

3. 非線形方程式の解法

非線形方程式 $f(x)=0$ の根を求める。

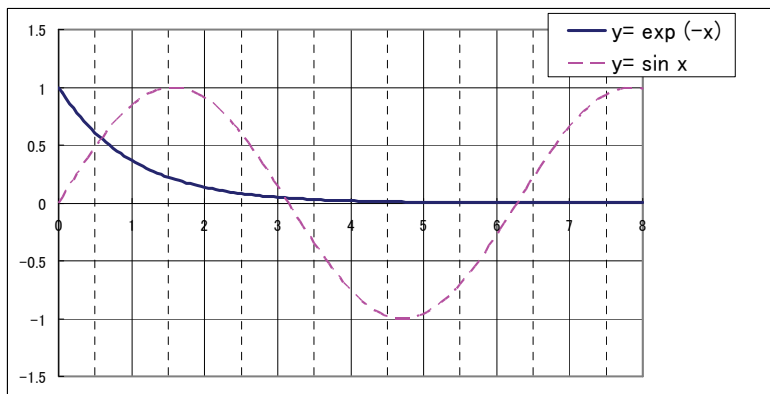
3.1 Newton 法

【アルゴリズム】

```

x := 与えられた初期値
for k=0, 1, 2, ...      [収束するまで繰り返す]
    Δx := -f(x)/f'(x)
    x := x + Δx
end
    
```

【問題】 $f(x) = \exp(-x) - \sin x = 0$ の根は無数にあるが、その根を一つずつニュートン法により求める。
 下図より、 x を大きくすると、根は $k\pi$ ($k=1, 2, 3, \dots$) に近づいていくことがわかる。



【プログラム 3.1.1】

```

1 /* Newton's Method to solve f(x)=0 */
2
3 #include <stdio.h>          /* いつものおまじない：標準入出力用ヘッダーファイル */
4 #include <math.h>          /* 数学関数用ヘッダーファイル：関数 fabs() 等を使用しているので必要となる */
5
6 #define KMAX 20            /* マクロにより、Newton 法における最大反復回数を定義する */
7
8 void eval(float, float *, float *); /* 関数を main 関数の後に置く場合、関数のプロトタイプ宣言が必要 */
9
10 void main(void)            /* ANSI 規格では main 関数の記述は void main(void) */
11 {
12     float x, eps, f, df, d; /* 変数の型宣言 */
13     int k;
14
15     printf("initial guess and tolerance: x, eps ? "); /* 変数の入力の際は、予め printf 文などで促す */
16     scanf("%g%g", &x, &eps); /* x の初期値と収束判定値 ε の値の読み込み */
17
18     /* 注意：日本語キーボードの¥は、US キーボードのバックスラッシュ\となる：\n, \t 等 */
19     printf("\n k\tx\tf(x)\tdf'(x)\tcorrection\n");
20     /* 上の文は、以下のようにタブ(\t)を使わずに書いてもよい */
21     /* printf(" k\t\t\t x\t\t\t f(x)\t\t\t f'(x)\t\t\t correction\n"); */
22     for(k=0; k<KMAX; k++){ /* Newton 法の反復により、根を求めるためのループ */
23         eval(x, &f, &df); /* 関数 eval で f(x) と f'(x) の値を計算する */
24         d = -f/df; /* Δx = f(x)/f'(x) の計算 */
25         printf("%2d %15.6e %15.6e %15.6e %15.6e\n", k, x, f, df, d);
26         x +=d; /* x ← x + Δx */
    
```

```

27     if(fabs(f)<eps) break;                /* |f(x)| < εならば、ループを出る */
28     }
29     if(k>=KMAX) printf("not convergent?\n"); /* k=KMAX のときは、収束していない恐れがある旨を出力 */
30     printf("\nroot=%g\n", x);            /* f(x)=0 の根 x を出力 */
31 }
32
33 void eval(float x, float *f, float *df)    /* ANSI 規格では、引数欄に変数の型宣言を記述 */
34 {
35     float e;                               /* 変数の型宣言 */
36     e= exp(-x);
37     *f = e - sin(x);                       /* f(x) = exp(-x) - sin x */
38     *df= -e - cos(x);                     /* f'(x) = -exp(-x) - cos x */
39 }

```

【実行例】

1) 初期値を 0.5, 収束判定値を 10^{-6} とする。

—————実行開始—————

initial guess and tolerance: x, eps ? 0.5 1.0e-6

k	x	f(x)	f'(x)	correction
0	5.000000e-01	1.271051e-01	-1.484113e+00	8.564382e-02
1	5.856438e-01	4.011282e-03	-1.390104e+00	2.885599e-03
2	5.885294e-01	4.611285e-06	-1.386901e+00	3.324884e-06
3	5.885327e-01	-1.274799e-08	-1.386897e+00	-9.191734e-09

root=0.588533

—————おしまい—————

2) 初期値を3.0とする。

—————実行開始—————

initial guess and tolerance: x, eps ? 3.0 1.0e-6

k	x	f(x)	f'(x)	correction
0	3.000000e+00	-9.133294e-02	9.402055e-01	9.714147e-02
1	3.097142e+00	7.416478e-04	9.538341e-01	-7.775439e-04
2	3.096364e+00	8.331214e-08	9.537641e-01	-8.735089e-08

root=3.09636

—————おしまい—————

3) 初期値を6.0とする。

—————実行開始—————

initial guess and tolerance: x, eps ? 6.0 1.0e-6

k	x	f(x)	f'(x)	correction
0	6.000000e+00	2.818942e-01	-9.626490e-01	2.928318e-01
1	6.292832e+00	-7.796926e-03	-1.001803e+00	-7.782893e-03
2	6.285049e+00	3.122672e-07	-1.001862e+00	3.116868e-07

root=6.28505

—————おしまい—————

3.2 二分法

【アルゴリズム】

```

 $x_1, x_2 :=$  初期区間の端点 ( $x_1 < x_2$ )
while ( $x_2 - x_1$ )  $> \epsilon$  do
   $x_m := (x_1 + x_2) / 2$ 
  if  $f(x_1) \cdot f(x_m) < 0$  then
     $x_2 := x_m$ 
  else
     $x_1 := x_m$ 
  end
end
end

```

【問題】 3.1 節と同じ問題 $f(x)=0$ の根を、一つずつ二分法により求める。

プログラム 3.2.1 は、 $f(x_1)$ と $f(x_m)$ が異符号かどうかの判定を、 $f(x_1) \cdot f(x_m) < 0$ かどうかで行っている。ただし、 $f(x_1)$ と $f(x_m)$ の値が非常に大きい（あるいは非常に小さい）とき、乗算 $f(x_1) \cdot f(x_m)$ はオーバーフロー（あるいはアンダーフロー）を招く恐れがある。

これを防ぐため、プログラム 3.2.2 では、符号を設定する関数 `sign()` を定義し、異符号かどうかの判定を $\text{sign}(f(x_1)) \cdot \text{sign}(f(x_m)) < 0$ で行う。

数値計算の実用プログラムは、このような細部にまで配慮が行き届いているものが多い。が、最初はあまり気にしないで始めましょう！

【プログラム 3.2.1】

```

1 /* bisectional method to solve f(x)=0 */
2
3 #include <stdio.h>          /* いつものおまじない：標準入出力用ヘッダーファイル */
4 #include <math.h>          /* 数学関数用ヘッダーファイル：関数 sin を使用している所以需要 */
5
6                             /* 関数を main 関数よりも前に置けば、関数のプロトタイプ宣言は必要なし */
7 float f(float x)           /* 関数 f(x)= exp(-x) - sin x ; ANSI 規格では、引数欄に変数の型宣言を記述 */
8 {
9     return( exp(-x) - sin(x) );
10 }
11
12 void main(void)            /* ANSI 規格では main 関数の記述は void main(void) */
13 {
14     float x1, x2, eps, f1, xm, fm;
15     int i;
16
17     printf("initial interval [x1, x2] (x1<x2) and tolerance: x1, x2, eps ? "); /* 入力促し */
18     scanf("%g%g%g", &x1, &x2, &eps); /* x1, x2, eps の入力 */
19
20     /* 注意：日本語キーボードの¥は、US キーボードのバックスラッシュ\となる：\n, \t 等 */
21     printf("\n i \tx1 \tx2 \txm \tf(xm) \n");
22     i=0; /* カウンター(i)の零設定 */
23
24     while( (x2-x1)>eps ) { /* 区間幅が eps 以下になるまで区間分割を繰り返す */
25         i+=1; /* カウンター(i)に1を加える */
26         xm=(x1+x2)/2; f1=f(x1); fm=f(xm); /* xm: x1 と x2 の中点座標 */
27         printf("%2d %15.6e %15.6e %15.6e %15.6e\n", i, x1, x2, xm, fm);
28
29         if( f1*fm < 0 ) { /* f(x1) と f(xm) が異符号ならば、解は区間[x1, xm]にある */

```

```

30         x2=xm;                               /* 区間[x1, xm]を新たに区間[x1, x2]とする */
31     }else{                                     /* f(x1)とf(xm)が同符号ならば, 解は区間[xm, x2]にある */
32         x1=xm;                               /* 区間[xm, x2]を新たに区間[x1, x2]とする */
33     }
34 }
35 printf("\nA root found between %g and %g\n", x1, x2); /* 解の出力 */
36 }

```

【プログラム 3.2.2】

```

1 /* bisectional method to solve f(x)=0 */
2
3 #include <stdio.h> /* いつものおまじない : 標準入出力用ヘッダーファイル */
4 #include <math.h> /* 数学関数用ヘッダーファイル : 関数 sin, cos を使用している所以需要 */
5
6 /* 関数を main 関数よりも前に置けば, 関数のプロトタイプ宣言は必要なし */
7 float f(float x) /* 関数 f(x)= exp(-x) - sin x ; ANSI 規格では, 引数欄に変数の型宣言を記述 */
8 {
9     return( exp(-x) - sin(x) );
10 }
11
12 float sign(float r) /* 符号を設定する関数 */
13 {
14     if( r==0 ){
15         return(0.0);
16     }else if( r>0 ){
17         return(1.0);
18     }else{
19         return(-1.0);
20     }
21 }
22
23 void main(void) /* ANSI 規格では main 関数の記述は void main(void) */
24 {
25     float x1, x2, eps, f1, xm, fm;
26     int i;
27
28     printf("initial interval [x1, x2] (x1<x2) and tolerance: x1, x2, eps ? "); /* 入力促し */
29     scanf("%g%g%g", &x1, &x2, &eps); /* x1, x2, eps の入力 */
30     /* 注意 : 日本語キーボードの¥は, US キーボードのバックスラッシュ\となる : \n, \t 等 */
31     printf("\n i\tx1\tx2\t\txm\t\tf(xm)\n");
32     i=0; /* カウンター(i)の零設定 */
33
34     while( (x2-x1)>eps ){ /* 区間幅が eps 以下になるまで区間分割を繰り返す */
35         i+=1; /* カウンター(i)に1を加える */
36         xm=(x1+x2)/2; f1=f(x1); fm=f(xm); /* xm: x1 と x2 の中点座標 */
37         printf("%2d %15.6e %15.6e %15.6e %15.6e\n", i, x1, x2, xm, fm);
38
39         if( sign(f1)*sign(fm) < 0 ){ /* f(x1)とf(xm)が異符号ならば, 解は区間[x1, xm]にある */
40             x2=xm; /* 区間[x1, xm]を新たに区間[x1, x2]とする */
41         }else{ /* f(x1)とf(xm)が同符号ならば, 解は区間[xm, x2]にある */
42             x1=xm; /* 区間[xm, x2]を新たに区間[x1, x2]とする */
43         }
44     }
45     printf("\nA root found between %g and %g\n", x1, x2); /* 解の出力 */

```

46 }

【実行例】

初期区間幅を [0, 1], 収束判定区間値を 10^{-6} とすると, プログラム 3.2.1, 3.2.2 とともに以下の結果を得る。

```
-----実行開始-----
initial interval [x1,x2] (x1<x2) and tolerance: x1,x2,eps ? 0.0 1.0 1.0e-6

i      x1      x2      xm      f(xm)
1      0.000000e+00  1.000000e+00  5.000000e-01  1.271051e-01
2      5.000000e-01  1.000000e+00  7.500000e-01 -2.092722e-01
3      5.000000e-01  7.500000e-01  6.250000e-01 -4.983585e-02
4      5.000000e-01  6.250000e-01  5.625000e-01  3.648015e-02
5      5.625000e-01  6.250000e-01  5.937500e-01 -7.220681e-03
6      5.625000e-01  5.937500e-01  5.781250e-01  1.449455e-02
7      5.781250e-01  5.937500e-01  5.859375e-01  3.603075e-03
8      5.859375e-01  5.937500e-01  5.898438e-01 -1.817276e-03
9      5.859375e-01  5.898438e-01  5.878906e-01  8.907820e-04
10     5.878906e-01  5.898438e-01  5.88672e-01 -4.637767e-04
11     5.878906e-01  5.88672e-01  5.883789e-01  2.133703e-04
12     5.883789e-01  5.88672e-01  5.886230e-01 -1.252363e-04
13     5.883789e-01  5.886230e-01  5.885010e-01  4.405871e-05
14     5.885010e-01  5.886230e-01  5.885620e-01 -4.059087e-05
15     5.885010e-01  5.885620e-01  5.885315e-01  1.733402e-06
16     5.885315e-01  5.885620e-01  5.885468e-01 -1.942886e-05
17     5.885315e-01  5.885468e-01  5.885391e-01 -8.847763e-06
18     5.885315e-01  5.885391e-01  5.885353e-01 -3.557188e-06
19     5.885315e-01  5.885353e-01  5.885334e-01 -9.118950e-07
20     5.885315e-01  5.885334e-01  5.885324e-01  4.107532e-07
```

A root found between 0.588532 and 0.588533

```
-----おしまい-----
```

先にニュートン法の実行例 1) で求めた数値解と同じ値が得られる。

3.3 多項式の効率的な計算アルゴリズム

n 次多項式

$$f(x) = a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n \quad \dots\dots\dots \textcircled{1}$$

の値を効率的に求めるアルゴリズムには以下のものがある。

ホーナー法： ①式は

$$f(x) = ((\dots(((a_0 x + a_1) x + a_2) x + a_3) \dots + a_{n-1}) x + a_n) \quad \dots\dots\dots \textcircled{2}$$

と書き直されることから、以下のアルゴリズムにより $f(\alpha) = p_n$ の値を計算できる。

$$\left\{ \begin{array}{l} p_0 = a_0 \\ p_i = p_{i-1} \alpha + a_i \quad (i=1, 2, \dots, n) \end{array} \right\}$$

組立て除法： ①式は

$$f(x) = a_0^{(n)} (x-\alpha)^n + a_1^{(n)} (x-\alpha)^{n-1} + a_2^{(n-1)} (x-\alpha)^{n-2} + \dots + a_{n-1}^{(2)} (x-\alpha) + a_n^{(1)} \quad \dots\dots\dots \textcircled{3}$$

と書き直すことができ、 $a_n^{(1)}, a_{n-1}^{(2)}$ は以下のアルゴリズムにより求まる。

$$\left\{ \begin{array}{l} a_0^{(1)} = a_0 \\ a_i^{(1)} = a_{i-1}^{(1)} \alpha + a_i \quad (i=1, 2, \dots, n) \end{array} \right\} \Longrightarrow a_n^{(1)}$$

$$\left\{ \begin{array}{l} a_0^{(2)} = a_0^{(1)} \\ a_i^{(2)} = a_{i-1}^{(2)} \alpha + a_i^{(1)} \quad (i=1, 2, \dots, n-1) \end{array} \right\} \Longrightarrow a_{n-1}^{(2)}$$

これは以下のように組立除法を二回行うことに相当する。

	$a_0^{(0)}$	$a_1^{(0)}$	$a_2^{(0)}$	\dots	$a_{n-2}^{(0)}$	$a_{n-1}^{(0)}$	$a_n^{(0)}$
+)		$a_0^{(1)} \alpha$	$a_1^{(1)} \alpha$	\dots	$a_{n-3}^{(1)} \alpha$	$a_{n-2}^{(1)} \alpha$	$a_{n-1}^{(1)} \alpha$
	$a_0^{(1)}$	$a_1^{(1)}$	$a_2^{(1)}$	\dots	$a_{n-2}^{(1)}$	$a_{n-1}^{(1)}$	$[a_n^{(1)}]$
+)		$a_0^{(2)} \alpha$	$a_1^{(2)} \alpha$	\dots	$a_{n-3}^{(2)} \alpha$	$a_{n-2}^{(2)} \alpha$	
	$a_0^{(2)}$	$a_1^{(2)}$	$a_2^{(2)}$	\dots	$a_{n-2}^{(2)}$	$[a_{n-1}^{(2)}]$	

③式より、

$$f(\alpha) = a_n^{(1)}$$

$$f'(\alpha) = a_{n-1}^{(2)}$$

であるから、結局 $f(\alpha) = p, f'(\alpha) = q$ の値は次のアルゴリズムにより計算できる。

【アルゴリズム】

組立て除法：

```

p := a0
q := p
for i=1 to n-1
    p := p*α + ai
    q := q*α + p
end
p := p*α + an
    
```

なお、ホーナー法のアルゴリズムは、組立除法により p を求めるアルゴリズムと同一のものである。

【問題】 n 次多項式 f(x) とその導関数 f'(x) を組立除法により求め、x の区間 $[x_{min}, x_{max}]$ におけるグラフをエクセル等で描く。

まず、多項式 f(x) の次数 n と係数 a_0, a_1, \dots, a_n を読み込んで関数を決定し、x の最小値 x_{min} と最大値 x_{max} 、およびその区間の分割数 m を読み込んで $\Delta x = (x_{max} - x_{min}) / m$ を算出し、

$$x_k = x_{min} + k \Delta x \quad (k=0, 1, 2, \dots, m)$$
 における関数値 $f(x_k)$ を計算し、出力する。
 これよりエクセル等によりグラフを作成し、関数の分布を視覚的に確認する。

【プログラム 3.3.1】

```
1 /* Efficient evaluation for polynomial and its derivative */
2
3 #include <stdio.h>
4
5 #define NMAX 10
6 #define KMAX 20
7
8 float a[NMAX+1];      /* external variables can be accessed by any function */
9 int n;
10
11 void eval(float x, float *f, float *df) /* computes p(x) and q(x)=p'(x) */
12 {
13     int i;
14     float p,q;
15
16     p= a[0];
17     for(q=p, i=1; i<n; i++) {
18         p=p*x+a[i];
19         q=q*x+p;
20     }
21     p=p*x+a[n];
22     *f = p;
23     *df= q;
24 }
25
26 void main(void)
27 {
28     float x,xmin,xmax,dx,f,df;
29     int i,m,k;
30
31     printf("degree of polynomial: n ? ");
32     scanf("%d", &n);
33     printf("coefficients of polynomial:\n");
34     for(i=0; i<=n; i++) {
35         printf("a[%d] ? ", i);
36         scanf("%g", &a[i]);
37     }
38     printf("x range [xmin, xmax] and division number m: xmin,xmax,m ? ");
39     scanf("%g%g%d", &xmin, &xmax, &m);
40     dx=(xmax-xmin)/m;
41
42     printf("\n k\t x\t f(x)\t f'(x)\n");
43
44     for(k=0; k<=m; k++) {
45         x=xmin+k*dx;
46         eval(x, &f, &df);
47         printf("%2d\t%12.5f\t%15.6f\t%15.6f\n", k, x, f, df);
48     }
49 }
```

【実行例】

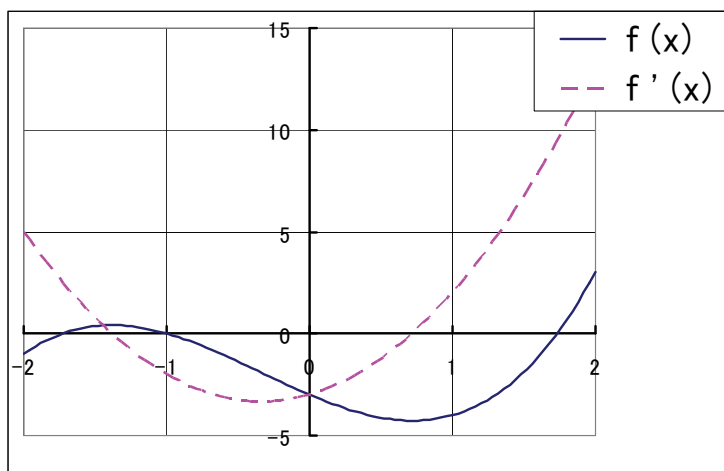
3次多項式 $f(x) = x^3 + x^2 - 3x - 3$ とその導関数の値を, 区間 $[-2, 2]$, 分割数100で出力する。

```
-----実行開始-----
degree of polynomial: n ? 3
coefficients of polynomial:
a[0] ? 1
a[1] ? 1
a[2] ? -3
a[3] ? -3
x range [xmin, xmax] and division number m: xmin,xmax,m ? -2 2 100
```

k	x	f(x)	f'(x)
0	-2.00000	-1.000000	5.000000
1	-1.96000	-0.807936	4.604800
2	-1.92000	-0.631488	4.219200
3	-1.88000	-0.470272	3.843200
4	-1.84000	-0.323904	3.476800
5	-1.80000	-0.192000	3.120000
6	-1.76000	-0.074176	2.772800
7	-1.72000	0.029952	2.435200
8	-1.68000	0.120768	2.107200
9	-1.64000	0.198656	1.788800
10	-1.60000	0.264000	1.480000
	⋮	⋮	⋮
	⋮	⋮	⋮
	⋮	⋮	⋮
95	1.80000	0.671999	10.319999
96	1.84000	1.095103	10.836799
97	1.88000	1.539071	11.363199
98	1.92000	2.004287	11.899199
99	1.96000	2.491135	12.444799
100	2.00000	2.999999	12.999999

```
-----おしまい-----
```

エクセルによるグラフ



グラフより, $f(x) = x^3 + x^2 - 3x - 3 = 0$ は3根を持つことがわかる。

【応用例】 多項式の効率的な計算アルゴリズムを用いて、 n 次多項式の根をニュートン法により求める。

【プログラム 3.3.2】

```

1 /* Newton's Method for f(x)=0 with efficient evaluation for a polynomial and its derivative */
2
3 #include <stdio.h>
4 #include <math.h>
5
6 #define NMAX 10
7 #define KMAX 20
8
9 float a[NMAX+1];      /* external variables can be accessed by any function */
10 int n;
11
12 void eval(float x, float *f, float *df) /* computes p(x) and q(x)=p'(x) */
13 {
14     int i;
15     float p,q;
16
17     p= a[0];
18     for (q=p, i=1; i<n; i++) {
19         p=p*x+a[i];
20         q=q*x+p;
21     }
22     p=p*x+a[n];
23     *f = p;
24     *df= q;
25 }
26
27 void main(void)
28 {
29     float x, f, df, d, eps;
30     int i, k;
31
32     printf("degree of polynomial: n ? ");
33     scanf("%d", &n);
34     printf("coefficients of polynomial:\n");
35     for (i=0; i<=n; i++) {
36         printf("a[%d] ? ", i);
37         scanf("%g", &a[i]);
38     }
39
40     while(1) {
41         printf("\ninitial guess and tolerance: x, eps ? "); /* EOF : 入力終了コードを検出 */
42         if( scanf("%g%g",&x,&eps)==EOF ) break; /* したときの関数scanfの戻り値 */
43                                             /* MS-DOSでは、終了コードの入力は、CTRL+Z */
44         printf("\n k \t x \t f(x) \t f'(x) \t correction \n");
45         for (k=0; k<KMAX; k++) {
46             eval(x, &f, &df);
47             d= -f/df;
48             printf("%2d\t%15.6e\t%15.6e\t%15.6e\t%15.6e\n", k, x, f, df, d);
49             x += d;
50             if(fabs(f)<eps) break;
51         }
52         if(k>=KMAX) printf("not convergent?\n");
53         printf("\nroot=%g\n", x);

```

```
54 }
55 }
```

【実行例】

直前の問題の実行例で分布を確認した関数 $f(x) = x^3 + x^2 - 3x - 3 = 0$ の3個の根を求める。

```
-----実行開始-----
degree of polynomial: n ? 3
coefficients of polynomial:
a[0] ? 1
a[1] ? 1
a[2] ? -3
a[3] ? -3

initial guess and tolerance: x, eps ? 1.5 1.0e-6      ..... 初期値1.5, 収束判定値 $10^{-6}$ の入力

k      x                f(x)      f'(x)      correction
0      1.500000e+00     -1.875000e+00  6.750000e+00  2.777778e-01
1      1.777778e+00     4.458163e-01  1.003704e+01 -4.441712e-02
2      1.733361e+00     1.240710e-02  9.480339e+00 -1.308719e-03
3      1.732052e+00     1.098788e-05  9.464116e+00 -1.161005e-06
4      1.732051e+00     -2.942129e-07  9.464101e+00  3.108725e-08

root=1.73205

initial guess and tolerance: x, eps ? -2 1.0e-6      ..... 初期値-2.0, 収束判定値 $10^{-6}$ の入力

k      x                f(x)      f'(x)      correction
0      -2.000000e+00     -1.000000e+00  5.000000e+00  2.000000e-01
1      -1.800000e+00     -1.919999e-01  3.120000e+00  6.153842e-02
2      -1.738461e+00     -1.642956e-02  2.589822e+00  6.343896e-03
3      -1.732118e+00     -1.695317e-04  2.536459e+00  6.683793e-05
4      -1.732051e+00     7.883410e-08  2.535898e+00 -3.108725e-08

root=-1.73205

initial guess and tolerance: x, eps ? 0 1.0e-6      ..... 初期値0, 収束判定値 $10^{-6}$ の入力

k      x                f(x)      f'(x)      correction
0      0.000000e+00     -3.000000e+00  -3.000000e+00 -1.000000e+00
1      -1.000000e+00     0.000000e+00  -2.000000e+00  0.000000e+00

root=-1

x (initial guess), eps (tolerence) ? ^Z (controlとZを同時に押す) ↵ (Enter Keyを押す)
..... 入力終了コードの入力
-----おしまい-----
```

3.4 連立非線形方程式に対するニュートン法

2 変数の連立非線形方程式

$$f(x, y)=0$$

$$g(x, y)=0$$

の根を求める。

【アルゴリズム】

```

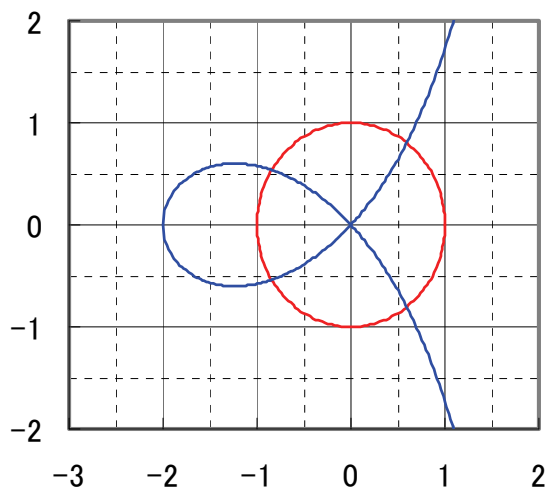
x, y := 与えられた初期値
for k=0, 1, 2, ...      {収束するまで繰り返す}
    r:=1.0/(fxgy-fygx)
    Δx:=-r( gyf-fyg)
    Δy:=-r(-gxf+fxg)
    x := x + Δx
    y := y + Δy
end
    
```

【問題】

下記の 2 曲線の 4 交点をニュートン法により求めよ。
初期値には、それぞれの交点に近い値を与えるよう留意せよ。

単位円 : $f(x, y)=x^2+y^2-1=0$

葉形線 : $g(x, y)=x^2(a+x)-y^2(a-x)=0$ (ただし $a=2$)



【プログラム 3.4.1】

```

1 /* Solve System of Nonlinear Equations: */
2 /*           f(x, y)=0, g(x, y)=0 */
3 /*           by Newtons' Method */
4
5 #include <stdio.h>
6 #include <math.h>
7
8 #define KMAX 20
9
10 void eval(float x, float y, float *f, float *g,
11          float *fx, float *fy, float *gx, float *gy)
    
```

```

12 {
13     float a=2.0;
14
15     *f= x*x+y*y-1;
16     *g= x*x*(a+x)-y*y*(a-x);
17     *fx= 2*x;
18     *fy= 2*y;
19     *gx= 3*x*x+2*a*x+y*y;
20     *gy=-2*y*(a-x);
21     return;
22 }
23
24 void main(void)
25 {
26     float eps, x, y, f, g, fx, fy, gx, gy, rden, dx, dy;
27     int k;
28
29     printf("tolerance: eps ? "); scanf("%g", &eps);
30     while( printf("initial guess: x, y ? "), scanf("%g%g", &x, &y) != EOF ) {
31         printf("\n k \t x \t y \t f(x, y) \t g(x, y)\n");
32         for(k=0; k<KMAX; k++) {
33             eval(x, y, &f, &g, &fx, &fy, &gx, &gy);
34             printf("%2d\t%15.6e\t%15.6e\t%15.6e\t%15.6e\n", k, x, y, f, g);
35             rden=1.0/(fx*gy-fy*gx);
36             dx= -rden*(f*gy-fy*g);
37             dy= -rden*(fx*g-f*gx);
38             x += dx;
39             y += dy;
40             if(fabs(dx)+fabs(dy)<eps) break;
41         }
42         if(k>=KMAX) printf("not convergent ?\n");
43         printf("Solutions%15.6g %15.6g\n", x, y);
44     }
45 }

```

【実行例】

```

-----実行開始-----
tolerance: eps ? 1.0e-6          ..... 収束判定値 $10^{-6}$ の入力
initial guess: x, y ? 1.0 0.5    ..... 初期値 (x, y)=(1.0, 0.5) の入力

k      x          y          f(x, y)  g(x, y)
0      1.000000e+00  5.000000e-01  2.500000e-01  2.750000e+00
1      6.756757e-01  8.986486e-01  2.641070e-01  1.520629e-01
2      6.004339e-01  8.082747e-01  1.382882e-02  2.316294e-02
3      5.931251e-01  8.051496e-01  6.319632e-05  2.257559e-04
4      5.930703e-01  8.051507e-01  4.412852e-08  -7.691604e-08
Solutions      0.59307      0.805151

```

```

initial guess: x, y ? 1.0 -0.5    ..... 初期値 (x, y)=(1.0, -0.5) の入力

k      x          y          f(x, y)  g(x, y)
0      1.000000e+00  -5.000000e-01  2.500000e-01  2.750000e+00
1      6.756757e-01  -8.986486e-01  2.641070e-01  1.520629e-01
2      6.004339e-01  -8.082747e-01  1.382882e-02  2.316294e-02
3      5.931251e-01  -8.051496e-01  6.319632e-05  2.257559e-04

```

```

4          5.930703e-01  -8.051507e-01  4.412852e-08  -7.691604e-08
Solutions  0.59307      -0.805151

```

initial guess: x, y ? -1.0 0.5 初期値 (x, y)=(-1.0, 0.5) の入力

```

k      x      y      f(x, y)  g(x, y)
0      -1.000000e+00  5.000000e-01  2.500000e-01  2.500000e-01
1      -8.518519e-01  5.462963e-01  2.409127e-02  -1.795011e-02
2      -8.430871e-01  5.379139e-01  1.471392e-04  -3.221282e-04
3      -8.430703e-01  5.378034e-01  2.364374e-08  -8.205114e-08
Solutions  -0.84307      0.537803

```

initial guess: x, y ? -1.0 -0.5 初期値 (x, y)=(-1.0, -0.5) の入力

```

k      x      y      f(x, y)  g(x, y)
0      -1.000000e+00  -5.000000e-01  2.500000e-01  2.500000e-01
1      -8.518519e-01  -5.462963e-01  2.409127e-02  -1.795011e-02
2      -8.430871e-01  -5.379139e-01  1.471392e-04  -3.221282e-04
3      -8.430703e-01  -5.378034e-01  2.364374e-08  -8.205114e-08
Solutions  -0.84307      -0.537803

```

initial guess: x, y ? ^Z (controlとZを同時に押す) ↵ (Enter Keyを押す)

..... 入力終了コードの入力

-----おしまい-----

参考文献

高倉葉子：数値計算の基礎—解法と誤差—，コロナ社（2007）

森口繁一，伊理正夫，武市正人 編：Cによる算法痛論，東京大学出版会（2000）

Heath, Michael T.: Scientific Computing, An Introductory Survey, McGraw-Hill (2002)