードが汚いとしても黙っていてほしい（笑）