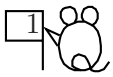
数値計算講義　第9 回
複素数の取り扱い – フラクタル図形の描画 –

カーネンコ　アレクセイ
金子　晃

kanenko@mbk.nifty.com
alexei.kanenko@docomo.ne.jp
http://www.kanenko.com/

FORTRAN における複素数型 (プログラム例 `mandelbrot.f`)



FORTRAN には複素数型が存在する

```
COMPLEX C,Z,W
```

と宣言すると,

```
W=Z**2+C
```

と書くだけで, 複素数の計算 $w = z^2 + c$ が実現できる.

複素数と実数や整数との混合算も OK. 結果は複素数型となる:

```
W=X*Z+2-1/Z
```

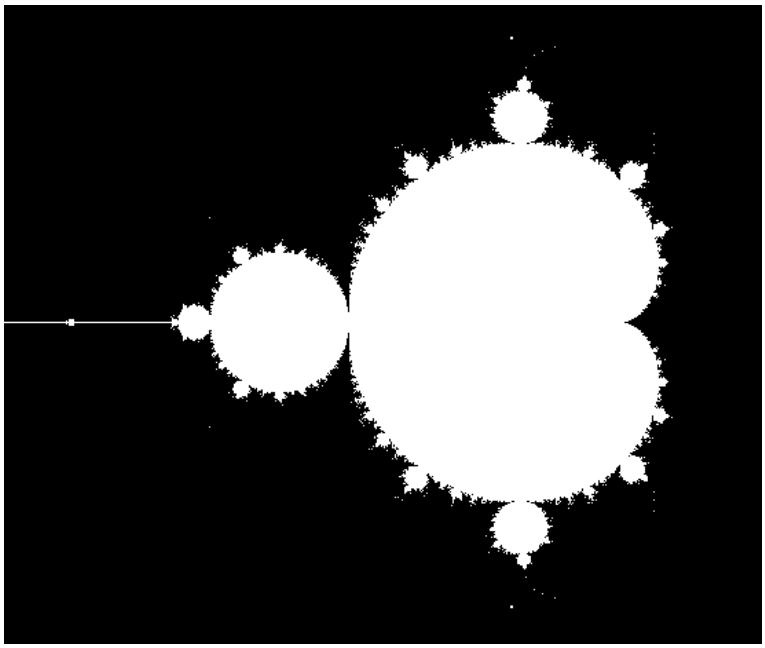
関連した函数:

```
Z=CMPLX(X,Y)   ! 実数 x, y から複素数 z = x+iy を作る
X=REAL(Z)      ! 複素数 z の実部 x を取り出す
Y=AIMAG(Z)     ! 複素数 z の虚部 y を取り出す
W=CONJG(Z)     ! 複素数 z の複素共役を作る
```

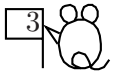
複素数に対する組み込み函数は次のような名前となる:

```
CABS(Z)        ! 複素数 z の絶対値
CEXP(Z)        ! 複素数 z の指数函数
CSIN(Z)        ! 複素数 z の sin
CCOS(Z)        ! 複素数 z の cos
```

mandelbrot.f の実行結果



Mandelbrot 集合の解説



Mandelbrot 集合は次の規則で定義される：
複素平面の点 $c \in \boldsymbol{C}$ が Mandelbrot 集合に属するとは,
原点 $z_0 = 0$ から出発し $z_{n+1} = z_n^2 + c$ という漸化式で定まる複素数列が
いつまで経っても有界集合に留まっていることをいう．

Mandelbrot 集合は $|z| \leq 2$ に含まれる．
$\because$ $|c| > 2$ なら $z_1 = c$, $|z_2| = |c^2 + c| = |c||c + 1| \geq |c|(|c| - 1)$.
以下帰納的に，$|z_n| \geq |c|(|c| - 1)^{2^{n-2}}$ なら, $|c| - 1 > 1$ に注意して

$$\begin{aligned}|z_{n+1}| &= |z_n^2 + c| \geq |c|^2(|c| - 1)^{2^{n-1}} - |c| = |c|\{|c|(|c| - 1)^{2^{n-1}} - 1\} \\ &> |c|\{|c|(|c| - 1)^{2^{n-1}} - (|c| - 1)^{2^{n-1}}\} \\ &= |c|\{(|c| - 1)^{2^{n-1}+1}\} > |c|\{(|c| - 1)^{2^{n-1}}\}\end{aligned}$$

よって $|z_n| \to \infty$ となる．

$|c| \leq 2$ であっても，一度 $|z_n| > 2$ となったら発散する．
$\because$ $|z_{n+1}| = |z_n^2 + c| \geq |z_n|^2 - |c| > 2|z_n| - 2 = 2(|z_n| - 1) > 2 \times 1 = 2$
(従って以後ずっと $|z_{n+k}| > 2$)
よって

$$|z_{n+1}| = |z_n^2 + c| \geq |z_n|^2 - |c| > |z_n|^2 - 2$$

$$\therefore \quad |z_{n+1}| - 2 > |z_n|^2 - 4 = (|z_n| + 2)(|z_n| - 2 > 4(|z_n| - 2)$$

となり，これを繰り返せば $|z_{n+k}| - 2 \geq 4^k(|z_n| - 2)$ が示せるので，
$|z_n| - 2 \to \infty$, 従って $|z_n| \to \infty$.
Q.E.D. 昨年までの証明はもう少し複雑でした．ここに紹介したのは
教科書の読者，鈴木春生氏による簡易化です．

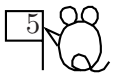
Mandelbrot 集合の FORTRAN プログラム


```
    PARAMETER(IXMIN=0,IXMAX=800,IYMIN=0,IYMAX=600)
    PARAMETER(XMAX=2,XMIN=-XMAX,YMAX=XMAX/IXMAX*IYMAX,YMIN=-YMAX)
    COMPLEX C,F,Z
    GX(IX)=XMIN+(XMAX-XMIN)*(IX-IXMIN)/(IXMAX-IXMIN)
    GY(IY)=YMIN+(YMAX-YMIN)*(IY-IYMIN)/(IYMAX-IYMIN)
    F(Z)=Z*Z+C
    CALL INIT(IXMIN,IYMIN,IXMAX,IYMAX)
    CALL CLS()
    DO 200 I=IXMIN,IXMAX
       DO 100 J=IYMIN,IYMAX
          C=CMPLX(GX(I),GY(J))
          Z=(0,0)               ! 複素定数 0
          DO 50 K=1,80
             Z=F(Z)
             IF (CABS(Z).GT.2.0) GO TO 100
 50       CONTINUE
          CALL PSET(I,J,15)
100    CONTINUE
200 CONTINUE
    PAUSE
    CALL CLOSEX()
    END
```

※ GX, GY は今まで用いてきた IGX,IGY の逆函数で,
整数型のスクリーン座標を実数型のワールド座標に変換する.

※ a, b が定数のときに限り (a,b) で複素数 $a + bi$ を表すことができる.

倍精度の複素数型 (プログラム例 mandelbrotd.f)



複素数はデフォールトでは実部・虚部とも単精度の実数である.
実部・虚部とも倍精度の実数である倍精度複素数は,
F77 のコンパイラでは必ずしもサポートされていなかったが,
FORTRAN 90 から正式に採り入れられ, g77 でも使える. 型宣言は

```
      DOUBLE COMPLEX  Z, W
```

あるいは

```
      COMPLEX*16  Z, W
```

などとする. 倍精度複素数に対しては, 対応する組み込み函数は

```
    CMPLX,CABS,CEXP,CSIN,CCOS,REAL,AIMAG
```

等はそれぞれ,

```
    DCMPLX,DCABS,DCEXP,DCSIN,DCCOS,DREAL,DIMAG
```

等となる.

- 倍精度の複素数を作るには, 構成要素の実数も倍精度にしなければ精度が保てない.
  函数 `Z=DCMPLX(X,Y)` は `X`, `Y` が倍精度でなく単精度でもエラーにならないので, 特に注意が必要.

C 言語における複素数の取り扱い (プログラム例 mandelakko.c)

C 言語には複素数型は無いので，実部・虚部二つの実数のペアで表し，自分で管理しなければならない．

```
double z[2],w[2],zeta[2];
/* 複素数の和 zeta = z + w に相当する計算 */
zeta[0]=z[0]+w[0];
zeta[1]=z[1]+w[1];
/* 複素数の積 zeta = z * w に相当する計算 */
zeta[0]=z[0]*w[0]-z[1]*w[1];
zeta[1]=z[1]*w[0]+z[0]*w[1];
```

もうちょっと見やすくするため，構造型を使うことができる：

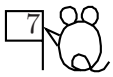
```
typedef struct {
    double x;  /* 実部のつもり */
    double y;  /* 虚部のつもり */
} complex;
```

のように型の定義をしておき，

```
complex z,w,zeta;
```

と複素数型の変数宣言すると，z の実部，虚部は構造型の規則によりそれぞれ z.x, z.y で表せる．こうしても複素数の演算が z+w 等と書ける訳ではなく，これらの演算を実行する函数を作らねばならない．

C 言語における複素数の取り扱い – 続き




```
/* 複素数の和 zeta = z + w を返す函数 */
complex cadd(complex z,complex w)[
    complex zeta;
    zeta.x=z.x+w.x;
    zeta.y=z.y+w.y;
    return zeta;
}
/* 複素数の積 zeta = z * w を返す函数 */
complex cmul(complex z,complex w)[
    complex zeta;
    zeta.x=z.x*w.x-z.y*w.y;
    zeta.y=z.y*w.x+z.x*w.y;
    return zeta;
}
```

これらを使うときは,

```
    complex alpha,beta,gamma,z,w,zeta;
    gamma=cadd(alpha,beta);
    zeta=cmul(z,w);
```

のようになり，ML の演算子のようで少々かっこうが悪い．(^^;

- gcc には，後述の C++ の機能を一部取り込む形で複素数型がある．

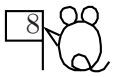
```
    #include<complex.h>
```

とすると，標準で倍精度の複素数型となり，

```
    complex z,w;
    z=2+1i;                     /* なぜか 2+i はエラーになる */
    w=1-2i;
    printf("%lf+%lf i\n",z+w);
    printf("%lf+%lf i\n",z*w);
```

で複素数の和や積が計算できる．C の標準仕様ではないので，互換性に注意．
複素数に対するライブラリ函数は cabs, conj 程度しかない．
出力フォーマットの書き方が見付からなかったが，上の間に合わせの例では
残念ながら 3.000000+-1.000000 i 等と出力される．

C++ 言語における複素数の取り扱い (プログラム例 julia.cc)



実は C 後継言語として C++ というものが有り，そこでは
複素数型が使える．
C++ は全く異なる言語だが, 複素数を使うだけなら, C の上位互換として
手っ取り早い使い方もできる：
① ソースファイル名を hoge.c から hoge.cc に変更する
② インクルードするヘッダファイルを

```
#include<cstdio>     /* C 言語の stdio.h の代わり */
#include<cstdlib>    /* C 言語の stdlib.h の代わり */
#include<cmath>      /* C 言語の math.h の代わり */
#include<complex>    /* C++ 言語の複素数型のクラス定義 */
```

としておけば，複素数型のみならず C の函数も引続き使える．その上で
③ 複素数型の宣言

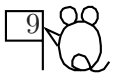
```
std::complex<double> z,w,zeta1,zeta2;
```

を行うと，演算は通常の演算形式

```
zeta1=z+w;    zeta2=z*w;
```

等で可能となる．
これは C++ 言語が演算の多重定義 (overload) を許す仕様のためである．
また C++ は型に厳密で，函数は引数の型が異なると同名でも別物とみなされる．
そのため，例えば指数函数も，単精度，倍精度，複素数のすべてに共通して
exp という一つの名前で済み (函数の多重定義)，数学により近くなる．

C++ 言語における複素数の取り扱い – 続き (おたく向け)



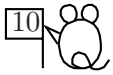
C++ 言語は C 言語とコンパイル時の函数名の付け方が異なる.
C 言語において函数 hoge の定義

```
int hoge(int n){
    return n+2;
}
```

を含むライブラリファイル libhoge.c を gcc -c libhoge.c でコンパイル
すると，できた object module libhoge.o の中に hoge という函数が
登録される.
これに対し C++ 言語の場合は同様のファイル libhoge.cc を g++ -c hoge.cc
でコンパイルすると, libhoge.o の中に hogei という函数が登録される.
(型の異なる同名の函数 hoge がもう一つあれば hogeii として登録される.
これにより函数の多重定義を実現している.)
これに合わせて, main 函数で hoge を呼んだとき, リンク時に
C 言語では hoge が,
C++ 言語では hogei, hogeii, 等の函数が
ライブラリから探される.
このため, C 言語で書かれたライブラリ函数をそのままリンクできない.
C 言語の別ファイルでコンパイルされている函数 int sub(int, int) 等を
C++ のソースから呼ぶときは，使用する函数の型宣言として

```
extern "C" {
    int sub(int,int);
    int sub2(double);
}
```

などとすればそのまま使え，既存のライブラリ遺産を活かすことができる.
C 言語用の xgrc.c を C++ で使うときも，このようにする.



- C++ の複素数演算を可能にしているのが，class の概念である．
complex (複素数) という class を新たに定義し，この型のオブジェクトに対して
既存の演算記号 + や * 等の意味を拡張定義する (演算子の overload).
複素数型には，実部・虚部が整数，単精度実数，倍精度実数のものなど
いろいろ必要で，これらを template という機構で区別する
(最初の例は double complex の場合で `complex<double>` と宣言されていた).
これに合わせて，演算子 + や * の定義も，
複素数対複素数, 複素数対倍精度実数，倍精度実数対複素数，複素数対整数，etc.
など，あらゆる可能性について与えておく必要がある．
更に，オブジェクトの生成と消滅，入出力フォーマットなどの定義も必要.
class ライブラリは，使う方は極楽だが，作る方は地獄.

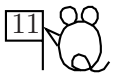
- complex など，標準的に使われるものについては，標準ライブラリ
std が用意されている．
同じ名前の class が衝突しないよう，namespace という概念で区別し，
標準クラスライブラリを使うときは `std::` をかぶせるか，
`using namespace std` と冒頭で宣言しなければならない.
(ver.2 までは省略できたが，ver.3 以降は明白に宣言しないとエラーになる．)
クラス complex の詳細は定義ファイル (5 階の場合は
`/usr/include/c++/4.0.0/complex`) を覗いて見よ.
(Java 言語を学んだ人は，オブジェクト指向言語としての共通性から，
理解しやすいだろう．)

- `julia++.cc` は C++ 言語固有の書き方になっている．例えば，

`#include<cstdio>`	の代わりに	`#include<iostream>`
`printf("message");`	の代わりに	`std::cout << "message";`
`scanf("%lf",&a);`	の代わりに	`std::cin >> a;`

など．この入出力の書き方は，フォーマットを気にせずに済むので便利.
(FORTRAN 77 の `WRITE(*,*)` と似てる!)
また，引数の参照渡しが可能なので scanf 函数のように引数で値を返さねば
ならない場合もポインターを使わずに済み，プログラムが安全になる.

Julia 集合の解説



Julia 集合はパラメータ c に依存し，次の規則で定義される：
複素平面の点 $z \in \boldsymbol{C}$ がパラメータ c の Julia 集合に属するとは，
$z_0 = z$ から出発し $z_{n+1} = z_n^2 + c$ という漸化式で定まる複素数列が
いつまで経っても有界集合に留まっていることをいう．

- Julia 集合は $|z| \leq 2$ に含まれる．従って発散の判定も $|z_n| > 2$ で可能．
証明は Mandelbrot 集合の場合で実質的に済んでいる．

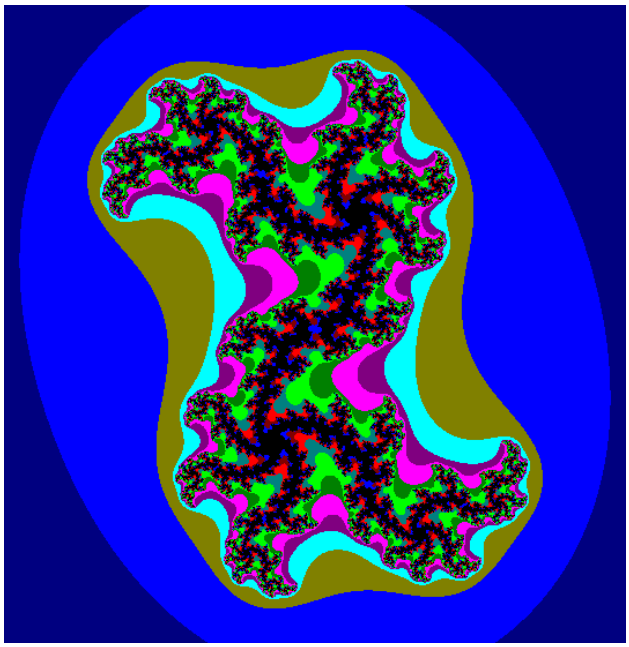
- Julia 集合はパラメータ c を変化させると，形が非常に変化する．

Mandelbrot 集合も Julia 集合も，単に有界に留まるか，発散するかで
白黒に塗り 分けるのでなく，$|z| > 2$ の領域に飛び出すまでに何回かかったか
で色分けすると，非常に美しい模様ができる．色づけには美的才能が必要．
C 言語で書かれた `mandelakko.c` を味わって見よ．

フラクタル性

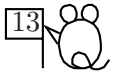
Mandelbrot 集合も Julia 集合も 部分を拡大してゆくと，
いくらでも複雑になっているだけでなく，同じようなパターンが
繰り返し 現れる (自己相似性).
これを調べるため，改良版のプログラムでは，マウスで描画範囲を 指定し，
その部分を拡大描画できるようにしてある．

julia.f の実行結果



$c = 0.4 + 0.31i$

Newton 法の吸引領域の解説



複素数に対しても Newton 法は通用する.
実は一般に N 元連立の非線型方程式系 $F(\boldsymbol{x}) = \mathbf{0}$ に対しても
Newton 法が通用する:

$\boldsymbol{x}_0$ (初期値) を適当に選ぶ.
$\boldsymbol{x}_1 = \boldsymbol{x}_0 - DF(\boldsymbol{x}_0)^{-1}[F(\boldsymbol{x}_0)]$
.....
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - DF(\boldsymbol{x}_n)^{-1}[F(\boldsymbol{x}_n)]$

この漸化式は，近似解 $\boldsymbol{x}_n$ における方程式 $F(\boldsymbol{x}) = 0$ の線型近似

$$F(\boldsymbol{x}) \doteqdot F(\boldsymbol{x}_n) + DF(\boldsymbol{x}_n)[\boldsymbol{x} - \boldsymbol{x}_n] = \mathbf{0}$$

を $\boldsymbol{x}$ について解いたものである.
($DF(\boldsymbol{x}_n)$ は写像 F の点 $\boldsymbol{x}_n$ における微分すなわち Jacobi 行列,
$DF(\boldsymbol{x}_n)^{-1}$ はその逆行列を表す.)
D^2F が非退化なら，初期値が真の解に十分近ければ，近似解の列が
真の解に2 次の収束をすることが，1 変数の場合と全く同様にして示される.
複素方程式 $f(z) = 0$ に対する Newton 法は，この $N = 2$ の場合にすぎない:

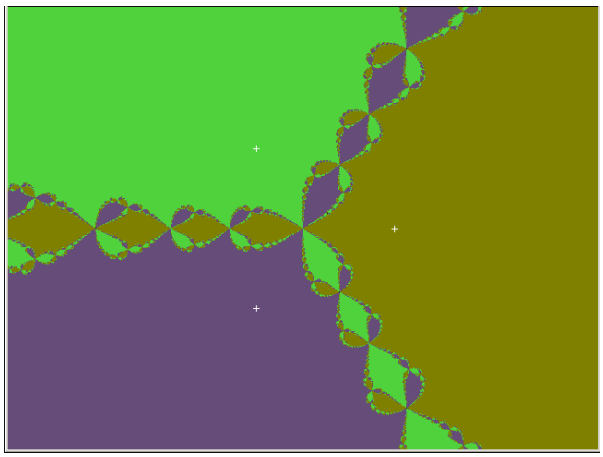
$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

Newton 法の吸引領域の解説 – 続き

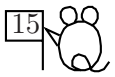
$f(z)$ が多項式のときは，反復列が $f'(z)$ の零点にぶつかるか，
周期軌道にぶつかるという例外的な場合を除き，必ずいつかは
$f(z) = 0$ のどれかの根に収束する．
全複素平面を，その点を初期値とする Newton 法の反復列がどの根に
収束するかにより色分けすると，美しい図ができる．
この図形も，$f'(z) = 0$ の零点の逆像の周りにフラクタル的な構造を持つ．

プログラム例:

- 第7 回に配布した
  `num7-3.f` は
  Wilkinson 多項式に対する
  Newton 法の吸引領域を
  可視化したもの.
- `C++` 言語の例としては
  $z^3 - 1 = 0$ に対する
  `cubic.cc`
  (右図).



参考: FORTRAN と C のリンク (おたく向け)



FORTRAN で書いたプログラムをコンパイルすると,

```
CALL HOGE(X)
```

というサブルーチンコールは, `hoge_` という副プログラム名に翻訳される.
引数の渡し方は, スタック経由なので, 参照渡しになっていることさえ
留意すれば, 副プログラムの方を, この名前で C で書いてもリンク可能.
xgrf.c の FORTRAN 用描画ライブラリは実際このようにして作られている.
この他にも, FORTRAN では書けないが, FORTRAN でも使いたい
システムコール関連の指令を, C で書いておき FORTRAN から
サブルーチンコールして使うことができる.

例えば:

- 時刻呼び出し
- CPU の切り捨て・切り上げ指定
- キーボードでキー押し下げの即時反映
- 環境変数の値の取得

など.

高速な FORTRAN のライブラリを作るのに, サブルーチンを
アセンブラで書いてリンクすることも, 同じ要領で可能となる.

本日の講義内容の自習課題

[1] FORTRAN 見本プログラム `mandelbrot.f` および
`mandelbrotd.f` のソースをエディターで読みながら，
`g77 mandelbrot.f xgrf.o -lX11 -L/usr/X11R6/lib` 等で
コンパイルし，実行して講義で学んだことを確認する．

[2] C 言語の見本プログラム `mandelakko.c` のソースを
エディターで読みながら，
`gcc mandelakko.c xgrc.o -lX11 -L/usr/X11R6/lib` で
コンパイルし，実行して講義で学んだことを確認する．

[3] `C++` 言語の見本プログラム `julia.cc` と `julia++.cc` のソースを
比較しながら読み，
`g++ julia.cc xgrc.o -lX11 -L/usr/X11R6/lib`
等でコンパイル・実行して講義で学んだことを確認する．

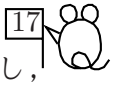
[4] `C++` 言語の見本プログラム `cubic.cc` のソースをエディターで読み，
コンパイルし，実行して講義で学んだことを確認する．

[5] `julia.f` をいろんなパラメータで実行し，できればプログラムを
変更して色付けも変え，気に入った図形を作ってみよ．どうしても
僕に見せたい素敵な図ができたら，メールで送ってください．
僕を感動させることができれば，レポートとしてカウントします．

[6] `cubic.cc` を $x^3 - 1 = 0$ の代わりに別の代数方程式, 例えば
$x^5 - x - 1 = 0$ に対するものに書き直してみよ．
[ヒント： 根の数が3 から5 に増えるので，それに応じて
解の近似値の格納や，色指定のための配列も拡大する必要がある．
(最初から大きめにとっておけば安直に汎用化できる (^^;)]

[7] フラクタルの参考プログラム `leaf.f`, `von_koch.f`,
`peano.c`, `sierpinski.c` を `xgrf.o` あるいは `xgrc.o` と
リンクしてコンパイルし, 実行してみよ．

本日の範囲の試験予想問題



問題 9.1 C 言語で複素数の計算を行う方法を一つ説明せよ．ただし，gcc の complex ライブラリのようなものは用いないものとする．

問題 9.2 次はマンデルブロート集合を描画する FORTRAN プログラムの一部である．これを C 言語に翻訳せよ．ただし，GX(), GY() はスクリーンのピクセル座標をワールド座標 $-2 \leq x \leq 2$, $-1.5 \leq y \leq 1.5$ に変換する線形関数であり，PSET() はピクセル (I,J) を着色するサブルーチンである．これらは C 言語に翻訳後もそのまま用いてよいものとする．

```
      COMPLEX C,Z
      ...
      DO 200 I=0,800
         DO 100 J=0,600
            C=CMPLX(GX(I),GY(J))
            Z=(0,0)
            DO 50 K=1,80
               Z=Z**2+C
               IF (CABS(Z).GT.2.0) GO TO 100
 50         CONTINUE
            CALL PSET(I,J,15)
100      CONTINUE
200   CONTINUE
```

問題 9.3 $z^3 - 1 = 0$ の根で, 単位円周上 1 のすぐ次に現れるもの (1 の原始 3 乗根 ω) の近似値を複素 Newton 法で計算するとき, 初期値 i から出発して 2 回反復したときの値を求めよ．