物理学情報処理演習

9. 数値計算

数値計算

方程式の解

数値積分

数値微分

参考文献

Numerical Recipes in C

W. H. Press他著　　技術評論社

Advanced Engineering Mathematics

Erwin Keryszig　　John Wiley & Sons, Inc.

大久保晋

E-mail: buturi-johoshori@tiger.kobe-u.ac.jp

http://extreme.phys.sci.kobe-u.ac.jp/staffs/okubo/lectures/Programming/index.html

数値計算

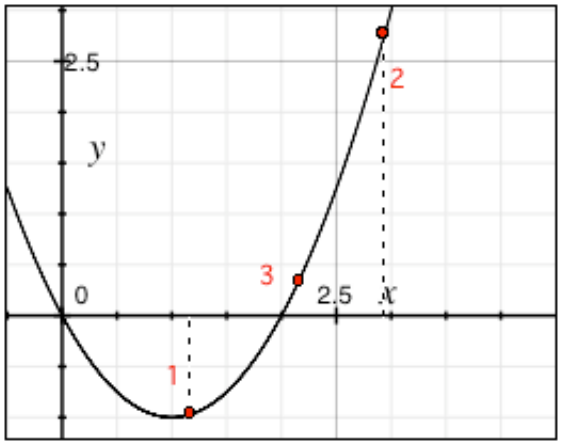
- 計算機が行える演算は、数値計算のみ。
- 具体的な解なしでは計算できない。
- 数値解は、解析的な解ではなく、数値を入れて解に最も近い値を探す方法

- 計算機の数値計算も万能ではない
  - 計算機内部で数値は２進数で表記される
  - 実数表現：浮動小数点　$s \times M \times B^{e-E}$
    - s: 符号ビット(+ or -)
    - M:仮数（正の整数）
    - B:基底(=2)
    - e:指数（符号なし整数）
    - E:指数のゲタ

    近似値としての表現
- 丸め誤差
- 打ち切り誤差

２次方程式の実解

$ax^2 + bx + c = 0$
の解を考えてみる。この解は
$y = ax^2 + bx + c$
がx軸と交わる点である。



次の３つの方法で解を求めてみよう

- やみくも法

  適当に決めたa, bの区間を十分細かい精度で分割し、その分割点における関数 y の値を調べ、十分ゼロに近い値を解とする。ただし十分の判断が難しいので前後で解の値の符号が変化することで解を判別する

関数が連続ならば解の前後で関数の値の符号は異符号（+/-）であることを利用して次の２つの方法

- 二分法

  「ある区間[a, b]の２つ端点（a点, b点）で関数 y が異符号である」ことが判っている場合、その区間の中点（ (a+b) / 2 ）での関数の値を調べ、同符号の端点と中点を入れ替える。

  入れ替えた新しい区間で同様の手続きを繰り返す。図では1-2区間、1-3区間と区間を縮める。

- 挟み撃ち法

  二分法の中点の代わりに、端点を結ぶ直線がx軸と交わる点を中点の代わりに採用する。（関数形を大雑把に1次式と近似する方法）

演習９−１Ａ：やみくも法

- やみくも法

  適当に決めたa, bの区間を十分細かい精度で分割し、その分割点における関数 y の値を調べ、十分ゼロに近い値を解とする。

  1. 方程式の係数を入力
  2. 区間a, bの値、その間の刻み幅を入力
  3. for文は整数値Loopなので(b-a)/刻み幅 で回す回数を求める
  4. y=a*x*x + b*x + c を計算
  5. y==0.0 あるいは　一つ前のyの値(y2) * y が負ならば解として解の配列 sol[] に代入
  6. 5.の評価が終わったらyを一つ前のyの値として y2 に代入
  7. やみくもLoopが終わったら答えを出力

次のページにある見本プログラム(9-1a.c)を入力して、コンパイルして

$$1.0x^2 - 2.0x + 0.0 = 0$$

のとき、区間[-10.0, 10.0]、刻み幅 0.001 で計算してみよう。

入力は 1.0 -2.0 0.0　リターン　　-10.0 10.0 0.001 リターンと入力

演習９－１Ａ： 9-1a.cサンプルプログラム 1/2

```
/* yamikumo method of solution for a quadratic equation */
#include <stdio.h>
#include <math.h>

#define hantei 0.001

int main(void)
{
          double a, b, c;  /* equation coefficients  */
          double area_a, area_b, step;  /* searching area */
          int i, j=0, kizami, nsol=0;
          double sol[2], soly[2];      /* there are two solutions */
          double x, y1=1000.0, y2=1000.0;

          /* input equation coefficients and searching area  */
          printf("input floating number of a,b,c of ax^2 + bx + c =0 \n");
          printf("by spaceing a b c\n");
          scanf("%lf %lf %lf", &a, &b , &c);
          printf("given equation is %lf x^2 + %lf x + %lf = 0\n", a, b, c);

          printf("input floating point number of start point, end point and step
\n");
          printf("by space start end step\n");
          scanf("%lf %lf %lf", &area_a, &area_b, &step);
          printf("search area is from %lf to %lf by step %lf\n", area_a, area_b,
step);
```

演習９−１Ａ： 9-1a.cサンプルプログラム 2/2

```
          /* Loop number count from area_a to area_b by step */
          kizami = round((area_b - area_a)/step);
          /* calculate y of starting point */
          y2 = a * area_a * area_a + b * area_a + c;

          /* search! */
          for (i=0; i<kizami; i++)
          {
                    x = area_a + step * i;
                    y1 = a * x * x + b * x + c;
                    printf("i/end= %d/%d y1= %lf \t y2=%lf y1*y2= %lf \r", i,
kizami, y1, y2, y1*y2);

                    /* solution exists at y==0.0 or between +y - -y */
                    if ( (y1 * y2)<0.0 || y1==0.0 )
                    {
                              sol[j] = x;
                              soly[j] = y1;
                              printf("\nFind! sol[%d]= %lf  soly[%d]=%lf \n", j,
sol[j], j, soly[j]);
                              j++;      /* include solution number */
                    }
                    y2 = y1;  /* transfer to next cell */
          }

          /* display solutions */
          printf("\n\nThe solutions are %lf, %lf \t y=%lf, %lf\n", sol[0], sol[1],
soly[0], soly[1]);

          return (0);
}
```

演習９−１Ａ：やみくも法

どれくらい計算に時間がかかったでしょうか？
プログラムを走らせて、ストップウォッチで時間計測する？
いえいえ、unixには計算結果の評価法があります。

time プログラム

でプログラムが消費した時間を表示してくれます。

`time ./9-1a` などとすると以下の感じで出力されます。

```
real    0m25.673s
user    0m0.150s
sys     0m0.050s
```

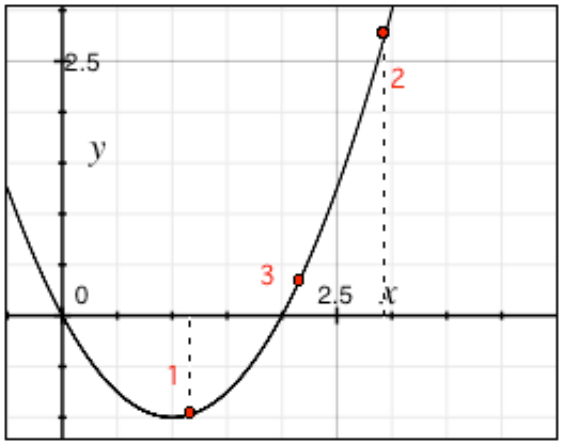
`real` は手で入力した部分まで含めて実際にかかった時間
`user` はプログラム `9-1a` が消費したCPUの計算時間
`sys` はUNIXシステムがこの実行に消費したCPUの時間

今の場合、やみくも法で消費したCPU時間は0.150秒ということになる。

演習９−１Ｂ：二分法

下図の[1, 2]区間の解を探す。

1. 方程式の係数を入力
2. 区間a, b の値、解の精度を入力
3. 入力されたa, bに対する関数の値*f(a), f(b)*が異符号になっているかチェック。異符号なら解が必ず存在する。
4. 二分法のループを行う。解が存在しないときのために繰り返しを50回としている
   - 二分点を求める(x)
   - 区間長(dx)を求める（絶対値）
   - もし区間長が解の精度(xacc)より小さければ解とする
   - 二分点の関数の値*f(x)*と端点*a'*の関数の値*f(a')*の符号を調べ、同符号の端点を二分点に置き換える
5. 解の出力



演習９−１Ｂ：二分法

次のページにある見本プログラム(9-1b.c)を入力して、、コンパイルして

$$1.0x^2 - 2.0x + 0.0 = 0$$

のとき、区間[0.5, 100.0]、刻み幅 0.00001 で計算してみよう。

入力は 1.0 -2.0 0.0　リターン　　0.5 100.0 0.00001 リターンと入力

どれくらいの計算時間がかかったでしょうか？

あっという間で、time表示で有効桁数以内の数値はでてこないと思います。

9-1b.c は２次関数は別呼び出しとしていますので、複雑な関数をいれてみましょう。（関数を明示的に書いた場合は、２次関数の係数入力は意味がありませんが、適当な値を与えましょう）

その際、Excelの表計算でグラフを書いてみて適当な区間を与える必要があります。

演習９−１Ｂ： 9-1b.cサンプルプログラム 1/2

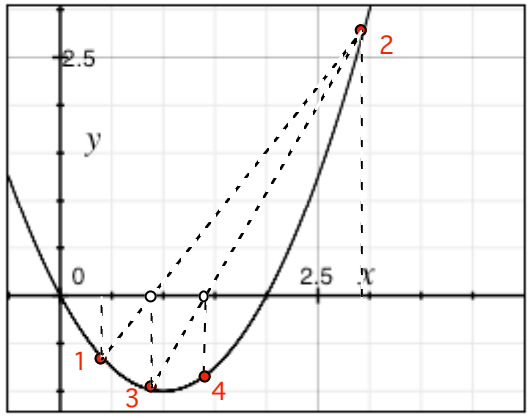
```
/* nibun method of solution for a quadratic equation */
#include <stdio.h>
#include <math.h>

#define NMAX 50

double quadratic(double a, double b, double c, double x);

int main(void)
{
          double a, b, c;  /* equation coefficients  */
          double area_a, area_b, xacc;  /* searching area and x_accuracy */
          int nloop;
          double x1, x2, x, dx;

                    /* input equation coefficients and searching area  */
          printf("input floating number of a,b,c of ax^2 + bx + c =0 \n");
          printf("by spaceing a b c\n");
          scanf("%lf %lf %lf", &a, &b , &c);
          printf("given equation is %lf x^2 + %lf x + %lf = 0\n", a, b, c);

          printf("input floating point number of start point, end point and step
\n");
          printf("by space start end x accuracy\n");
          scanf("%lf %lf %lf", &area_a, &area_b, &xacc);
          printf("search area is from %lf to %lf with solution accuracy %lf\n",
area_a, area_b, xacc);

          /* check bad range  f(area_a) x f(area_b) >0 */
          if ( quadratic(a, b, c, area_a) * quadratic(a, b, c, area_b) >=0.0 ){
                    printf("illegal section [%g, %g]\n", area_a, area_b);
                    return 1;
          }
```

演習９−１Ｂ： 9-1b.cサンプルプログラム 2/2

```
        x1 = area_a; x2 = area_b;
        for (nloop=0; nloop<NMAX; nloop++)
        {
                x = ( x1 + x2 )/2.0;
                dx = fabs(x2-x1);
                if (dx < xacc)
                {
                        printf("answer: %g \n", x);
                        return 0;
                }
                if ( quadratic(a, b, c, x1) * quadratic(a, b, c, x) < 0.0)
                {
                        x2 = x;
                }else{
                        x1 = x;
                }
        }

        printf(" can not find solution! \n");
        return 1;
}


double quadratic(double a, double b, double c, double x)
{
        double y;

        y = a * x * x + b * x + c;
        return y;
}
```

演習９−１C：挟み撃ち法

二分法では、区間の中点を考えたが、
その区間の端点を結ぶ直線がx軸と交わる点cを中点の代わりに用いる方法が挟み撃ち法という。



二分法では、いつでも半分ずつ近づくが、
挟み撃ち法では、直線近似で近づくため
より早く解に近づけられる。

- 二分法で選んだ中点を

$$x = a - \frac{(b-a)f(a)}{f(b)-f(a)}$$

に替えるだけである。

9-1b.c を改変し 9-1c.c として挟み撃ち法のプログラムを作って、動作を確かめよ。

２次方程式の実解：注意!

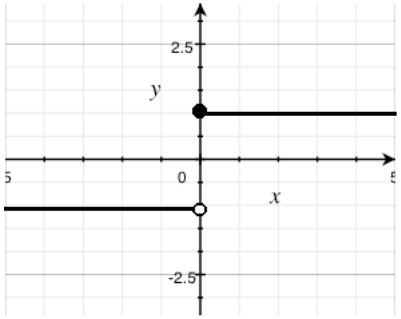
二分法、挟み撃ち法　では次の解は解けない

不連続関数

例）　$f(x) = -1$　　$x<0$

$f(x) = +1$　　$x \geq 0$

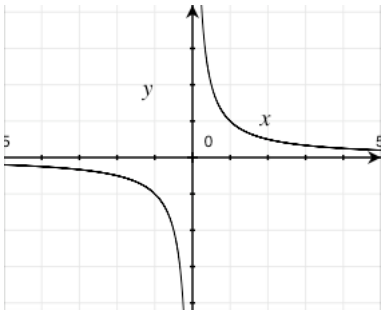
といった関数では、実際には解がないが、反復法（二分法、挟み撃ち法）では解は0に収束する。この場合では、0近傍で急激に-1から1に変化する連続関数と区別ができない。



特異点を持つ場合

例）　$f(x) = x^{-1}$

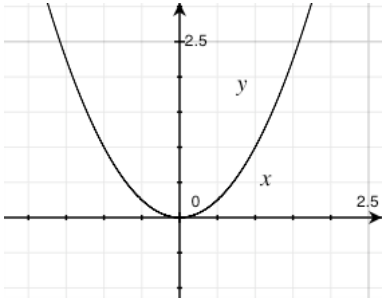
といった関数では、0の前後で、f(x)の符号は変化するが、解はない。このような場合は、x=0近傍でもf(x)の絶対値は小さくならないので、解がないことは判る。



重根

例）　$f(x) = x^2$　の解は２重根 x=0 である。

この場合、解の前後でf(x)の符号は正のままで変化しない。このような問題では二分法、挟み撃ち法などの囲い込み法では解けない。

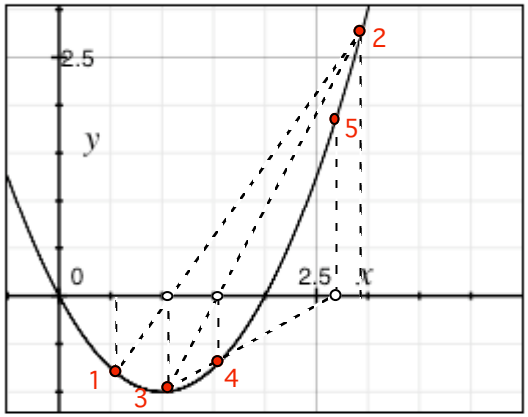


方程式の解：その他の方法

二分法、挟み撃ち法　の他

割線法

挟み撃ち法では、中点と異符号の端点を残したが、割線法では必ず新しい端点を残す方法

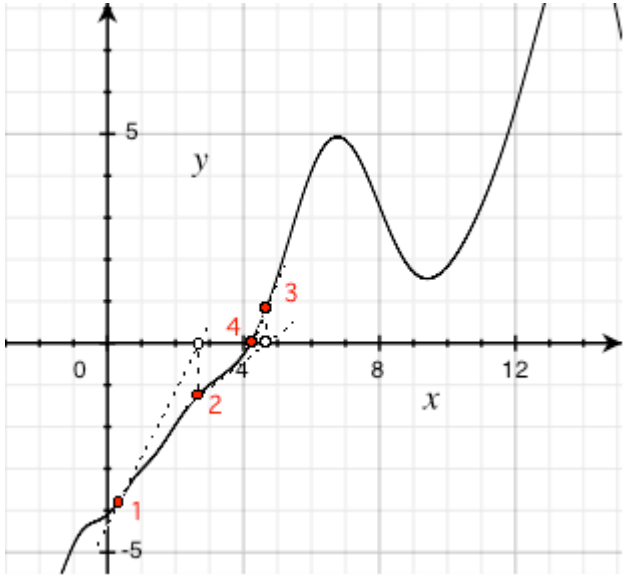


割線法

Newton法

関数f(x)の導関数も判明している場合、Newton法が使用できる。

割線法では、次の近似解を求めるのに、区間の端点を結んだ直線を使用したが、これを現在の解 x_i での接線を使用する方法。

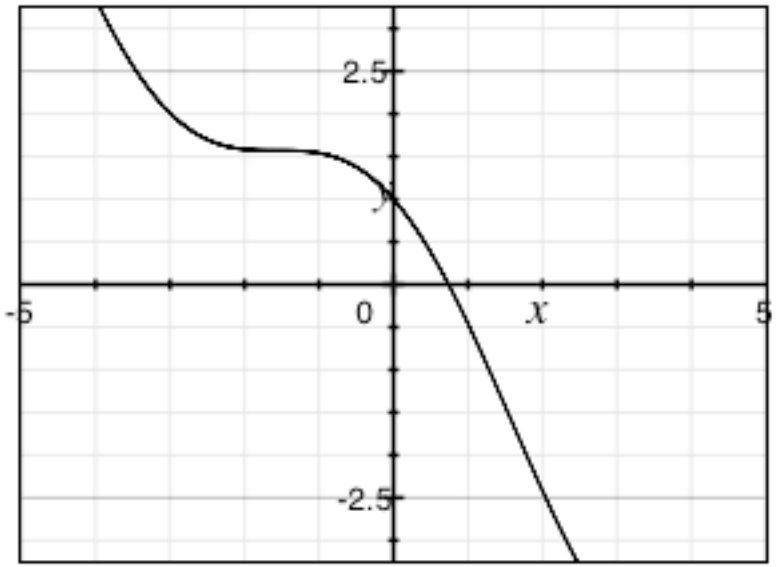


Newton法

以上述べてきた方法は、２次方程式のみならず、非線形方程式にも適用できる。

演習９−１Ｄ：その他の方程式

これまで述べてきた方法は、２次方程式のみならず、非線形方程式にも適用でき



f(x) = cos(x) -x の解を

- やみくも法
- 二分法
- 挟み撃ち法

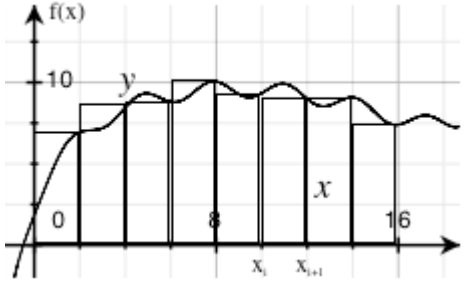
で求めてみよう。

プログラムを少し改造する

だけ。範囲は-1.0 ~ 3.0 くらい

数値積分

関数 f(x) の区間 [a, b] の積分をすることを考えよう。



計算機は、積分公式を知らないので（もちろん知っている人間が与えるという手はあります）、数学の定義に立ち返って数値的に積分を行う。

$$I = \int_a^b f(x)dx = \sum_{i=1}^{N} f(x_i)\cdot\Delta x = \Delta x\left[f(x_0) + f(x_1) + \cdots + f(x_N)\right]$$

この和を数値的に行えばよい。

便宜的に、積分区間[a, b]を等間隔でN等分し、その幅をh=(b-a)/Nとすると、各点は

$$x_0=a,\ x_1=x_0+h,\ \ldots,\ x_N=b$$

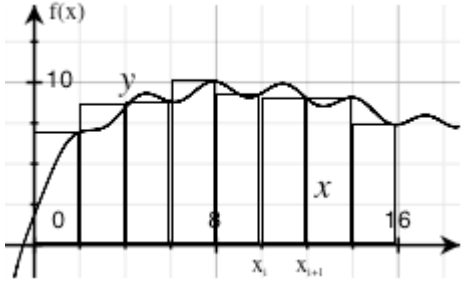
と表される。

これらを関数f(x) に代入し、足し合わせればよい。

数値積分：矩形近似

数値積分を行う際、関数f(x)をN等分し、関数をの積分を矩形で近似する方法。右図参照

関数の変化に比べ、分割する数が多ければ近似として成り立つが、分割が少なければ悪い近似となる。



$$I = \int_a^b f(x)dx = \sum_{i=1}^{N} f(x_i)\cdot\Delta x = h\left[f(x_0) + f(x_0+\Delta x) + \cdots + f(x_N)\right]$$

この和を数値的に行えばよい。

数値積分：台形近似

数値積分を行う際、関数f(x)をN等分し、関数をの積分を台形で近似する方法。

分割区間の両端で直線近似するので、矩形近似よりよい近似になっているが、関数の変化に比べ分割点が十分にあることが必要

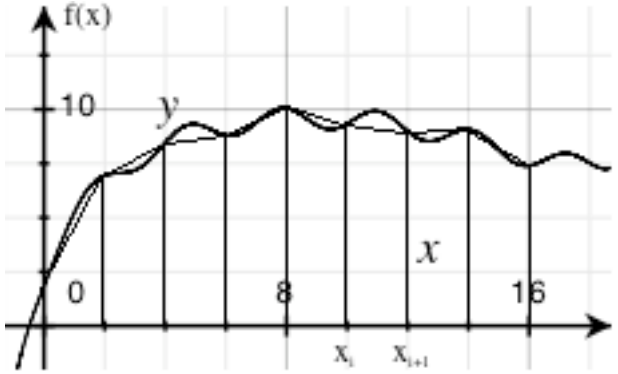
$$I = \int_a^b f(x)dx = \sum_{i=1}^{N}\left[\frac{1}{2}f(x_i) + \frac{1}{2}f(x_{i+1})\right]\cdot\Delta x = h\left[\frac{1}{2}f(x_0) + \sum_{i=1}^{N-1} f(x_i) + \frac{1}{2}f(x_N)\right]$$

この和を数値的に行えばよい。この式を台形公式と呼ぶ。

台形近似は、各区間でf(x)を直線近似したことになる。各区間でf(x)を２次曲線で近似すると積分は

$$I = \int_a^b f(x)dx = \sum_{i=1}^{N}\left[\frac{1}{3}f(x_i) + \frac{4}{3}f(x_{i+1}) + \frac{1}{3}f(x_{i+2})\right]\cdot\Delta x$$

となる。これをSimpson則という。



演習９−２Ａ：矩形近似

矩形近似を用いて積分公式

$$\int \cos x dx = \sin x$$

を確かめてみよう。

0.01 刻みで 0.0から 9.42 までのcos x を計算し、幅0.01 でかけて、その和をとる

$$I = \int_0^{9.42} \cos x dx = \sum_{i=1}^{N} \cos(x_i) \cdot 0.01$$

出来上がった xyデータをOpenOfficeでプロットし、もとのcos xがsin xとなっていることを確認する。

可視化にOpenOfficeを用いるので、OpenOfficeで数値積分してもよいし、

Cで計算してもよい。

次のページにある見本プログラム(9-2a.c)を入力して、コンパイルして、9-2aを実行すると、区間と刻み値を浮動小数点で聞いてくる。0.0 9.42 0.1などとして入力する。

出力は、画面ならびに 9-2a.txt というテキストファイルを出力する。

演習９−２A： 9-2a.cサンプルプログラム 1/2

```
/* kukei sekibun */
#include <stdio.h>
#include <math.h>

double xsin(double x);
double integ_xsin(double x);

int main(void){
          double start_p, end_p, kizami;
          int i, max;
          double x, integ=0.0;  /* set arb. const of integral to zero */
          FILE *file1;       /* file pointer, nessary to use file input/output */
          /* FILE *file2      file pointer nessary to use file input/output */

          file1 = fopen("9-2a.txt","w");   /*  open 9-2a.dat to write data*/
          /* file2 = fopen("text data","r");   open test.data  to read */
          /* fscanf(file2,"%lf %lf %lf", &start_p, &end_p, &kizami);   read data from
file */

          printf("input start, end points and step by floating point number\n");
          scanf("%lf %lf %lf", &start_p, &end_p, &kizami);
          printf("given integral section is [%lf, %lf] step %lf\n", start_p, end_p,
kizami);

          max = round((end_p - start_p)/kizami);

          printf("x\tcos x\tnumeric integration\tsin x\n");
          for (i=0; i<max; i++){
                    x = start_p + kizami * i ;
                    integ += cos( x ) * kizami;
                    printf("%lf\t%lf\t%lf\t%lf\n", x, cos(x), integ, sin(x) );
```

演習９−２A： 9-2a.cサンプルプログラム 2/2

```
/* output data to standard output*/
                  fprintf(file1,"%lf\t%lf\t%lf\t%lf\n", x, cos(x), integ, sin(x));  /*
output data to file2 */
          }

          fclose(file1);   /* close file1 */
          /* fclose(file2);    close file2 */
          return 0;
}


/* function ha  Integral(x sin x dx) = sin x - x cos x */
double xsin(double x){
          return (x*sin(x));
}

/* sin x - x cos x */
double integ_xsin(double x){
          return (sin(x)-x*cos(x));
}
```

演習９−２Ｂ：台形近似

矩形近似を用いて積分公式

$$\int x \sin x dx = \sin x - x \cos x$$

を確認すると共に矩形近似でも計算し、矩形近似のときと台形近似の際の違いを見てみよう。近似精度の確認のため刻み値は大きめにとる。

0.1 刻みで 0.0から 9.42 までのsin x - x cos x を台形公式で計算する。

$$I = \int_0^{9.42} \cos x dx = \frac{1}{2}\cos(a) + \sum_{i=2}^{N-1} \cos(x_i) \cdot 0.1 + \frac{1}{2}\cos(b)$$

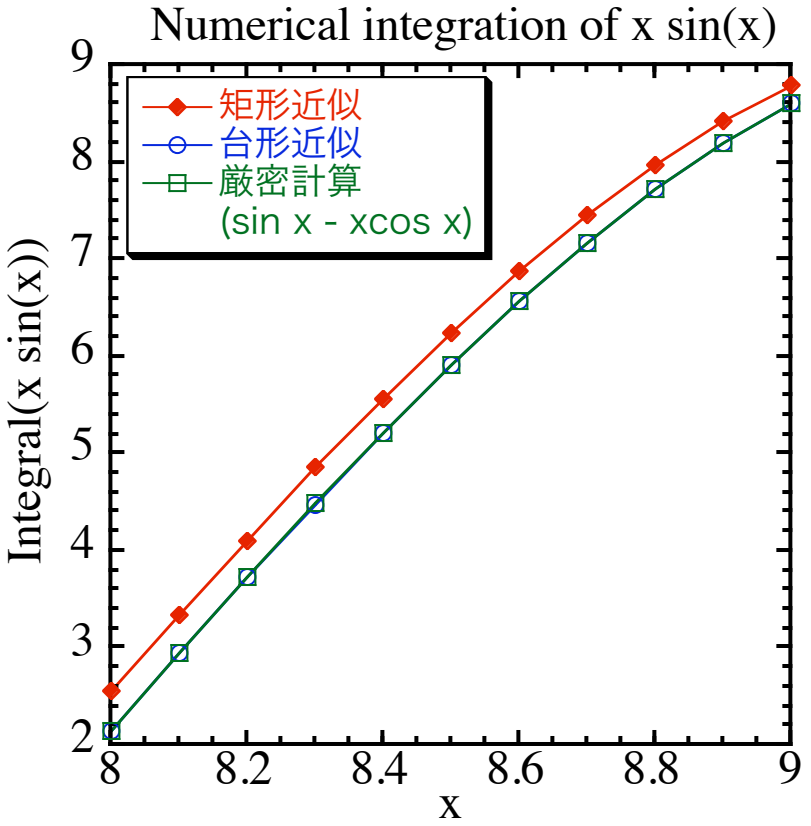
可視化にOpenOfficeを用いるので、OpenOfficeで数値積分してもよいし、Cで計算してもよい。C言語でプログラムするなら、矩形近似の見本プログラムを改変してよい

演習９−２Ｂ：台形近似

計算して比較した結果は右図の通り。
区間[8, 9]を拡大した。

台形近似のプロット点が厳密計算の真上に乗っていることが確認できる。一方、矩形近似は厳密計算のやや上方にプロット点がきている。

これからも、台形近似は計算量が少ない割によい近似を与えることが判る。

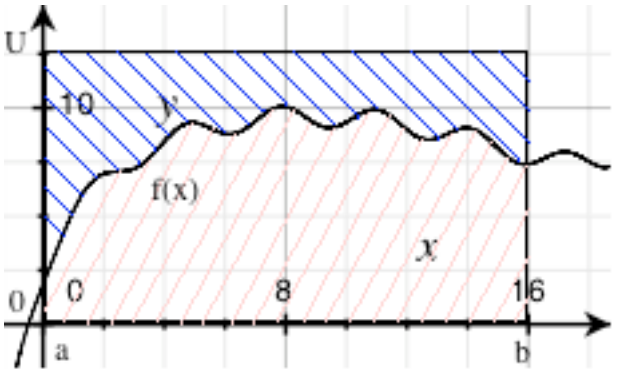


台形近似の誤差は h^3 のオーダーである。Simpson 近似の誤差は h^5 のオーダーであるが、多数の区間分割で数値計算をする必要があるので、丸め誤差が支配的になり、必ずしも h^5 のオーダーにはならない。そのため実用的には台形近似で十分である。

数値積分：モンテカルロ法（変わり種）

関数f(x)を区間[a, b]で積分することを考える。このとき、この区間内の全ての点で

$$0 \le f(x) \le U$$



であることが保証されているとする。すると、積分 *I* はxy平面で区間[a, b]で、 y=0 (x軸)と y=f(x) で囲まれた領域の面積である。（上図のピンクの領域）

領域 $a \le x \le b, 0 \le y \le U$ 内での任意の点(x, y)について

$$y \le f(x)$$

となる確率は

$$P = I / [U \cdot (b-a)]$$

である。これを用いて、領域内で一様な乱数の組(x, y)を多数作りそれらが、 $y \le f(x)$ にある存在確率を計算することで積分 *I* を求める方法である。要するに、ランダムに玉入れを行い $y \le f(x)$ に入った確率で面積を求めるというもの。良質の乱数が必要である。

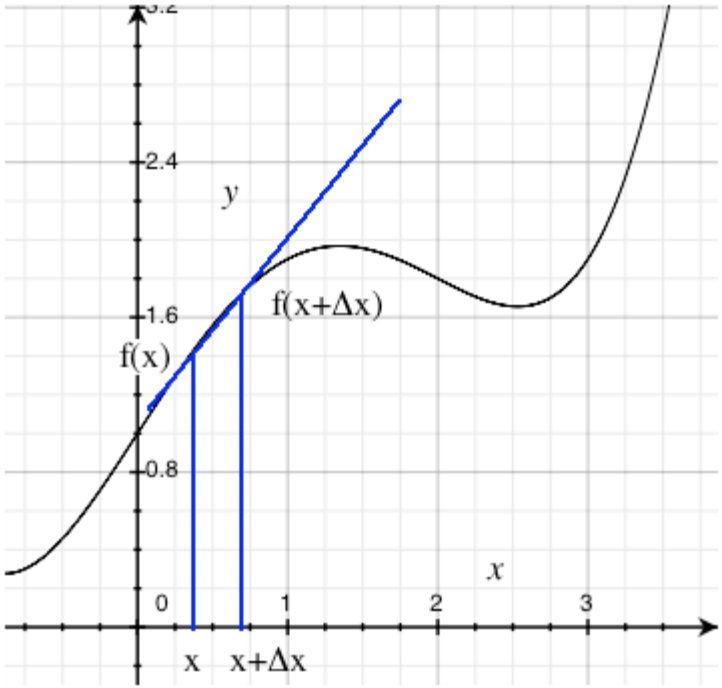
数値微分：

関数f(x)を微分することを考える。

微分の定義

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

より、この関数f(x)のx点周りの微分は、Δxを十分小さくとれば数値的に求めることができる。



演習９−３：数値微分

演習９−２Aで、矩形近似を用いて積分公式

$$\int \cos x dx = \sin x$$

を確かめてみた。今度は、

$$\frac{d \sin x}{dx} = \cos x$$

を確かめてみよう。計算は、区間[a, b]をN等分に分け（Δx=(b - a)/N）、

$$\frac{d \sin x}{dx} = \frac{\sin(x + \Delta x) - \sin(x)}{\Delta x}$$

を計算するだけである。区間[0.0, 9.24]で刻み値0.01で計算した結果をプロットし、解析解sin xと比較してみよう。

出来上がった xyデータをOpenOfficeでプロットし、もとのsin xがcos xとなっていることを確認する。

可視化にOpenOfficeを用いるので、OpenOfficeで数値積分してもよいし、Cで計算してもよい。

さらに、刻み値を 0.1、1.0 と変化させてみると解析解 sin x とどれくらい異なるか見てみよう。