8 数値計算

この節では、実数計算の代表例として数値計算問題を2つ取り上げる。

- 方程式の数値解を求める問題
- 数値積分を行う問題

8–1 方程式の解法 — Newton-Raphson法 —

例題8. 1 (方程式の数値解法；Newton-Raphson法)　方程式

$$\mathbf{f(x) = x - \cos x = 0}$$

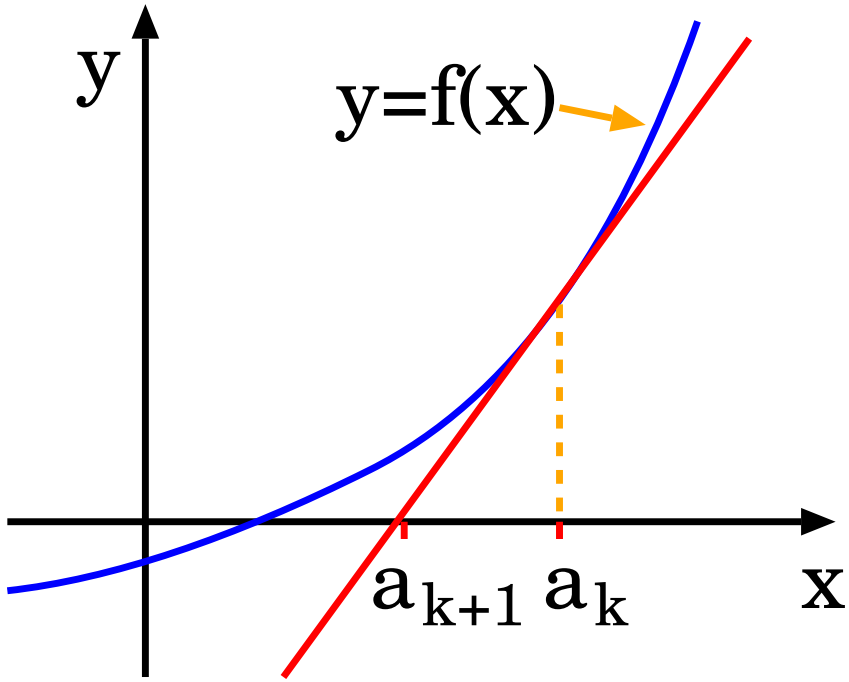
の実根を数値的に求めて出力するCプログラムを作成せよ。

(考え方)　一般に、方程式 $\mathbf{f(x)=0}$ の実根を数値的に求めるための方法としては、Newton-Raphson法(またはNewton法) , 二分法, 擬点法等が有名である。 このうち、

Newton-Raphson法は 、適当な近似解 $x=a_1$ から出発して、漸近式

$$a_{k+1} = a_k - \frac{f(a_k)}{f'(a_k)} \quad (k=1,2,3, \ldots)$$

により、次々とより良い近似解 $x=a_k$ $(k=1,2,3,\ldots)$ を求めていこうというものである。



もちろん、$a_1, a_2, a_3, \ldots$ が十分に収束したと判断できる時点で、このアルゴリズムは終了させる。 具体的な終了条件としては、正数ϵ を十分小さく選んで、

$$|a_{k+1}-a_k| \leq \epsilon \quad \text{または} \quad \left|\frac{a_{k+1}-a_k}{a_{k+1}}\right| \leq \epsilon$$

という形のものを考えれば良い。

[近似根の列 $a_1, a_2, a_3, \ldots$ は収束するとは限らないが、ほとんどの場合速く収束することが知られている。]

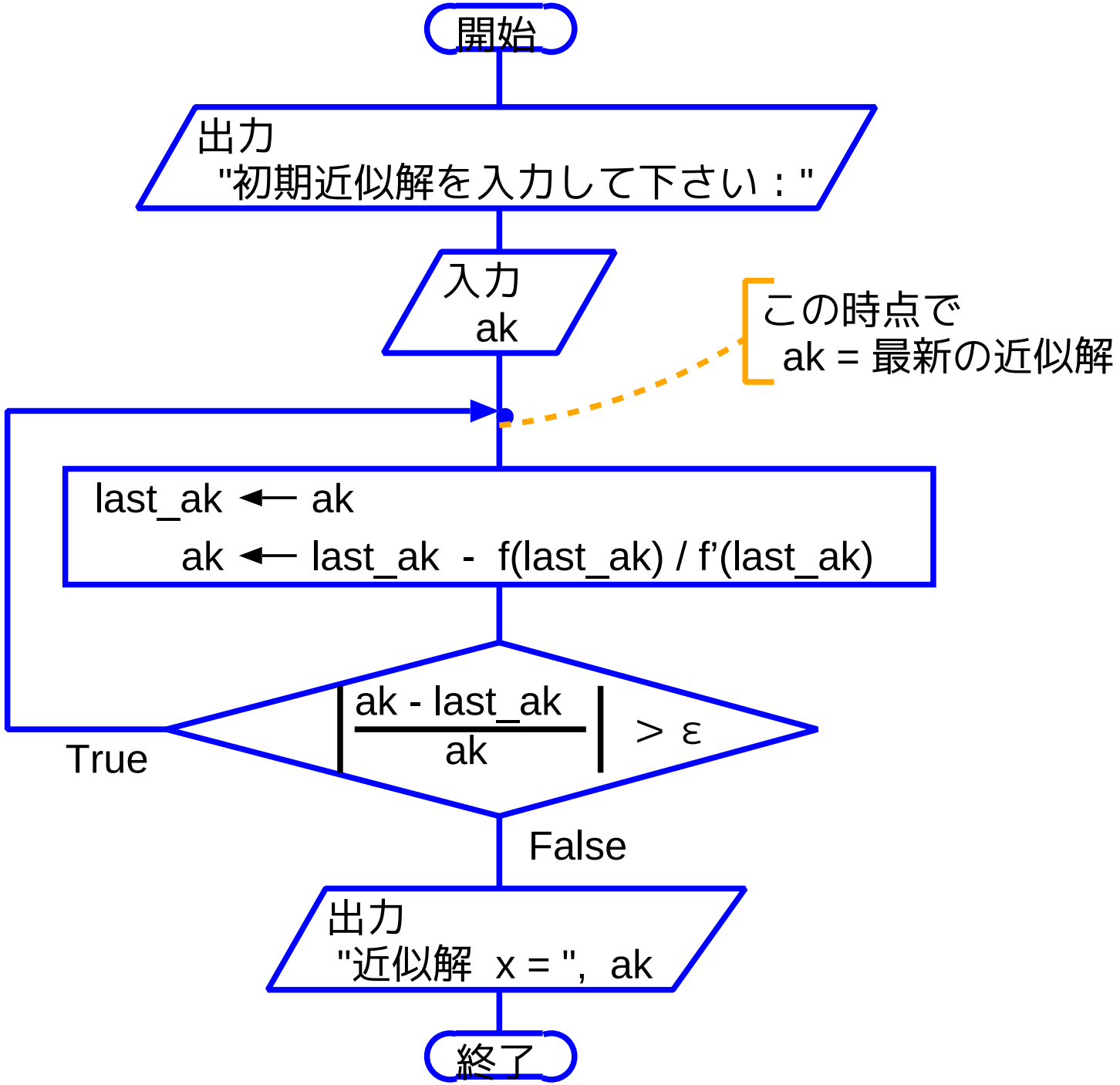
$$a_{k+1} = a_k - \frac{f(a_k)}{f'(a_k)} \qquad (k=1,2,3,\ \ldots)$$

(プログラミング)

次々と近似根 a_k を更新してゆくだけなら、a_k が出来た時点でそれまでの $a_1,\ a_2,\ a_3,\ \ldots,\ a_{k-1}$ は不要であるので、$a_1,\ a_2,\ a_3,\ \ldots$ を保持するために常に最新の a_k を記憶する変数だけがあれば良い。

しかし、$a_1,\ a_2,\ a_3,\ \ldots,\ a_k$ が収束したかどうかの判定を行うために、最も最近の2つの近似根 $a_{k-1},\ a_k$ を (終了判定の時に) 保持しておく必要がある。

⇨ これらの連続した2つの近似根 $a_{k-1},\ a_k$ を保持するために、
それぞれ `last_ak`, `ak` という名前の変数を用意すれば、...


開始
出力
"初期近似解を入力して下さい："
入力
ak
この時点で
ak = 最新の近似解
last_ak ← ak
ak ← last_ak - f(last_ak) / f'(last_ak)
|ak - last_ak / ak| > ε
True
False
出力
"近似解 x = ", ak
終了

```
[motoki@x205a]$  nl newton-raphson.c   Enter
     1 /* 方程式 f(x)=x-cos(x)=0 の実根を Newton-Raphson法に...
     2 /* 数値的に求めて出力するCプログラム

     3 #include <stdio.h>
     4 #include <math.h>

     5 #define  f(x)                ((x) - cos(x))   引数付きマクロ
     6 #define  derivative_f(x)     (1.0 + sin(x))
     7 #define  EPSILON             (0.5e-15)

     8 int main(void)
     9 {
    10   double  ak, last_ak;

    11   printf("方程式 x-cos(x)=0 の初期近似解を入力して下さい:"
                                                    "  x= ");
```

```
12    scanf("%lf", &ak);

13    do {
14      last_ak = ak;
15      ak = last_ak - f(last_ak)/derivative_f(last_ak);
16    }while (ak==0.0 || fabs((ak-last_ak)/ak) > EPSILON );
```

((last_ak) - cos(last_ak)) と展開される

↓

↑

0で割って終了判定が乱れるのを避けるため

⇨ 繰返し終了時に次が成立するはず。

$ak \neq 0$ かつ

$ak - |ak| \times 0.5 \times 10^{-15} \leqq last\_ak \leqq ak + |ak| \times 0.5 \times 10^{-15}$

(ak の上位16桁目以下を四捨五入で丸めた時、
akの丸め誤差の範囲内に last_ak がある。)

有効桁15桁で出力

↓

```
   17    printf("最終の近似解は      x=%#21.15g.\n", ak);
   18    return 0;
   19 }
[motoki@x205a]$  gcc newton-raphson.c -lm  Enter
[motoki@x205a]$  ./a.out  Enter
方程式 x-cos(x)=0 の初期近似解を入力して下さい:  x=  1.0  Enter
最終の近似解は      x=     0.739085133215161.
[motoki@x205a]$
```

- 繰り返しを続ける条件として16 行目以外の記述を行った、
  「難あり」のプログラム例を幾つか次に例示する。

16行目の代わりに `}while (fabs(f(ak)) > EPSILON);`

⇨ 例えば $f(x)=10^{-20}(x - \cos x)$ の時に不十分な精度の解

```
[motoki@x205a]$  nl newton-raphson_bad1.c
     1 /* 方程式 f(x)=1e-20*(x-cos(x))=0 の実根を Newton-   */
     2 /* Raphson法により数値的に求めて出力するCプログラム */
     3 /* (繰り返しを続ける「難あり」の条件を用いた例1)     */

     4 #include <stdio.h>
     5 #include <math.h>

     6 #define  f(x)                (1e-20*((x) - cos(x)))
     7 #define  derivative_f(x)     (1e-20*(1.0 + sin(x)))
     8 #define  EPSILON             (0.5e-15)

     9 int main(void)
```

```
10 {
11   double  ak, last_ak;

12   printf("方程式 x-cos(x)=0 の初期近似解を入力して下さい:
13   scanf("%lf", &ak);

14   do {
15     last_ak = ak;
16     ak = last_ak - f(last_ak)/derivative_f(last_ak);
17   }while (fabs(f(ak)) > EPSILON);

18   printf("最終の近似解は     x=%#21.15g.\n", ak);
19   return 0;
20 }
[motoki@x205a]$  gcc newton-raphson_bad1.c -lm
[motoki@x205a]$  ./a.out
方程式 x-cos(x)=0 の初期近似解を入力して下さい:  x=  1.0
最終の近似解は     x=    0.750363867840244.
[motoki@x205a]$
```

16行目の代わりに `}while (((last_ak-ak)/ak) > EPSILON);`

⇨ 例えば $(a_0-a_1)/a_1 < 0$ の時には収束してなくても繰り返しを終了してしまう。結果として、不十分な精度の解が多い。

```
[motoki@x205a]$  nl newton-raphson_bad2.c
     1 /* 方程式 f(x)=x-cos(x)=0 の実根を Newton-Raphson法に...
     2 /* 数値的に求めて出力するCプログラム
     3 /* (繰り返しを続ける「難あり」の条件を用いた例2)              *

     4 #include <stdio.h>
     5 #include <math.h>

     6 #define  f(x)                ((x) - cos(x))
     7 #define  derivative_f(x)     (1.0 + sin(x))
     8 #define  EPSILON             (0.5e-15)

     9 int main(void)
    10 {
    11   double  ak, last_ak;
```

```
    12   printf("方程式 x-cos(x)=0 の初期近似解を入力して下さい:
    13   scanf("%lf", &ak);

    14   do {
    15     last_ak = ak;
    16     ak = last_ak - f(last_ak)/derivative_f(last_ak);
    17   }while (((last_ak-ak)/ak) > EPSILON);

    18   printf("最終の近似解は     x=%#21.15g.\n", ak);
    19   return 0;
    20 }
[motoki@x205a]$  gcc newton-raphson_bad2.c -lm
[motoki@x205a]$  ./a.out
方程式 x-cos(x)=0 の初期近似解を入力して下さい:  x=  -1.0
最終の近似解は     x=      8.71621695877957.
[motoki@x205a]$
```

16行目の代わりに `}while (fabs(ak-last_ak) > EPSILON);`

⇨ 例えば$f(x)=(10^{20}x) - \cos(10^{20}x))$ の時に不十分な精度の解

また、$f(x)=(10^{-20}x) - \cos(10^{-20}x))$ の時に、

$a_{k+1} \neq a_k$, $a_{k+2}=a_k$, $a_{k+3}=a_{k+1}$, ... という風に振動し

繰り返しを終了しない可能性も

```
[motoki@x205a]$  nl newton-raphson_bad3.c
     1 /* 方程式 f(x)=(1e20*x)-cos(1e20*x)=0 の実根を Newton- *
     2 /* Raphson法により数値的に求めて出力するCプログラム     */
     3 /* (繰り返しを続ける「難あり」の条件を用いた例3)        */

     4 #include <stdio.h>
     5 #include <math.h>

     6 #define  f(x)                ((1e20*x) - cos(1e20*x))
     7 #define  derivative_f(x)     (1e20 + 1e20*sin(1e20*x))
     8 #define  EPSILON             (0.5e-15)
```

```
 9 int main(void)
10 {
11   double  ak, last_ak;

12   printf("方程式 x-cos(x)=0 の初期近似解を入力して下さい:
13   scanf("%lf", &ak);

14   do {
15     last_ak = ak;
16     ak = last_ak - f(last_ak)/derivative_f(last_ak);
17   }while (fabs(ak-last_ak) > EPSILON);

18   printf("最終の近似解は     x=%#21.15g.\n", ak);
19   return 0;
20 }
[motoki@x205a]$  gcc newton-raphson_bad3.c -lm
[motoki@x205a]$  ./a.out
方程式 x-cos(x)=0 の初期近似解を入力して下さい:  x=  1.0
最終の近似解は     x=-1.52870240130842e-16.
[motoki@x205a]$
```

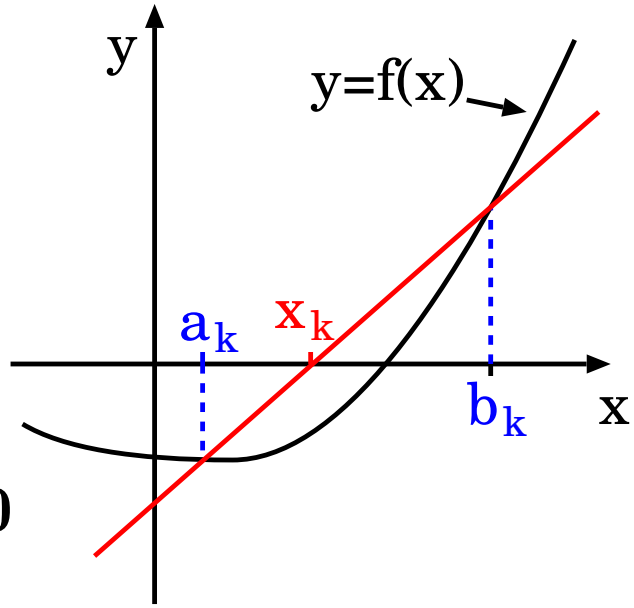
□演習8. 2 (擬点法) 擬点法は

$$f(a_1)f(b_1)<0$$

となるような近似解の組 $x=a_1$, $x=b_1$ から出発して、漸化式

$$x_k = \frac{b_k f(a_k) - a_k f(b_k)}{f(a_k) - f(b_k)} \quad (k=1,2,3,\ \ldots)$$

$$(a_{k+1},\ b_{k+1}) = \begin{cases} (a_k,\ x_k) & \text{if } f(a_k)f(x_k)<0 \\ (x_k,\ b_k) & \text{otherwise} \end{cases}$$

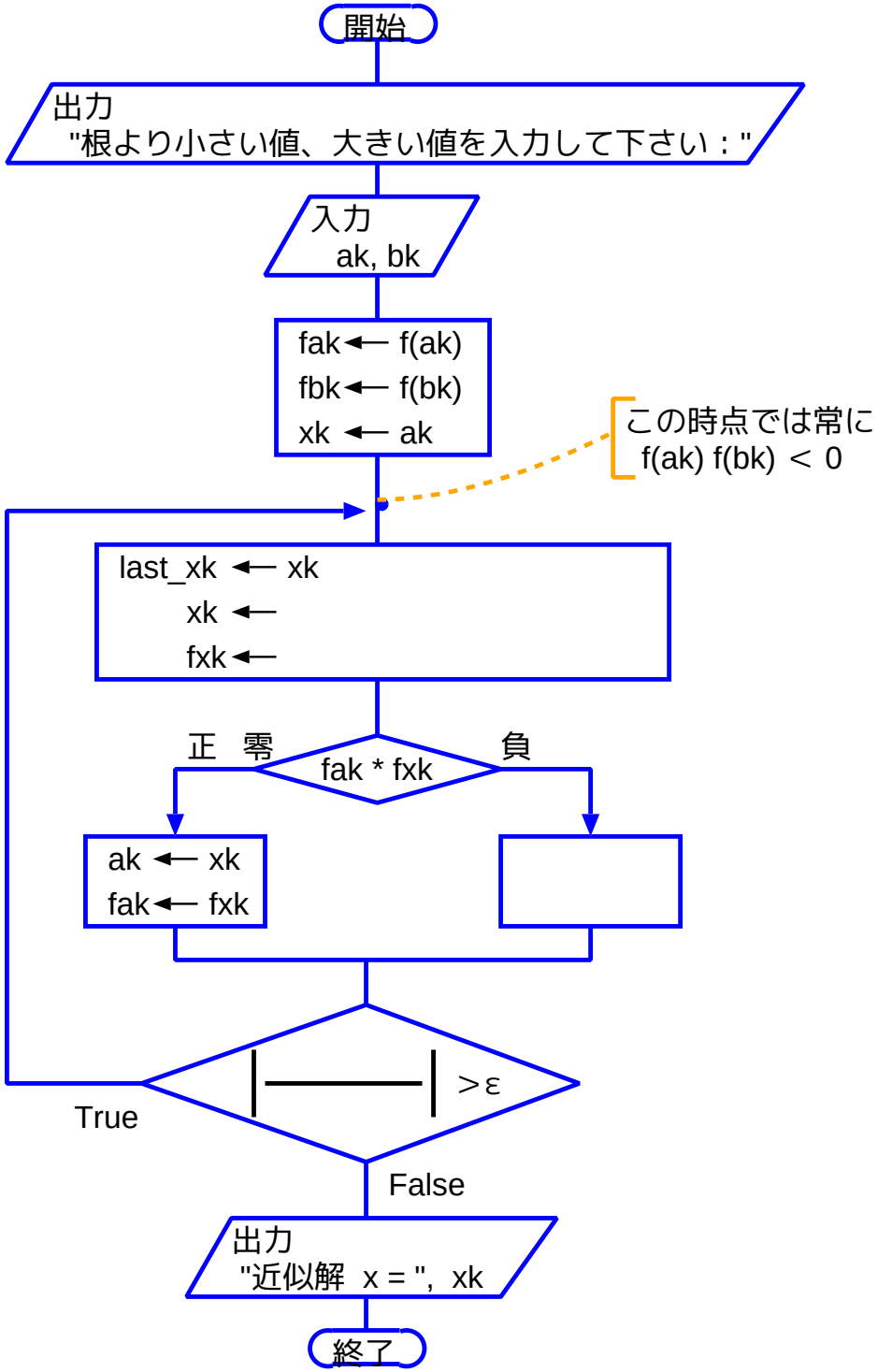


により、次々とより良い近似解 $x=x_k$ $(k=1,2,3,\ldots)$ を求めていこうというものである。 $f(x_k)=0$ となった時点、あるいは $x_1, x_2, x_3, \ldots$ が十分に収束したと判断できる時点で、この計算は終了する。　擬点法を用いて方程式 $f(x)=x-\cos x=0$ の近似解を小数点以下15桁まで求めるCプログラムを作成せよ。但し、ここでは $a_1=0$, $b_1=1$ とせよ。

$$\begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \longrightarrow \begin{pmatrix} a_2 \\ b_2 \end{pmatrix} \longrightarrow \begin{pmatrix} a_3 \\ b_3 \end{pmatrix} \longrightarrow \begin{pmatrix} a_4 \\ b_4 \end{pmatrix} \longrightarrow \cdots\cdots$$

解の両側の値を常に保持

Hint


開始
出力
"根より小さい値、大きい値を入力して下さい："
入力
ak, bk
fak ← f(ak)
fbk ← f(bk)
xk ← ak
この時点では常に
f(ak) f(bk) ＜ 0
last_xk ← xk
xk ←
fxk ←
正
零
fak * fxk
負
ak ← xk
fak ← fxk
＞ε
True
False
出力
"近似解 x = ", xk
終了

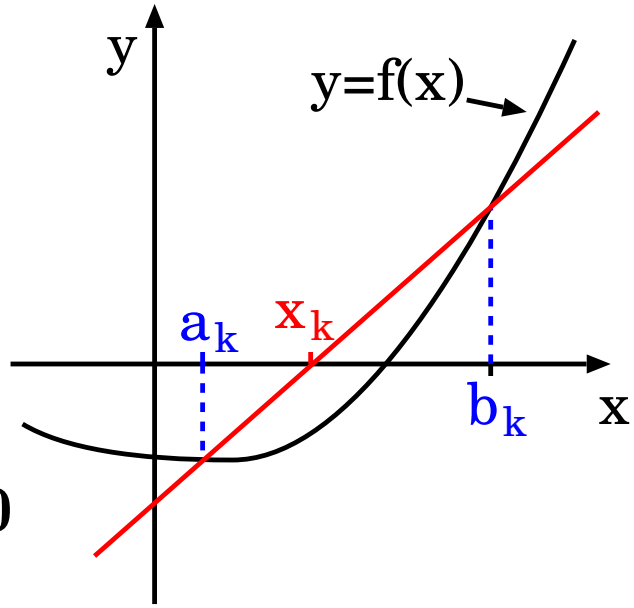
□演習8. 2 (擬点法; 収束条件を変更すると...) 擬点法は

$$f(a_1)f(b_1)<0$$

となるような近似解の組 $x=a_1$, $x=b_1$ から出発して、漸化式

$$x_k = \frac{b_k f(a_k) - a_k f(b_k)}{f(a_k) - f(b_k)} \quad (k=1,2,3,\ \ldots)$$

$$(a_{k+1},\ b_{k+1}) = \begin{cases} (a_k,\ x_k) & \text{if } f(a_k)f(x_k)<0 \\ (x_k,\ b_k) & \text{otherwise} \end{cases}$$

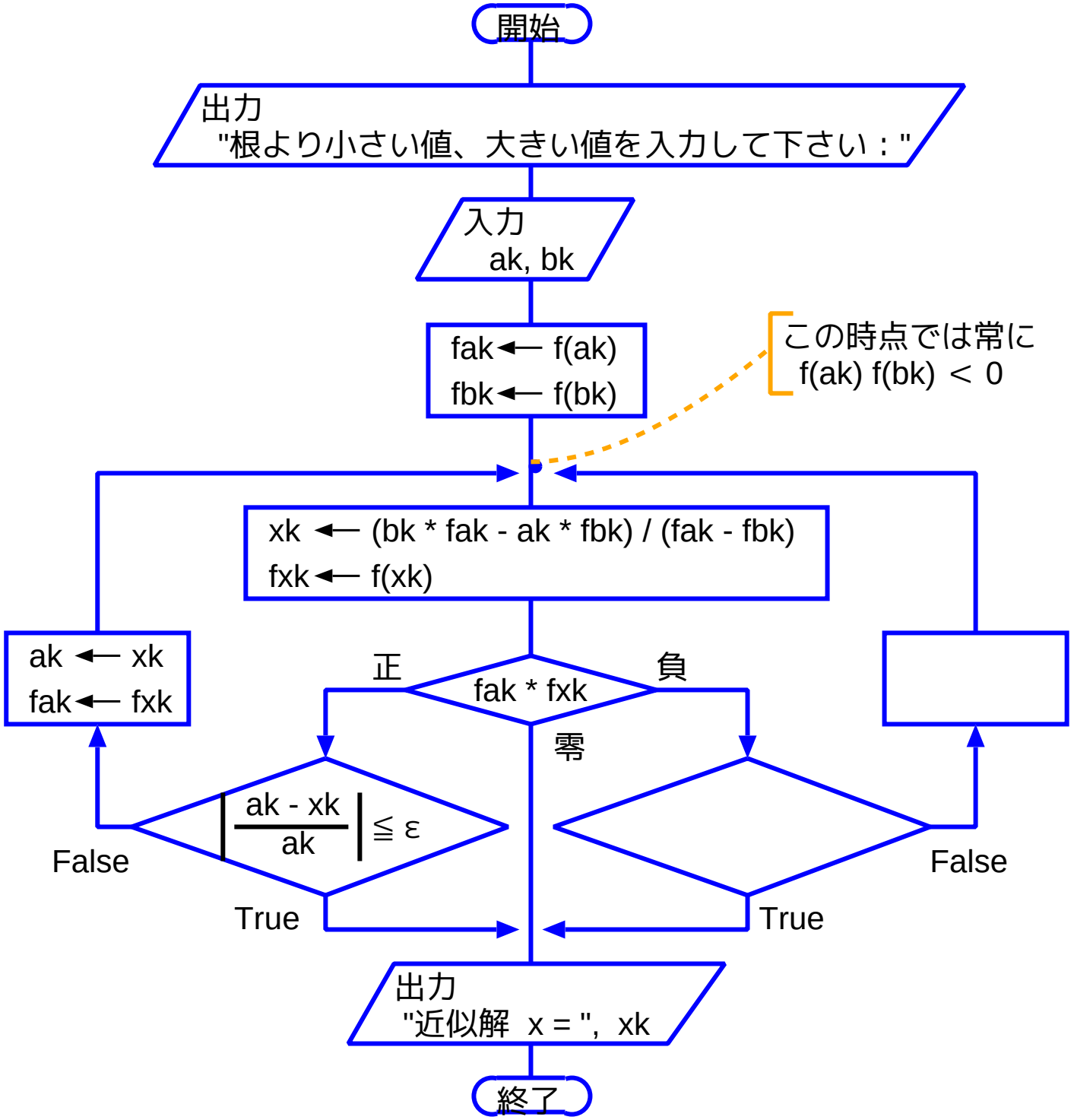


により、次々とより良い近似解 $x=x_k$ $(k=1,2,3,\ldots)$ を求めていこうというものである。 $f(x_k)=0$ となった時点、あるいは (a_1,b_1), (a_2,b_2), (a_3,b_3), ... が十分に収束収束条件を変更すると...したと判断できる時点で、この計算は終了する。　擬点法を用いて方程式 $f(x)=x-\cos x=0$ の近似解を小数点以下15桁まで求めるCプログラムを作成せよ。但し、ここでは $a_1=0$, $b_1=1$ とせよ。

$$\begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \longrightarrow \begin{pmatrix} a_2 \\ b_2 \end{pmatrix} \longrightarrow \begin{pmatrix} a_3 \\ b_3 \end{pmatrix} \longrightarrow \begin{pmatrix} a_4 \\ b_4 \end{pmatrix} \longrightarrow \cdots\cdots$$

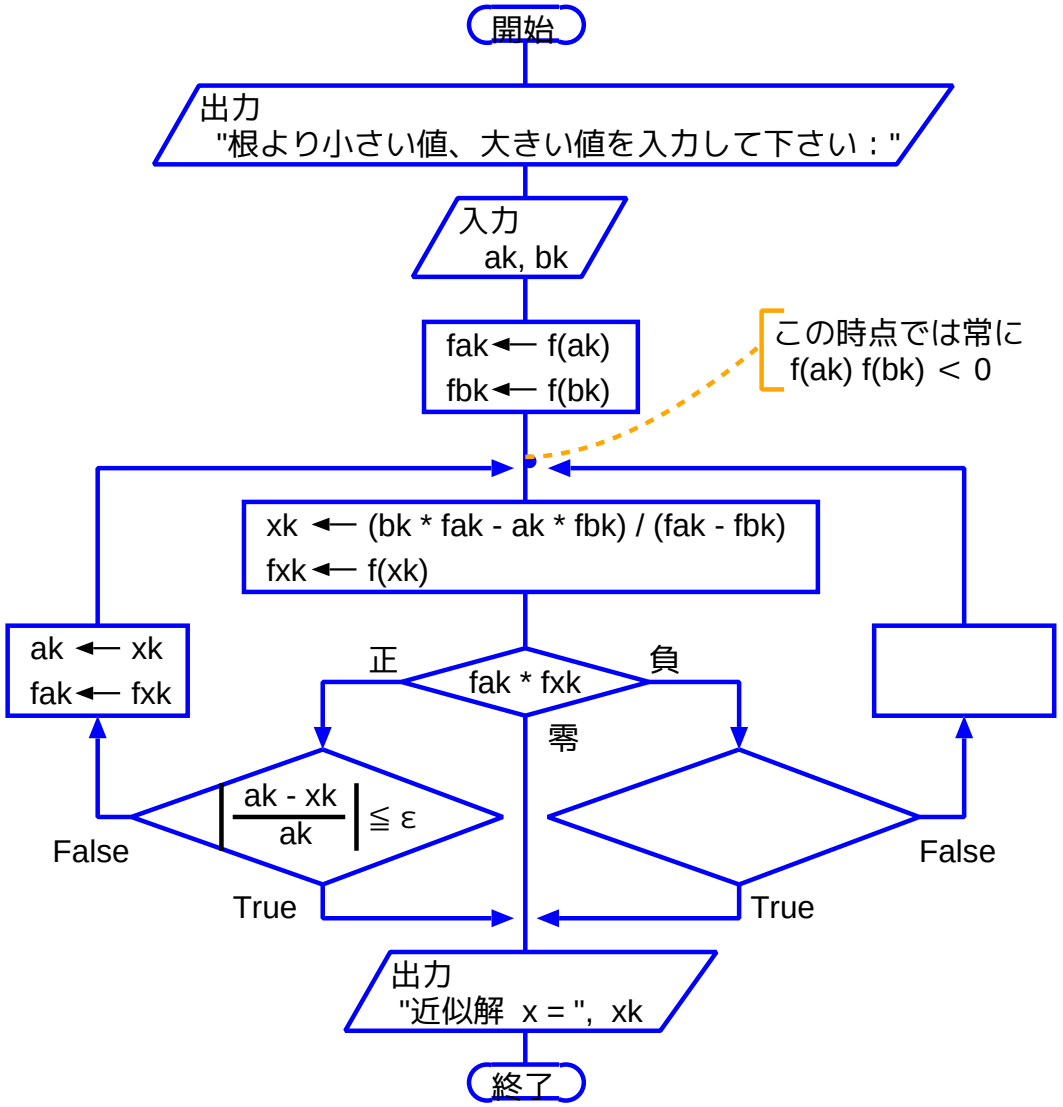
解の両側の値を常に保持

Hint


開始
出力
"根より小さい値、大きい値を入力して下さい："
入力
ak, bk
fak ← f(ak)
fbk ← f(bk)
この時点では常に
f(ak) f(bk) ＜ 0
xk ← (bk * fak - ak * fbk) / (fak - fbk)
fxk ← f(xk)
ak ← xk
fak ← fxk
正
負
fak * fxk
零
ak - xk / ak ≦ ε
False
True
True
False
出力
"近似解 x = ", xk
終了

```
    ......
#define  TRUE  1
    ......
main()
{
   ......

   while (TRUE) {
      ......
      if (fak*fxk >0) {
         if (  )
            break;
         ......
      }else if (fak*fxk <0) {
         if (  )
            break;
         ......
      }else
         break;
   }
   ......
}
```



8–2 数値積分 —Simpsonの公式—

例題8. 3 (数値積分；Simpsonの公式)　定積分 $\int_0^1 \frac{4}{1+x^2}dx$ の値を数値的に求めて出力するCプログラムを作成せよ。

(考え方)

一般に、定積分 $\int_a^b f(x)\ dx$ の値を数値的に求めるための方法としては、ニュートン・コーツの積分公式、エルミットの内挿公式、ガウス形の積分公式 等がある。 このうち、

シンプソンの公式は Newton-Cotesの積分公式の一種で、解析学の教科書にも載るほど有名なものである。

シンプソンの公式によれば 、定積分値は

$$\int_a^b f(x)dx \approx \frac{h}{3}[\ f(a_0)+f(a_{2n}) + 4(\ f(a_1)+f(a_3)+\cdots+f(a_{2n-3})+f(a_{2n-1})\ ) + 2(\ f(a_2)+f(a_4)+\cdots+f(a_{2n-2})\ ) \ ]$$

但し、$h = \frac{b-a}{2n}$

$a_i = a+ih$

と近似でき、この近似の際の誤差は

$$|\text{誤差}| \leq \frac{(b-a)^5}{2880\ n^4} \max\left\{ |f^{(4)}(\xi)| \ \mid \ a \leq \xi \leq b \right\}$$

と見積もられる。

補足：

- $\mathbf{n}$ は1以上の整数で、十分大きく選ぶ。

  この $\mathbf{n}$ で指定された個数に積分区間 $[\mathbf{a},\mathbf{b}]$ は等分割され、各々の小区間において $\mathbf{f(x)}$ が2次多項式で近似され定積分の近似値が求められることになる。

- 特に $\mathbf{f(x)=\frac{4}{1+x^2}}$ の場合は、$\mathbf{\max\left\{|f^{(4)}(\xi)| \mid 0\leq\xi\leq 1\right\} = 96}$

  ⇨ $\mathbf{\int_0^1 \frac{4}{1+x^2}dx}$ の近似値を求めるのにシンプソンの公式を使った場合、

  小区間の個数が $\mathbf{n=10}$ では

$$|\text{誤差}| \leq \frac{(1-0)^5}{2880\times 10^4}\times 96 + |\text{計算に伴う誤差}|$$
$$\approx 3.3\times 10^{-6} + |\text{計算に伴う誤差}|$$

  $\mathbf{n=1000}$ では

$$|\text{誤差}| \leq \frac{(1-0)^5}{2880\times 1000^4}\times 96 + |\text{計算に伴う誤差}|$$
$$\approx 3.3\times 10^{-14} + |\text{計算に伴う誤差}|$$

- 真値は $\mathbf{\int_0^1 \frac{4}{1+x^2 dx} = \pi = 3.1415926535\ 8979323846\ 2643383279\cdots}$

(プログラミング)　与えられた式に従って計算するだけである。

累算する箇所が2つあるが、そのうち
$4(f(a_{2n-1})+f(a_{2n-3})+\cdots)$ を保持するためにdouble型変数 sum4 を、
$2(f(a_{2n-2})+f(a_{2n-4})+\cdots)$ を保持するためにdouble型変数 sum2 を
用意してプログラムを構成した。$n=1000$ として.....

```
[motoki@x205a]$  nl numerical-integral-by-simpson.c   Enter
   1 /* 定積分 \int_0^1 \frac{4}{1+x^2} dx の値を Simpsonの...
   2 /* により数値的に求めて出力するCプログラム                  >
   3 #include <stdio.h>

   4 #define  N      (1000)
   5 #define  A      (0.0)
   6 #define  B      (1.0)
   7 #define  f(x)   (4.0/(1.0+(x)*(x)))
```

```
 8 int main(void)
 9 {
10   double ans, sum4, sum2;
11   int    i;

12   sum4 = 0.0;
13   for (i=2*N-1; i>=1; i-=2)
14     sum4 += f(A+(B-A)*((double)i)/(2.0*(double)N));

15   sum2 = 0.0;
16   for (i=2*N-2; i>=2; i-=2)
17     sum2 += f(A+(B-A)*((double)i)/(2.0*(double)N));

18   ans = (f(A)+f(B)+4.0*sum4+2.0*sum2)*(B-A)
             /(6.0*(double)N);
19   printf("定積分値(近似)は %22.16g\n", ans);
20   return 0;
```

```
  21 }
[motoki@x205a]$  gcc numerical-integral-by-simpson.c  Enter
[motoki@x205a]$  ./a.out  Enter
定積分値(近似)は      3.141592653589791
[motoki@x205a]$
```