コンピュータ

北海道大学理学部数学科

数値積分

関数 $f(x)$ を区間 $[a, b]$ で定積分するには、公式

$$\int_a^b f(x)dx = F(b) - F(a)$$

を用いる。原始関数 $F(x)$ を求めなければならない。
しかし、原始関数を常に計算できるとは限らないのだった。定積分は関数 $y = f(x)$ と x 軸の間の図形の面積であるから、区分求積法をもとにしたプログラムによって定積分の値を求める。

数値積分

原始関数を解析的に求めることの不可能な関数 $f(x)$ について、定積分 $\int_a^b f(x)dx$ を計算機プログラムを用いた数値計算によって近似値を求める手法を数値積分と呼ぶ。原始関数を求められない関数の例は $exp(-x^2)$, $\frac{sin(x)}{x}$ などであった。
数値積分の手法として、直感的に理解しやすい台形則と、精度よく計算できるシンプソン法を扱う。また、確率的な手法であるモンテカルロ法を扱う

- 台形則
- シンプソン法
- モンテカルロ法

実用的には目的に応じて許容する精度が決まってくる。精度に応じた近似値を数値計算によって求めることが重要である。

台形則

$\int_a^b f(x)dx$ を小区間 $[c, c+h]$ では台形の面積

$$\frac{f(c)+f(c+h)}{2}h$$

で近似する手法を台形則と呼ぶ。
区間 $[a, b]$ を幅 h の小区間に n 分割するなら $h=(b-a)/n$ であり、k 番目の小区間は $[a+kh, a+(k+1)h]$ となる。
$0 \leq k \leq n-1$ である。
従って、定積分

$$\int_a^b f(x)dx = \sum_{k=0}^{n-1} \int_{a+kh}^{a+(k+1)h} f(x)dx$$

は台形則によって次のように書くことができる。

$$\sum_{k=0}^{n-1} \frac{f(a+kh)+f(a+(k+1)h)}{2}h$$

台形則

$$\sum_{k=0}^{n-1} \frac{f(a+kh)+f(a+(k+1)h)}{2}h$$

では $k=0$ と $k=n-1$ 以外では $f(a+kh)$ を二回計算しているので、この和を整理すると次のプログラムのようになる。

$$\sum_{k=0}^{n-1} \frac{f(a+kh)+f(a+(k+1)h)}{2}h = \frac{f(a)+f(b)}{2}h+\sum_{k=1}^{n-1} f(a+kh)h$$

台形則の誤差

まず次の式を計算しておく。小区間での台形の斜辺を表す直線と $f(x)$ の差であり、端点で一致していることに注意する。

$$f(c) + (f(c+h) - f(c))(x-c)/h - f(c + (x-c))$$

テイラー展開を利用して次を得る。

$$f''(c)(x-c)h/2 - f''(c)(x-c)^2/2 + o(h^3)$$

$[c, c+h]$ で積分すると次を得る。つまり、この段階で台形則は h^3 程度の誤差を含む。

$$f''(c)h^3/4 - f''(c)h^3/6 = f''(c)h^3/12 = f''(c)(b-a)^3 n^{-3}/12$$

n 分割する場合、小区間の和は n 項の和だから誤差は n^{-2} 程度となる。

台形則を使い、真の値がわからない場合にプログラムを停止させる条件を考える。
前回のプログラムは、あらかじめ定めた分割数 n に対して数値積分した結果を返すプログラムだった。実際に与えられる条件は真の値と数値積分結果との差を精度とすることが多いから、この要請に応えられるプログラムとする。

1. 定積分の真の値に近い値（近似値）を求めるために、収束する条件を考える。分割数が n である時の値と n+1 である時の値が小さくなった時に止めよう。
2. 定積分の真の値と、プログラムでの数値計算結果との間で許容できる精度をあらかじめ与えておく。精度を格納する変数名を eps とする。
3. プログラム中で n を自動的に増やしながら計算する。
4. 真の値と計算結果との差が精度を下回ったら計算を止めることにしたい。

$f(x) = x^2$ を 0 から 1 まで定積分するとして、真の値は 1/3 である。真の値が分かっていれば次のようなプログラムが書ける。

```
import math
def f(x):
  return(x*x)

eps=1.0e-3
err=10
a=0.0
b=1.0
n=1
h=(b-a)/n
while(err>eps):
  h=(b-a)/n
  s=(f(a)+f(b))*h/2
  for k in range(1,n):
    s=s+f(a+k*h)*h
  err=math.fabs(s-1.0/3)
  print(n,s,err)
  n=n+1
```

分割数 n の場合の s の値と分割数 n+1 の場合の s の値の差が精度を下回るまで n を増やして計算を続けるように変更する。

```
import math
def f(x):
  return(x*x)
eps=1.0e-7
err=10
a=0.0
b=1.0
n=1
h=(b-a)/n
s0=(f(a)+f(b))*h/2
while(err>eps):
  n=n+1
  h=(b-a)/n
  s=(f(a)+f(b))*h/2
  for k in range(1,n):
    s=s+f(a+k*h)*h
  err=math.fabs(s-s0)
  print(n,s,err)
  s0=s
```

しかし、プログラム例を実行してみると、次の出力結果となる。

```
141 0.3333417165467867 4 1.201879071266454e-07
142 0.333341598889109 3 1.176576774342130 7e-07
143 0.3333414836911338 7 1.151979754365761 7e-07
144 0.3333413708847736 7 1.128063601973927 6e-07
145 0.333341260404280 5 1.104804931428837 7e-07
146 0.3333411521861512 3 1.082181292910711 1e-07
147 0.333341046169034 5 1.060171167521772 9e-07
148 0.3333409422936451 4 1.038753893456423 3e-07
149 0.3333408405026800 4 1.017909651013226 8e-07
150 0.333340740740740 8 9.976193920957499e-08
```

1. プログラムは停止しているのに、必要な精度である 7 桁が得られていない。
2. f(x) の計算回数が多い。

改良するために、分割の取り方を変更する。

- $n = 149$ と $n = 150$ の出力を比較すると、両者が 6 桁まで一致しており、差を取ることで見かけ上の誤差が精度を下回った。これはまずい。
- 分割数 n での計算と $n + 1$ での計算の間に関連性がない点に問題がある。n での計算結果を $n + 1$ でもうまく利用して、計算の効率を上げたい。（計算の効率とは、計算回数を減らすということである。）

具体的に改良する

分割数を 2^n とする。分割数 2^{n+1} の場合の各小区間の端点は、2^n の場合の小区間の端点と一致しているか、中点となっているかのどちらかである。次の式が成立している。分割数 2^n の場合の和をうまく利用して、分割数 2^{n+1} の場合の和を効率よく計算している。

$$\sum_{k=1}^{2^{n+1}-1} f(a+k*h)*h = \sum_{k=1}^{2^n-1} f(a+k*(2*h))*(2*h)/2 + \sum_{k=1}^{2^n-1} f(a+(2*k-1)*h)*h$$

プログラム例

```
import math
def f(x):
  return(x*x)

n=1
a=0
b=1
h=(b-a)/n
eps=1e-7
err=10
s0=(f(a)+f(b))/2
s=0
```

プログラム例

```
while(err>eps):
  n=n*2
  h=h/2
  s=s0/2
  k=1
  while(k<n):
    s=s+____
    k=k+2
  err=math.fabs(s-s0)
  s0=s
  print(n,s,err)
```

実行結果

```
2 0.375000000 0.125000000
4 0.343750000 0.031250000
8 0.335937500 0.007812500
16 0.333984375 0.001953125
32 0.333496094 0.000488281
64 0.333374023 0.000122070
128 0.333343506 0.000030518
256 0.333335876 0.000007629
512 0.333333969 0.000001907
1024 0.333333492 0.000000477
2048 0.333333373 0.000000119
4096 0.333333343 0.000000030
```

シンプソン法

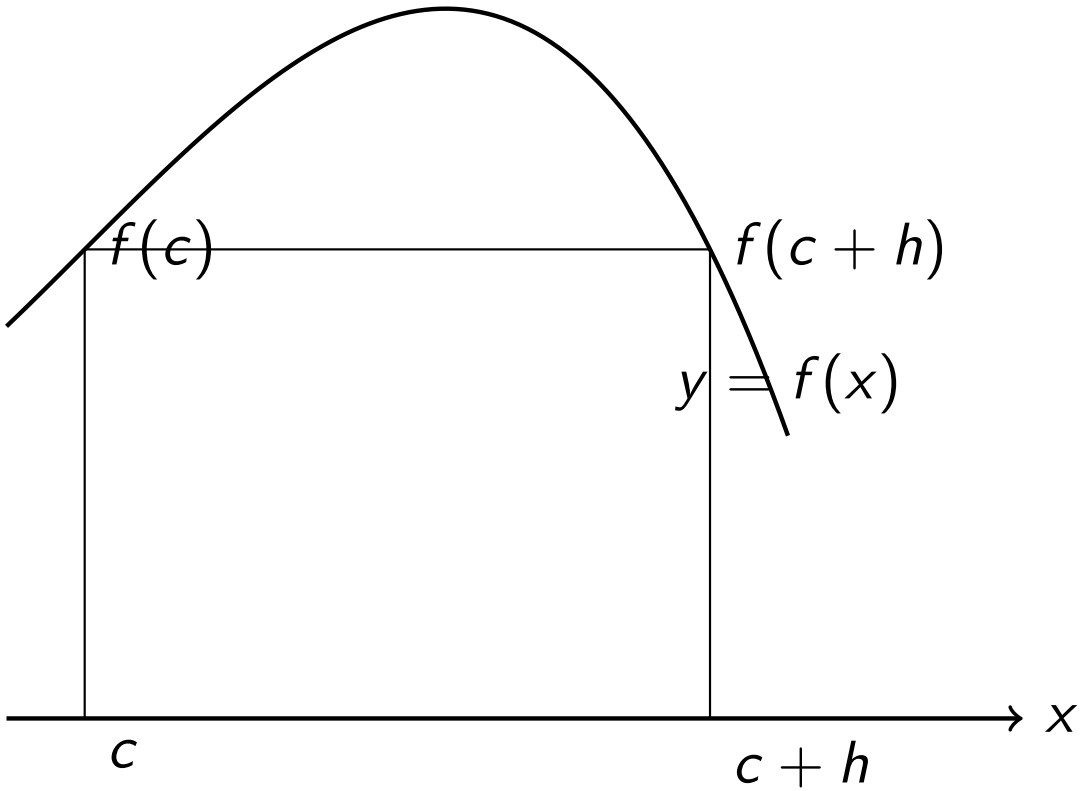
台形則よりも効率の良いシンプソン法によって数値積分を与えるプログラムを作成する。

台形則は小区間 $[c, c+h]$ の積分値を台形の面積

$$(f(c) + f(c+h)) * h/2$$

で近似した。二点 $(c, f(c)), (c+h, f(c+h))$ を通る直線で $f(x)$ を近似したことになっている。この直線の方程式は次の通り。
$[c, c+h]$ で積分すると台形の面積が出る。

$$y = \frac{f(c+h) - f(c)}{c+h-c}(x-c) + f(c)$$

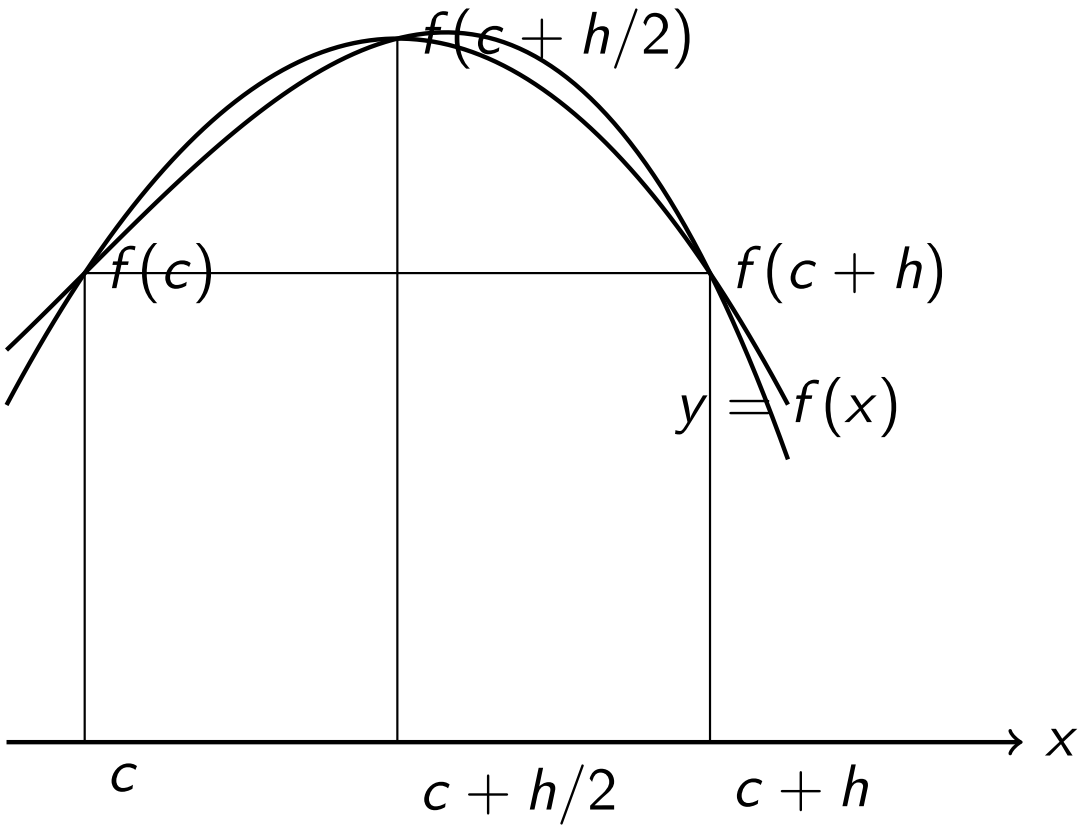

$f(c)$
$f(c+h)$
$y = f(x)$
x
c
$c+h$

シンプソン法では、小区間 $[c, c+h]$ で $\int_c^{c+h} f(x)dx$ を三点 $(c, f(c)), (c+h/2, f(c+h/2)), (c+h, f(c+h))$ を通る二次関数の積分で近似する。この二次関数は下記の通り、ルジャンドル補間から決まるものである。

$$f(c+h)\frac{(x-c)(x-(c+h/2))}{(c+h-c)(c+h-(c+h/2))}$$

$$+f(c+h/2)\frac{(x-c)(x-(c+h))}{(c+h/2-c)(c+h/2-(c+h))}$$

$$+f(c)\frac{(x-(c+h)(x-(c+h/2)}{(c-(c+h))(c-(c+h/2))}$$


$f(c+h/2)$
$f(c)$
$f(c+h)$
$y=f(x)$
x
c
$c+h/2$
$c+h$

上記の二次関数を $[c, c+h]$ で積分すると次の式を得る。

$$(f(c) + 4f(c + h/2) + f(c + h))h/6$$

従って、次の和を計算すれば良い。

$$\int_a^b f(x)dx = \sum_{k=0}^{n-1}(f(a+kh)+4f(a+(k+1/2)h)+f(a+(k+1)h))h/6$$

台形則と同様、同じ項を繰り返し計算している部分を簡約し、次のように変形。

$$(f(a)+f(b))\frac{h}{6}+\sum_{k=1}^{n-1}2f(a+kh)\frac{h}{6}+\sum_{k=0}^{n-1}4f(a+(k+1/2)h)\frac{h}{6}$$

さらに、分割数を 2^n で増加させる。

$$(f(a)+f(b))\frac{h}{6}+\sum_{k=1}^{2^n-1}2f(a+kh)\frac{h}{6}+\sum_{k=0}^{2^n-1}4f(a+(k+1/2)h)\frac{h}{6}$$

$2^0 = 1$ 分割のとき、上の和は次のようになっている。($h = b - a$)

$$(f(a) + f(b))\frac{h}{6} + 0 + 4f(a + h/2)\frac{h}{6}$$

$2^1 = 2$ 分割のときは次のようになっている。($h = (b - a)/2$)

$$(f(a) + f(b))\frac{h}{6} + 2f(a + h)\frac{h}{6} + (4f(a + h/2) + 4f(a + 3h/2))\frac{h}{6}$$

1 分割の第二項までの和を se0, 第三項を so0 とおく。2 分割のときの和は、

1. 上式の第二項までの和 se について $se = se0/2 + so0/4$ である。
2. 第三項 so は和を計算する。

分割数が 4,8,16,... と増加しても同様である。

プログラム例

```
import math

def f(x):
  return(x*x)

n=1
a=0
b=1
h=(b-a)/n

se0=(f(a)+f(b))*h/6
so0=4*f((a+b)/2)*h/6
se=0
so=0

eps=1e-7
err=10
```

プログラム例

```
while(err>eps):
  h=h/2
  n=n*2
  se=se0/2+so0/4
  so=0
  k=0
  while(k<n):
    so=so+____
    k=k+1
  err=math.fabs(so+se-so0-se0)
  se0=se
  so0=so
  print(n,se+so,err)
```

モンテカルロ法

確率的な手法の典型例として、円周率を実験的に評価する手法の一つ「ビュフォンの針」を思い出そう。
半径 1m の円を描き、そこに短い針を無作為に投げ入れる。円内に入った針の本数を全ての針の本数で割ることで円周率を評価できる。

アルゴリズムとしてビュフォンの針を考えると次のようになる。

- 座標平面上に $f(x) = \sqrt{1 - x^2}$ のグラフを描く。
- $[0, 1] \times [0, 1]$ 内にランダムに点 (x_n, y_n) を置く。
- $f(x_n) > y_n$ となった点の数を総数で割ると $\pi/4$ の近似値を得る。

点の数を十分に大きくとる必要がある。

プログラム例

```
import random
import math
def f(x):
return(math.sqrt(1-x*x))
N=10000
s=0
r=list(range(N))
for k in range(1,N):
  rx=random.uniform(0,1)
  ry=random.uniform(0,1)
  if f(rx)>ry:
    s=s+1
  r[k]=s/k
print(k, s/k)
```

課題

$f(x) = \frac{1}{1+x^2}$ について、$I = \int_0^1 f(x)dx$ を精度 1.0×10^{-8} で数値積分する。$I = \pi/4$ であることを手計算によって確かめ、次のプログラムを作成せよ。

1. 台形則によって求めよ (3 点)。
2. シンプソン法で求めよ (4 点)。
3. プログラム例を参考にモンテカルロ法で求め、収束の様子をグラフに示せ (3 点)。モンテカルロ法の精度は 1.0×10^{-2} 程度で良い。