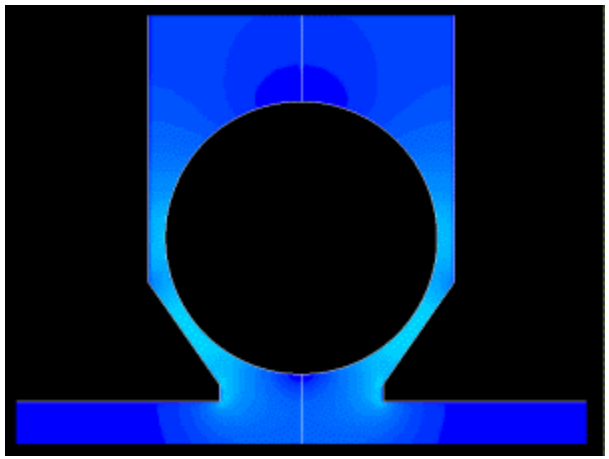
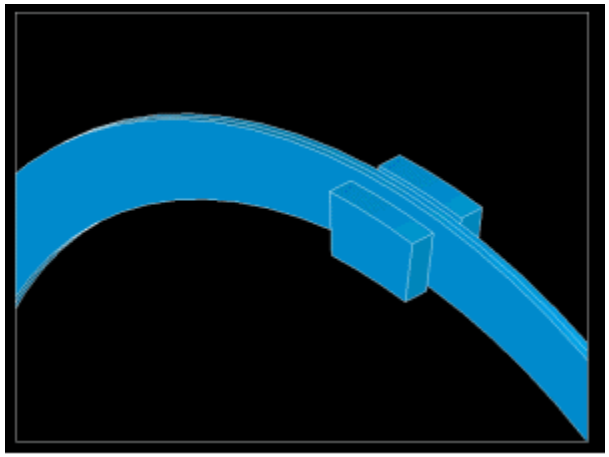
数値解析

2021年度前期　第7週 [2021年5月27日]





静岡大学
工学研究科機械工学専攻
計測情報講座
創造科学技術大学院
情報科学専攻

三浦 憲二郎

講義アウトライン [5月27日]

- 数値積分
  - ニュートン・コーツ公式
    - 台形公式
    - シンプソン公式
    - 重積分

数値積分の必要性 p.135

•初等関数（しょとうかんすう）とは、複素数を変数とする多項式関数・指数関数・対数関数主値の四則演算・合成によって表示できる関数である．三角関数や双曲線関数、そして両者の逆関数主値も初等関数と考えられる．

•微分

　•初等関数の導関数は必ず初等関数になる

•積分

　•初等関数の積分は初等関数であらわされるとは限らない

　例　フレネル積分

$$f(t) = \int_0^t \cos\frac{\pi t^2}{2} dt$$

$$g(t) = \int_0^t \sin\frac{\pi t^2}{2} dt$$

数値的に積分する以外方法がない！

ニュートン・コーツ公式 p.135

•定積分 $$I = \int_a^b f(x)dx$$

を求めるために，分点$a=x_0<x_1<x_2<\cdots<x_n=b$をとり，$f_k=f(x_k)$を通るラグランジュ補間多項式 $P_n(x)$ を考える．ただし，それぞれの分点は等間隔$h=(b-a)/n$で並んでいるとする．$P_n(x)$は多項式なので以下のI_nを計算することができる．

$$\begin{aligned} I_n &= \int_a^b P_n(x)dx \\ P_n &= \sum_{k=0}^{n} f_k l_k(x) \\ \int_a^b f(x)dx &\approx \int_a^b P_n(x)dx = \sum_{k=0}^{n} f_k \int_a^b l_k(x)dx = \sum_{k=0}^{n} \alpha_k f_k \\ \text{where } \alpha_k &= \int_a^b l_k(x)dx \end{aligned}$$

この近似積分公式をn+1点のニュートン・コーツ公式(Newton-Cotes)と呼ぶ．

台形公式　p.135

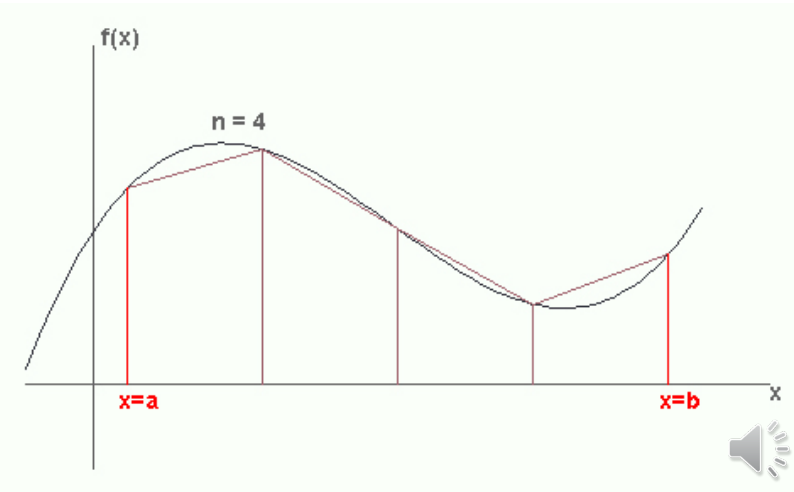
•ニュートン・コーツ公式　n=1 とする. h=b−a, x_0=a, x_1=b, x=a+sh

$$\alpha_0 = \int_a^b l_0(x)dx = \int_a^b \frac{x-x_1}{x_0-x_1}dx = -\frac{1}{h}\int_0^1 h(s-1)hds = \frac{h}{2}$$

$$\alpha_1 = \int_a^b l_0(x)dx = \int_a^b \frac{x-x_0}{x_1-x_0}dx = \frac{1}{h}\int_0^1 sh\cdot hds = \frac{h}{2}$$

Hence $$\int_a^b f(x)dx \approx \frac{h}{2}(f_0+f_1)$$

これを台形公式(Trapezoidal rule)という.



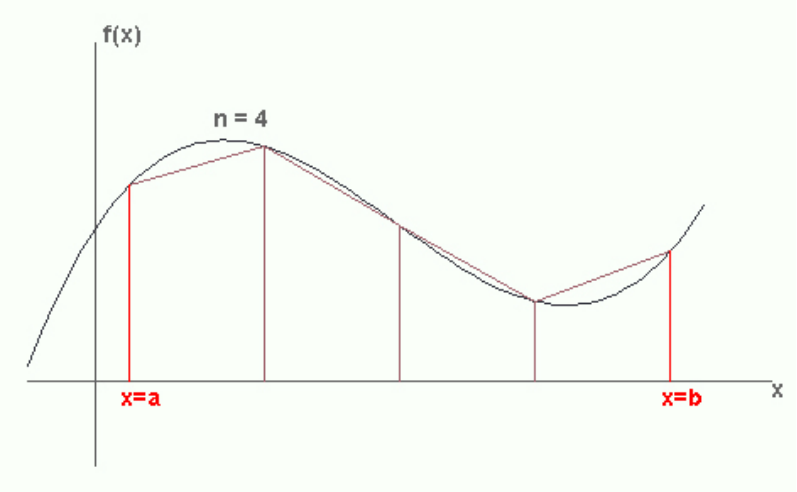
台形公式　p.136

•実際の計算　区間[a,b]を n 等分．その分点 x_k, 各小区間[x_k,x_{k+1}]に適用

$$\int_a^b f(x)dx = \sum_{k=0}^{n-1}\int_{x_k}^{x_{k+1}} f(x)dx \approx \sum_{k=0}^{n-1}\frac{h}{2}(f_k + f_{k+1})$$

$$= \frac{h}{2}\{f_0 + f_n + 2(f_1 + f_2 + \cdots + f_{n-1})\},\ h = \frac{b-a}{n}$$

これを複合台形公式，あるいは台形公式という．



台形公式:プログラム p.137

```
#include <stdio.h>

/* 関数の定義 */
double func1(double x);
double func2(double x);
/* 台形公式 */
double trapezoidal( double a, double b, int n, double (*f)(double) );

int main(void)
{
  int n=100;

  printf("2.0/(x*x)を[1,2]で積分します。分割数は%dです¥n", n);
  printf("結果は%20.15fです¥n",trapezoidal(1.0, 2.0, n, func1) );

  printf("4.0/(1+x*x)を[0,1]で積分します。分割数は%dです¥n", n);
  printf("結果は%20.15fです¥n",trapezoidal(0.0, 1.0, n, func2) );

  return 0;
}

/* 台形公式 */
double trapezoidal( double a, double b, int n, double (*f)(double) )
{
  double T, h;
  int i;

  h = ( b - a ) /n ;  /* 刻み幅の指定 */

  /* 台形公式 */
  T = ( (*f)(a) + (*f)(b) ) / 2.0;
  for ( i = 1; i < n; i++) T += (*f)( a + i*h );
  T *= h;

  return T;
}
```

```
 /* 関数の定義 */
double func1(double x)
{
  return( 2.0/(x*x) );
}

double func2(double x)
{
  return( 4.0 / (1.0+x*x) );
}
```

実行結果

2.0/(x*x)を[1,2]で積分します。分割数は100です

結果は　1.000029166020881

4.0/(1+x*x)を[0,1]で積分します。分割数は100です。

結果は　3.141575986923129

シンプソン公式　p.139

- ニュートン・コーツ公式　n=2 とする.

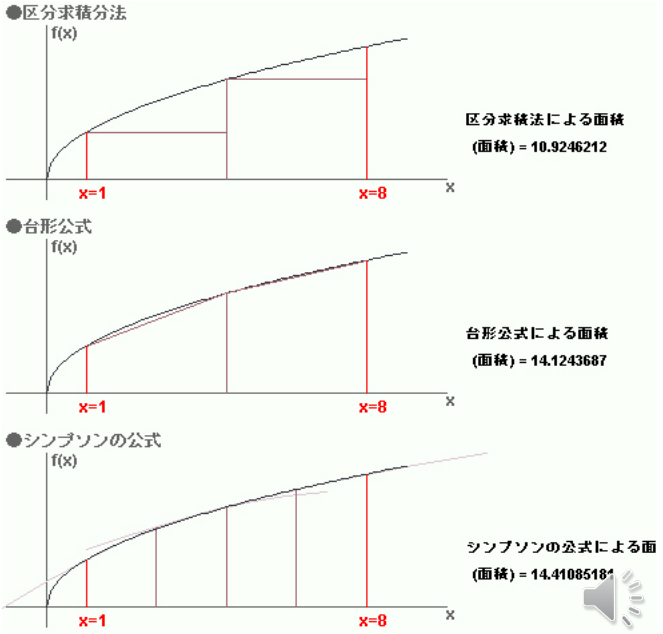
h=(b−a)/2, x_0=a, x_1=a+h, x_2=a+2h=b

$$\alpha_0 = \frac{1}{(x_0-x_1)(x_0-x_2)}\int_a^b (x-x_1)(x-x_2)dx = \frac{1}{2h^2}\int_0^2 (s-1)(s-2)h^3 ds = \frac{h}{3}$$

$$\alpha_1 = \frac{1}{(x_1-x_0)(x_1-x_2)}\int_a^b (x-x_0)(x-x_2)dx = \frac{4}{3}h$$

$$\alpha_2 = \frac{1}{(x_2-x_0)(x_2-x_1)}\int_a^b (x-x_0)(x-x_1)dx = \frac{h}{3}$$

Hence $$\int_a^b f(x)dx \approx \frac{h}{3}(f_0+4f_1+f_2)$$



これをシンプソン公式(Simpson's rule)という.

シンプソン公式　p.140

•実際の計算　区間[a,b]を 2n 等分．その分点 x_k, 各小区間[x_{2k},x_{2k+2}]に適用

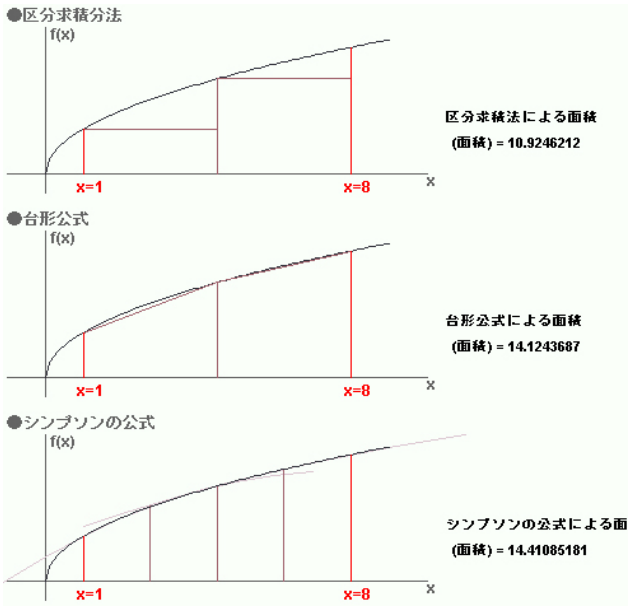
$$\int_a^b f(x)dx = \sum_{k=0}^{n-1}\int_{x_{2k}}^{x_{2k+2}} f(x)dx \approx \sum_{k=0}^{n-1}\frac{h}{3}(f_{2k}+4f_{2k+1}+f_{2k+2})$$

$$= \frac{h}{3}\{f_0+f_{2n}+4(f_1+f_3+\cdots+f_{2n-1})$$

$$+2(f_2+f_4+\cdots+f_{2n-2})\},\ h=\frac{b-a}{n}$$

これを複合シンプソン公式，

あるいはシンプソン公式という．



シンプソン公式:プログラム p.140

```
#include <stdio.h>

/* 関数の定義 */
double func1(double x);
double func2(double x);
/* シンプソン公式 */
double simpson( double a, double b, int n, double (*f)(double) );

int main(void)
{
  int n=50;
  printf("2.0/(x*x)を[1,2]で積分します。分割数は%dです¥n", 2*n);
  printf("結果は%20.15fです¥n",simpson(1.0, 2.0, n, func1) );

  printf("4.0/(1+x*x)を[0,1]で積分します。分割数は%dです¥n", 2*n);
  printf("結果は%20.15fです¥n",simpson(0.0, 1.0, n, func2) );

  return 0;
}

/* シンプソン公式 */
double simpson( double a, double b, int n, double (*f)(double) )
{
  double S, h;
  int i;

  h = ( b - a ) /( 2.0*n ) ;  /* 刻み幅の指定 */

  /* シンプソン公式 */
  S = ( (*f)(a) + (*f)(b) ) ;
  for ( i = 1; i < n; i++) {
    S += 4.0*(*f)( a + (2.0*i-1.0)*h ) + 2.0*(*f)( a + 2.0*i*h );
  }
  S += 4.0*(*f)( a + (2.0*n-1.0)*h );
  S *= h/3.0;

  return S;
}
```

```
 /* 関数の定義 */
double func1(double x)
{
  return( 2.0/(x*x) );
}

double func2(double x)
{
  return( 4.0 / (1.0+x*x) );
}
```

実行結果

2.0/(x*x)を[1,2]で積分します。分割数は100です

結果は　1.000000002582389

4.0/(1+x*x)を[0,1]で積分します。分割数は100です。

結果は　3.141592653589754

重積分　p.144

•重積分

$$I = \int_a^b \{\int_{\phi(x)}^{\psi(x)} f(x,y)dy\}dx$$

$$F(x) = \int_{\phi(x)}^{\psi(x)} f(x,y)dy$$

$$I = \int_a^b F(x)dx$$

•台形公式

$$I \approx T_n = \frac{h}{2}\{F_0 + F_n + 2(F_1 + F_2 + \cdots + F_{n-1}\},\ h = \frac{b-a}{n}$$

•シンプソン公式

$$I \approx S_{2n} = \frac{h}{3}\{F_0 + F_{2n} + 4(F_1 + F_3 + \cdots + F_{2n-1}\} + 2(F_2 + F_4 + \cdots + F_{2n-2})\},\ h = \frac{b-a}{2n}$$

$$F(x_i) = \int_{\phi(x_i)}^{\psi(x_i)} f(x_i,y)dy$$

重積分　p.144

•台形公式　[$\phi(x_i), \Psi(x_i)$]　, $\phi(x_i)=y_0<y_1<\cdots<y_m=\Psi(x_i)$ とm等分,$k=(y_m-y_0)/m$

$$F(x_i) = \frac{k}{2}\{g_0 + g_n + 2(g_1 + g_2 + \cdots + g_{n-1}\},\ g_i = f(x_i, y_j)$$

•シンプソン公式

[$\phi(x_i), \Psi(x_i)$]　, $\phi(x_i)=y_0<y_1<\cdots<y_{2m}=\Psi(x_i)$ と2m等分, $k=(y_{2m}-y_0)/2m$

$$\begin{aligned} F(x_i) \ = \ & \frac{k}{3}\{g_0 + g_{2m} + 4(g_1 + g_3 + \cdots + g_{2m-1}\} \\ & +2(g_2 + g_4 + \cdots + g_{2m-2})\},\ k = \frac{y_{2m} - y_0}{2m} \end{aligned}$$

重積分:プログラム p.145

```
#include <stdio.h>
#include <stdlib.h>

/* 被積分関数の定義 */
double func(double x, double y);

/* yの積分区間 */
double phi(double x);
double psi(double x);

/* ベクトル領域の確保 */
double *dvector(int i, int j);
/* ベクトル領域の解放 */
void free_dvector(double *a, int i);

/* 重積分用の台形公式 */
double trapezoidal2( double a, double b, int m, int n, double (*p)(double),
                     double (*q)(double), double (*f)(double,double) );
int main(void){
  int n=20, m=20;
  printf("8x^2+4y を x=[1,2], y=[2-x,x^2] で積分します¥n");
  printf("xの分割数は%d, yの分割数%d, 結果は%15.10f¥n", m, n,
         trapezoidal2( 1.0, 2.0, m, n, phi, psi, func ) );
  return 0;
}

/* 重積分用の台形公式 */
double trapezoidal2( double a, double b, int m, int n, double (*p)(double),
                     double (*q)(double), double (*f)(double,double) )
{
  double T, h, k, *F, x, y1, y2;
  int i, j;

  F = dvector( 0, n );
  h = ( b - a ) /n ;          /* 刻み幅の指定(x方向) */
   /* F_i の計算 */
  for ( i = 0; i <= n; i++ ) {
    x = a + i*h;
    y1 = (*p)(x); y2 = (*q)(x);
    k = ( y2 - y1 )/m;        /* 刻み幅の指定(y方向) */
    F[i] = ( (*f)(x, y1) + (*f)(x, y2) ) / 2.0;
    for ( j = 1; j < m; j++ )  F[i] += (*f)(x, y1+j*k);
    F[i] *= k;
  }
  /* 積分の計算 */
  T = ( F[0] + F[n] ) / 2.0;
  for ( i = 1; i < n; i++) T += F[i];
  T *= h;
  free_dvector( F, 0 );
  return T;
}
/* 被積分関数の定義 */
double func(double x, double y) {
  return( 8.0*x*x + 4.0*y );
}
/* yの積分区間 */
double phi(double x) {
  return( 2.0-x );
}
double psi(double x){
  return( x*x );
}
double *dvector(int i, int j) /* a[i]～a[i+j]の領域を確保 */ {
  double *a;
  if ( (a=(double *)malloc( ((j-i+1)*sizeof(double)) ) == NULL ) {
    printf("メモリが確保できません(from dvector)¥n");
    exit(1);
  }
  return(a-i);
}
void free_dvector(double *a, int i)
{
  free( (void *)(a + i) );  /* (void *)型へのキャストが必要 */
}-
```

まとめ

- 数値積分
  - ニュートン・コーツ公式
    - 台形公式
    - シンプソン公式
  - 重積分