JAPLA 研究会資料 2011/9/24

Julia グラフィックスに向けての複素数写像変換

西川　利男

１．はじめに

ジュリア図形をゆっくり楽しみたいと思っている。これはフラクタルなる構造を持つものとして、マンデルブロート図形と並んであげられている。

「初めてのフラクタル」[1]を翻訳したときは、「めくら蛇におじず」で無我夢中でやったが、その内容を深く理解したわけでなく、今では冷や汗ものである。

[1] H.ラウヴェリエール著、西川利男訳「初めてのフラクタル」丸善(1996)

ジュリア図形のベースとなる複素数の写像変換については、図形が描ければよいということでプログラムを作ってきた。あらためて、例えばチャーチルの複素関数論[2]の本をひもといて勉強し直している。

[2] R.V.チャーチル、J.H.ブラウン著、中野実訳「複素関数入門」第４版
マグローヒル出版(1990).

複素数とはふしぎなものである。虚数なる名前のために、合点がいかない人が少なからずいる一方で、波動、振動、電磁気学、量子力学、あるいは等角写像を通して飛行機の設計など、複素数なしでは現代の科学、技術は成り立たない。ジュリア図形もその一つである。

Julia グラフィックスを行うには、いくつかの準備が必要である。本格的な Julia グラフィックスは次回にまわし、今回はその準備として、複素数の写像変換とそれにより、Julia 図形がどのようにして現れるか原理的なところを述べる。

２．Ｊは複素数の演算に極めて有能である！

Ｊでは複素数の演算はいとも容易である。複素数のままで四則演算はもちろんべき乗、べき乗根もそのまま計算される。

また、複素数の生成、変換なども自由自在に行われる。

```
              ２組の数          複素数              ２組の数
直交形式      x, y   ──→  z =. x j. y  ──→  'X Y' =. +. z     グラフ表示

極形式        r, t   ──→  z =. r r. t  ──→  'R T' =. *. z
              r 絶対値
              t 偏角
                偏角の変換      ラジアン(t)  t =. d % (1r180p1)
                                度(d)        d =. (1r180p1) * t
```

ここでは便利なので、数学表記に併せて、Ｊの表記も適宜用いる。

３．複素数の演算とグラフ表示

複素数とは実数と虚数とで出来た複合数と見る代わりにつぎのように考えたらよい。１つのものの性質を表わすのに、２つの値が必要な場合があってもよいだろう。それがある定まった演算規則を満たせばよいのである。これが複素数であると考えるのである。

例えば、平面上の１つの点の位置は２つの値の組で表わされる。このときも和や積の演算は行われる。２つの複素数の和は平面上では２つのベクトルを合成したものになる。

複素数をグラフ上の点として視覚化させることは、複素数の理解にとって極めて有効であり、これはガウス座標と呼ばれる。

複素数に関数を作用させてみる。これを次のように記す。

```
w =. func(z)
```

ここでｚとｗとは複素数であり、Ｊの表記を用いればつぎのようになる。

```
z =. xjy, w =. ujv
```

例えば、関数として２乗をとって、いくつかの値に対して演算をしてみよう。

```
func(z) =. *: z
```

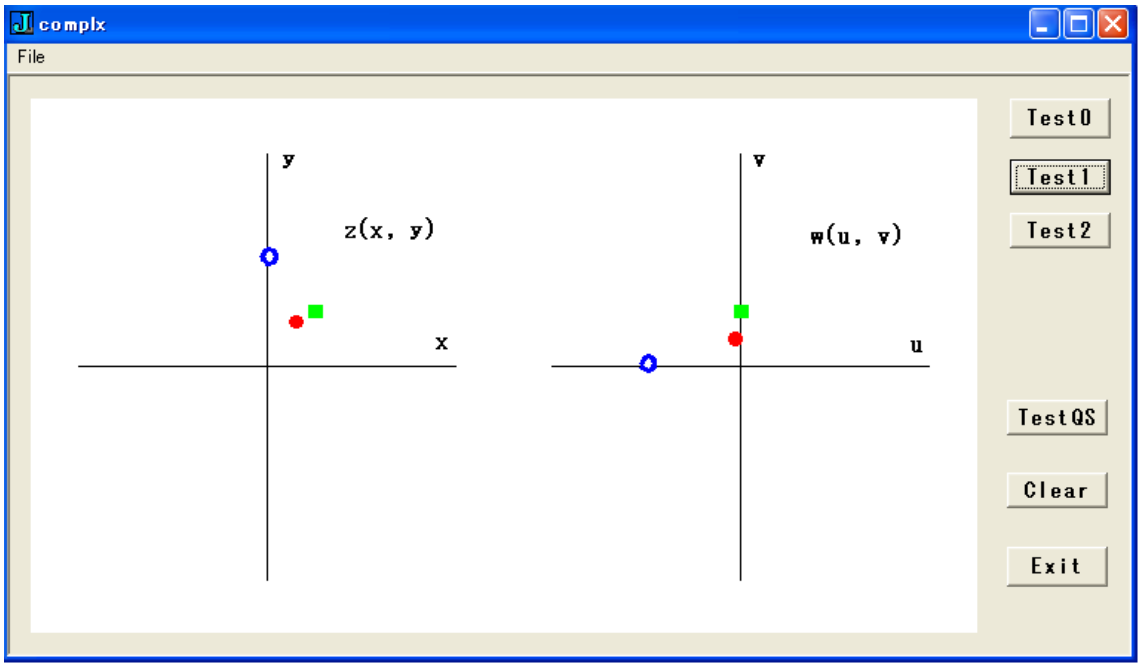
として

```
z0 =. 0.3j0.4   では
w0 =. func(z0) =. *: 0.3j0.4 =. 0.09 + 0j0.24 + _0.16 =. _0.07j0.24

z1 =. 0.5j0.5   では
w1 =. func(z1) =. *: 0.5j0.5 =. 0.25 + 0j0.5 + _0.25=. 0j0.5

z2 =. 0j1       では
w2 =. func(z2) =. *: 0j =. _1j0
```

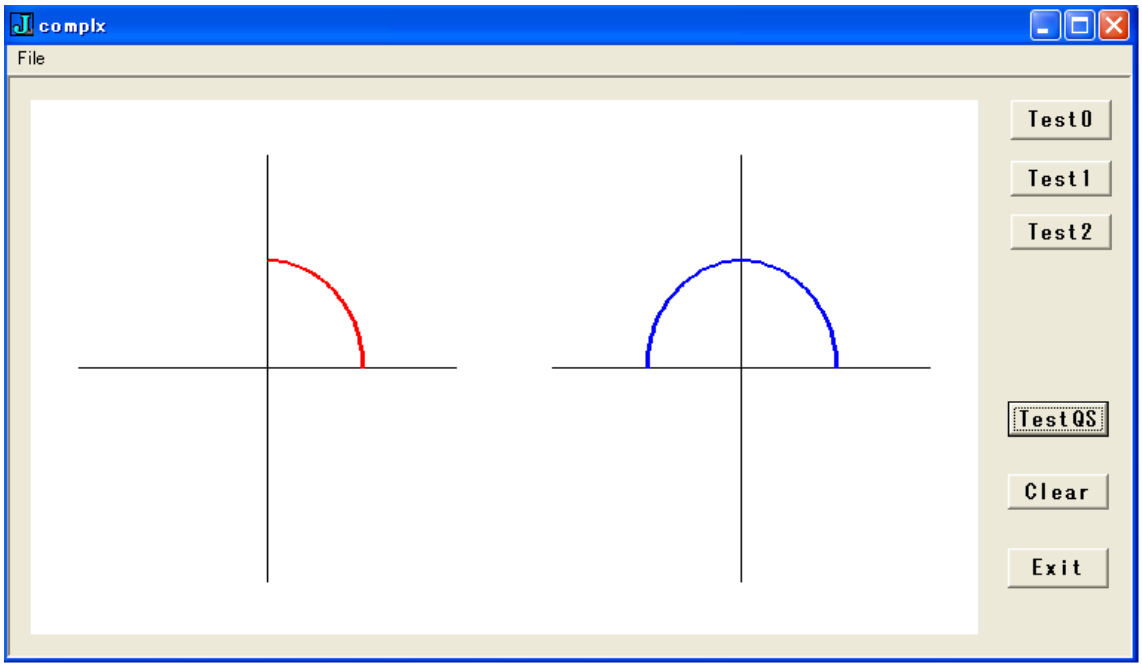
のようになる。もちろん、Ｊでは最後の結果だけが得られる。

このようすをガウス座標のグラフで見てみよう。このように複素数に関数を作用させると、ガウス座標表示ではそれぞれの値に対して、１対１で変換して対応する。

このような意味で複素数の写像(Complex Mapping)という語が使われる。



複素数の集合に対しても、写像は行われる。このように２乗に対しては、四半円弧は半円弧に写像される。先に紹介したチャーチルの本[2]には、このような写像の図が多数のせられている。



４．複素数の写像から Julia 図形へ

ここまでくれば、Julia 図形の実現はあともう一歩である。ある複素関数をきめて、ーこれが Julia 図形を決める最も重要なポイントであるがー、いろいろな複素数に対して、関数の写像を求め、そこで得た複素数に対してさらに関数の写像を求める。この操作を続ける。これにはＪのプリミティブ、パワー(^:)が威力を発揮する。

具体的にやってみよう。関数としては２乗にわずかな一定の複素数を加える。

```
   func1 =. 3 : '_0.4j_0.6 + *: y.'
```

次のようにしていろいろな初期値で、やってみる。

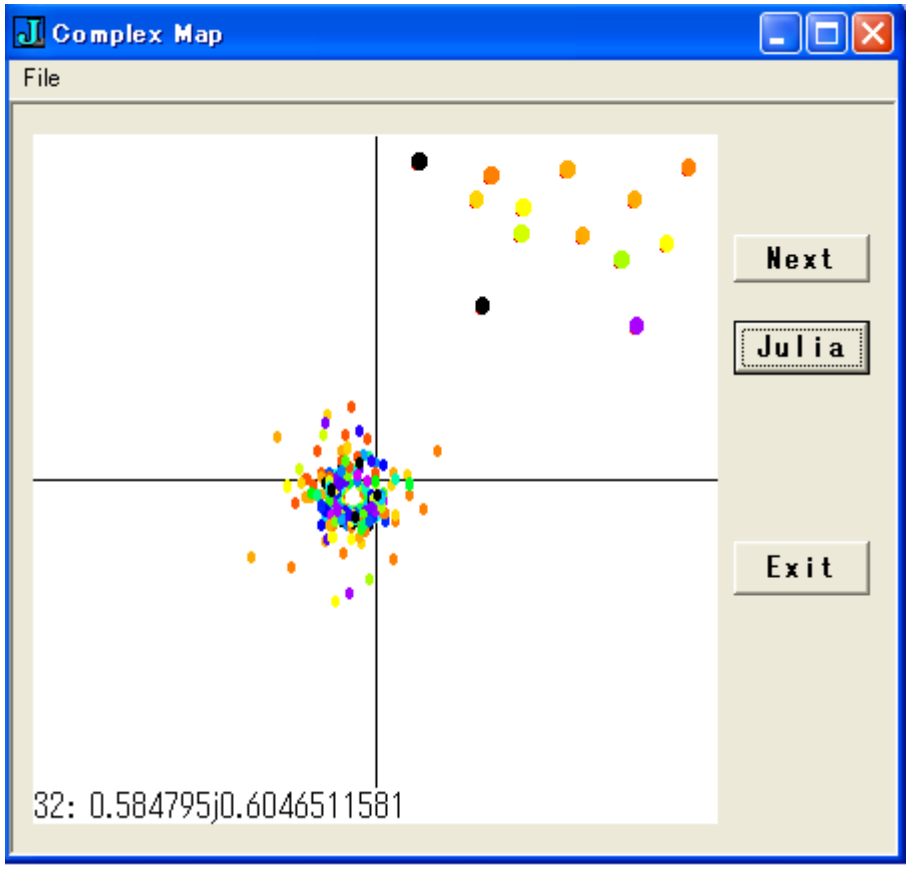
```
   z0 =: 0.3j0.4
   func1 z0
_0.47j_0.36
   func1 func1 z0
_0.3087j_0.2616
   func1^:(i.5) z0
0.3j0.4 _0.47j_0.36 _0.3087j_0.2616 _0.373139j_0.438488 _0.453039j_0.272766

   func1^:(i.5) 0.6j0.6
0.6j0.6 _0.4j0.12 _0.2544j_0.696 _0.819697j_0.245875 0.211448j_0.196914
   func1^:(i.5) 0.7j0.7
0.7j0.7 _0.4j0.38 _0.3844j_0.904 _1.06945j0.0949952 0.734705j_0.803186
   func1^:(i.5) 0.8j0.8
0.8j0.8 _0.4j0.68 _0.7024j_1.144 _1.21537j1.00709 0.0628921j_3.04798
   func1^:(i.5) 0.9j0.9
0.9j0.9 _0.4j1.02 _1.2804j_1.416 _0.765632j3.02609 _8.97105j_5.23375
```

これらの実験から、わかるように初期値の複素数が 0.3j0.4 や 0.6j0.6 のときには、関数 func1 を繰り返し作用させると、しだいに一つの値に収束していく。

一方、初期値が 0.8j0.8 や 0.9j0.9 では、すぐに大きな値に発散してしまう。途中の 0.7j0.7 ではしばらくしてからやがて発散するようである。

このような繰り返し写像の問題で、収束してゆくものは attractor と呼ばれ、発散してゆくものは repeller と呼ばれる。その程度の差を表すのに、初期値の複素数の位置を色の差で表した図形が Julia 図形である。

このようすを１点ずつ行ってみた Julia の原理を示すグラフィックスをおめに掛けよう。



実行は、マウスの左ボタンで初期値を決める。次に Next ボタンを押すと、Julia 関数にしたがって計算された写像の位置が現れる。さらに Next ボタンを押すと、次の写像の位置が示される。これを次々に行うとその回数に応じた色で表示される。最後には Julia ボタンを押すと、初期値の位置は発散する直前の色であららされる。このようにして、１点ずつの Julia 図形が作られることになる。

これを定められた複素数の範囲ですべて行う、このようにして出来る Julia 図形のグラフィックスは来月の JAPLA 例会へのお楽しみとしよう。ご期待あれ！

```
NB. Complex Mapping for Julia
NB.   2011/9/14

require 'trig'
require 'gl2'

MAP=: 0 : 0
pc map;pn "Complex Map";
menupop "File";
menu new "&New" "" "" "";
menu open "&Open" "" "" "";
menusep ;
menu exit "&Exit" "" "" "";
menupopz;
xywh 180 48 34 12;cc ok button;cn "Julia";
xywh 180 97 34 12;cc cancel button;cn "Exit";
xywh 5 7 171 153;cc compmap isigraph;
xywh 180 29 34 11;cc Next button;
pas 6 6;pcenter;
rem form end;
)

map_run=: 3 : 0
wd MAP
NB. initialize form here
gllines 0 500 1000 500
gllines 500 0 500 1000
glshow ''
iCol =: 1
MOUSEON =: 0
wd 'pshow;'
)

map_close=: 3 : 0
wd'pclose'
)

map_cancel_button=: 3 : 0
map_close''
)

map_ok_button=: 3 : 0
gltextalign TA_BOTTOM
gltext (": iCol), ': ', (": Z0)
```

```
glrgb ctest 10 * iCol
if. iCol > 30 do. glrgb 0 0 0 end.
glbrush ''
glpen 8 0
'X0 Y0' =. +. Z0
xx =. 500 * 1 + X0
yy =. 500 * 1 + Y0
glellipse xx, yy, 16, 20
glshow ''
)

func0 =: *:

func =: 3 : 0
:
x. + *: y.
)

ctest =: 3 : 0
y =. 360 | y.
ct =. <: +/ y >: 60 * i. 6
NB. wr ct
rt =. 256 % 60
select. ct
  case. 0 do. 255, (<. rt * 60 | y), 0
  case. 1 do. (<. 255 - rt * 60 | y), 255, 0
  case. 2 do. 0, 255, (<. rt * 60 | y)
  case. 3 do. 0,  (<. 255 - rt * 60 | y), 255
  case. 4 do. (<. rt * 60 | y), 0, 255
  case. 5 do. 255, 0, (<. 255 - rt * 60 | y)
end.
)

map_Next_button=: 3 : 0
Z =: _0.4j_0.6 func Z
  gltextalign TA_BOTTOM
  gltext (": iCol), ': ', (": Z)

  gltextalign TA_BOTTOM
  gltext (": iCol), ': ', (": Z)
  iCol =: iCol + 1
  if. iCol < 30
    do.
      glrgb ctest 10 * iCol
    else.
```

```
      glrgb 0 0 0
  end.
  glbrush ''
  glpen 8 0
  'X Y' =. +. Z
  glellipse (500 + 100 * (X, Y)), 6, 12
glshow ''
)

map_compmap_mbldown=: 3 : 0
iCol =: 1
MOUSEON =: 1
d=. ". sysdata
x=. (0{d) * 1000 % (2{d)
y=. (1{d) * 1000 % (3{d)
glrgb 0 0 255
glpen 1 0
NB. glmove x, y
glshow ''
)

map_compmap_mmove=: 3 : 0
d=. ". sysdata
x=. (0{d) * 1000 % (2{d)
y=. (1{d) * 1000 % (3{d)
if. MOUSEON = 1
 do.
   glrgb 255 0 0
   glpen 8 0
   glellipse  x, y, 6, 12
   gltextalign TA_BOTTOM
NB.    gltext ": x, y
   X =. _1 + x % 500
   Y =. _1 * _1 + (1000 - y) % 500
   Z =: X j. Y
   gltext ": Z
   Z0 =: Z
 end.
glshow ''
)

map_compmap_mblup=: 3 : 0
MOUSEON =: 0
d=. ". sysdata
x=. (0{d) * 1000 % (2{d)
```

```
y=. (1{d) * 1000 % (3{d)
glshow ''
)
```