福山大学工学部紀要
第29巻 2005年12月

代数方程式の精度保証付き数値計算法

尾関 孝史* 山崎 耕司* 小林 富士男*

A numerical method of algebraic equation with guaranteed accuracy

Takashi OZEKI*, Koji Yamasaki*, Fujio KOBAYASHI*

ABSTRACT

In numerical calculation using computers, it is not possible to avoid the influence of rounding errors. Therefore solutions obtained by numerical calculation include some errors. To evaluate the error, the computation with guaranteed accuracy has been proposed. The purpose of this paper is to explain the method easily. At first, we explain a system of the floating-point number based on IEEE standard 754 in detail. Then, we introduce a library for interval calculations of complex numbers using a new standard of C language, that is, C99. Finally, we verify the effect of the numerical calculation with guaranteed accuracy by solving an algebraic equation.

キーワード: 数値計算，精度保証，代数方程式，IEEE 標準 754，C99

Keywords: Numerical Calculation, Guaranteed Accuracy, Algebraic Equation, IEEE standard 754, C99

1. まえがき

計算機を用いて数値計算を行うと，必ず丸め誤差が発生する．それは，計算機が有限の記憶領域を用いて計算を行うにも係わらず，実数のような無限を扱う際には，避けて通れない宿命である．また，解析的に解けない方程式を反復法を用いて計算機で解く場合，得られた近似解が真の解にどれほど近いであろうかと知りたくなる．更に，今，数値計算で解こうとしている方程式に解が存在するかどうかを数値計算自身で知ることができるか，といった疑問が生まれて来る．これらの問題を解決しようという試みが，ここ十年ほど，数値解析の研究分野では活発に行われている．特に，日本では，早稲田大学の大石先生を中心としたグループが精度保証付き数値計算を確立しようとしている [1–5]．多くの計算機では，実数を浮動小数点数として取り扱う．そこで，本論文では始めにIEEE 標準 754 に基づく浮動小数点数に関して説明する．次に，C 言語の新しい規格である C99 を用いて，複素数の区間演算 (加減乗除) を行うライブラリの作成方法について説明する．最後にこのライブラリを用いて，代数方程式の解の誤差解析を行い，精度保証付き数値計算の有用性を確認する．

2. IEEE 標準 754 の浮動小数点数

一般に，コンピュータで小数を扱う場合，２進浮動小数点数表示が用いられる．特に，パソコンやワークステーションでは，IEEE 標準 754 に基づく浮動小数点システムが標準的に用いられている．本論文で，数値計算の際に用いた，インテル社の Pentium 4 もこの規格に準拠している．この CPU では，浮動小数点数は，単精度では４バイト即ち３２ビット，倍精度では８バイト即ち６４ビットで表される．以下では，この処理系の倍精度浮動小数点数に関して詳しく解説する．表１で示すように，倍精度では，６４ビットを符号１ビット，指数部１１ビット，仮数部５２ビットに分けている．

ここで，符号は正の場合は０を取り，負の場合は１を

* 福山大学工学部情報処理工学科

表 1　浮動小数点数
Table1 Floating-point Number

1	11	52
符号	指数部	仮数部
0 正		
1 負		

取る．また，精度を最大限に保持するため 2 進数を用いており，更に**けち表現**を用いている．けち表現とは，仮数を必ず $1.***\cdots$ の形に**正規化**することで，「1.」の部分を省略し，小数点以下の部分のみを記憶することで 1 ビット分の情報を節約する方法のことである．従って，指数部を e，仮数部のビットを $d_1, d_2, \cdots, d_{52}$ とすると，浮動小数点数は

$$\pm\left(1+\frac{d_1}{2^1}+\frac{d_2}{2^2}+\cdots+\frac{d_{52}}{2^{52}}\right)\times 2^e \quad \cdots\cdots\cdots (1)$$

と表される．そして，指数部は 11 ビットあるため，0 ～ 2047 まで表現可能であるが，1.0 より絶対値が小さな数を扱うため，1023 引いて[1]，$-1022 \leq e \leq 1023$ の範囲を取る．なお，e が -1023 と 1024 の場合は，表 2 に示すように，特別な意味を持つ．例えば，-1.0 は符号部が 1，指数部が 1023 (01111111111) で，仮数部が全て 0 ($00\cdots00$) だから

$$1011111111100\underbrace{\cdots\cdots00}_{52\text{ 個}}$$

となる．この処理系では，正規数での絶対値最大数は指数部が 1023 で仮数部が全て 1 の場合の

$$\left(1+\frac{1}{2^1}+\frac{1}{2^2}+\cdots+\frac{1}{2^{52}}\right)\times 2^{1023}$$
$$=\left(2-\frac{1}{2^{52}}\right)\times 2^{1023}$$
$$\approx 1.79\times 10^{308}$$

である．一方，正規数での絶対値最小数は，指数部が -1022 で仮数部が全て 0 の場合の

$$1.0\times 2^{-1022}\approx 2.22\times 10^{-308}$$

である．また，この処理系における有効精度は，仮数部が 52 ビットであるから

$$2^{-52}\approx 2.22\times 10^{-16}$$

となり，10 進数では約 16 桁となる．これはこの処理系の機械エプシロンに等しい．従って，この処理系では，小数を 10 進数で表した場合，有効桁数が最大で 16 桁までしか保証されないため，17 桁以降の数値には意味がない．

[1]下駄履き表現と呼ばれる

表 2　倍精度浮動小数点数
Table2 Doble Floating-Point Number

仮数部 f	指数部 e	表される数
$f=0$	-1023 (全て 0)	± 0
$f\neq 0$	-1023 (全て 0)	$0.f\times 2^{-1022}$
	$-1022\leq e\leq 1023$	$1.f\times 2^e$(正規表現)
$f=0$	1024 (全て 1)	$\pm\infty$
$f\neq 0$	1024 (全て 1)	Nan (Not a number)

3. C99 における区間解析と丸めの制御

前節で述べたように，浮動小数点数は有限ビットを用いて小数を表現している．従って，表すことができない実数が存在する．例えば，0.1 は浮動小数点では表現できない．0.1 に最も近い 0.1 以下の数は

$$0.09999999999999999167\cdots$$

であり，0.1 に最も近い 0.1 以上の数は

$$0.10000000000000000551\cdots$$

である．しかし，これらの 2 つの浮動小数点数は仮数部が 1 ビット異なるだけである．このように，0.1 のような 2 進浮動小数点数で表現できない数が存在する条件下で，しかも計算誤差を取り扱うためには，数を点で扱うのではなく，その数を含む区間で扱う方法が提案されている．そして，数の計算を区間どうしの計算に置き換えたものを**区間演算**と呼ぶ．例えば，

$$[1,2]+[2,3]=[3,5]$$

といったように，区間どうしの計算では，計算した結果の区間は，区間内の数を計算した結果，可能性のある全ての計算結果の数を含む区間として得られる．この方法では，計算した結果の区間の端が浮動小数点数で取り扱えなくなる可能性がある．しかし，IEEE 標準 754 では計算時に丸め制御が行える命令が用意されている．更に，C コンパイラの新規格である C99(ISO/IEC 9899:1999) [6] では，丸めの制御を行う関数 fesetround() が用意されている．

例えば，今回用いた C コンパイラ (gcc-2.95.3-2vl22) は，C99 に準拠しているので，プログラム round.c

round.c

```
#include <stdio.h>
#include <fenv.h>

int main(void){
  double x_u,x_l,u_n;
  double x1 = 1.,x10 = 10.;
  printf("\n0.1の下への丸めは\n");
  fesetround(FE_UPWARD);
  x_l=x1/x10;
  printf("x_l=%1.20lf\n",x_l);
  printf("\n0.1の上への丸めは\n");
  fesetround(FE_DOWNWARD);
  x_u=x1/x10;
  printf("x_u=%1.20lf\n",x_u);
  printf("\n0.1の最近値への丸めは\n");
  fesetround(FE_TONEAREST);
  x_n=x1/x10;
  printf("x_n=%1.20lf\n",x_n);
  return 0;
}
```

を実行すると，

```
0.1の下への丸めは
x_l=0.09999999999999991673
0.1の上への丸めは
x_u=0.10000000000000005551
0.1の最近値への丸めは
x_n=0.10000000000000005551
```

を得る[2]．

また，C99では，複素数も標準で対応している．そこで，C99を用いて，複素数の区間解析を行なうライブラリを作成した．複素数の区間を複素平面上の長方形とし，2つの複素数を用いて，$I=[z_l,z_u]$で表す．ここで，図1のように，長方形の左下の複素数をz_l，右上の複素数をz_uとしている．そして，実際のプログラムではinterval.c

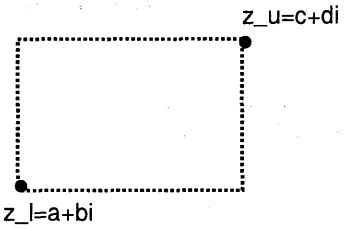

図1　複素数を表す区間
Fig.1 Interval of Complex Number

のように構造体INTERを用いている．

[2]コンパイル時に数学ライブラリのリンクが必要である．例えば，gcc -o round round.c -lm

interval.c

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

#define COMPLEX double _Complex

//---------------------------------------
// Definition of structure "INTER"
//---------------------------------------
typedef struct interval{
  COMPLEX lower;
  COMPLEX upper;
}INTER;

int main(void)
{
  COMPLEX p=-1.0+2.0*I,q=1.0+3.0*I;
  INTER z;
  z.lower=p;
  z.upper=q;
  printf("z.lower=% lf%+lfi\n",
          creal(z.lower),cimag(z.lower));
  printf("z.upper=% lf%+lfi\n",
          creal(z.upper),cimag(z.upper));
  return 0;
}
```

また，複素数に対する加減乗除の区間演算を以下のように定めた．2つの複素数の区間をI_1,I_2とした時，2つの区間集合を加減乗除して得られた集合を含む最小の長方形を考え，それを演算した結果の区間と定義する．ただし，除算において，I_2が零点を含む場合は除算不能である．付録に2つの複素区間の加減乗除を行なうプログラムを載せておく．

さて，区間演算では，

- 真の解を含む区間を得れば，真の解との誤差がどの程度かわかる．
- 観測誤差による初期定数の誤差を数値計算に考慮できる．

といった利点を持つが，

- 区間演算をすればするほど，区間幅が増大し，精度の意味がなくなるといった，区間膨張，区間爆発がしばしば起こる．

という欠点を持つ．

4. 事後誤差評価法

前述したように，区間演算法では，ϵの幅の区間をn個足すと，$n\epsilon$の区間に拡大してしまう．しかし，区間を用いない実際の数値計算では，切り上げ，切り下げの両方向の丸め誤差により，打ち消しあい，$\sqrt{n}\epsilon$程度になる．そこで，普通の計算で初めに近似解を得ておいて，後から真の解との誤差を区間演算法を用いて事後評価す

る，事後誤差評価法が提案されている．以下では，事後誤差評価法の原理を説明する．原因となる事象を x^* とし，それに A という物理現象が作用した結果，b という事象が観測されたとする (図 2 参照)．もし，作用素 A が

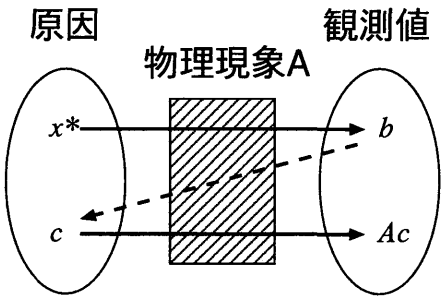

図 2　事後誤差評価法
Fig.2 After Evaluation of Error

有界作用素[3] であれば，

$$\|x^* - c\| = \|A^{-1}(b - Ac)\| \leq \|A^{-1}\|\|b - Ac\| \quad \cdot (2)$$

が成立するから，真の解 x^* と数値計算で求めた近似解 c の誤差 $\|x^* - c\|$ は $\|A^{-1}(b - Ac)\|$ を評価するか，逆作用素 A^{-1} のノルム $\|A^{-1}\|$ で評価できる．即ち，事後誤差評価法では，原因側での誤差 $\|x^* - c\|$ を求めるかわりに，計算可能な観測側の誤差 $\|b - Ac\|$ と，ノルム $\|A^{-1}\|$ を計算することができる．例えば，方程式 $f(x) = 0$ の零点 x^* を求める場合に反復法

$$x^{(i+1)} = x^{(i)} - Rf(x^{(i)}) \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (3)$$

を用いたとする．ここで，行列 R は 解 x^* のある近傍 $B(x^*, \delta)$ 内の x に対して，

$$\|I - Rf^{'}(x)\| < 1 \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (4)$$

を満たす行列 $f^{'}(x^{(i)}$ の近似逆行列である．ここで，

$$k = \max_{x \in B(x^*, \delta)} \|I - Rf^{'}(x)\| \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (5)$$

とすると，三角不等式により，任意のベクトル y に対して，x^* の近傍 $x \in B(x^*, \delta)$ では

$$\|(f^{'}(x))^{-1}y\| \leq \frac{\|Ry\|}{1-k} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (6)$$

が成立している．従って，平均値の定理を用いると，事後誤差評価法では，

$$\begin{aligned} \|x^* - x^{(i)}\| &= \|(f^{'}(\theta))^{-1}(f(x^{(i)}) - f(x^*))\| \\ &\leq \frac{1}{1-k}\|Rf(x^{(i)})\| \\ &= \frac{1}{1-k}\|x^{(i)} - x^{(i+1)}\| \quad \cdots\cdots\cdots\cdots (7) \end{aligned}$$

を得る [1]．

[3]バナッハ空間では，連続作用素と同値である [3]

5.　代数方程式の解の精度保証

n 次代数方程式の n 個の全ての近似解を求める方法として，Durand-Kerner-Aberth 法 [8](以下，DKA 法) がある．本節では，最初に DKA 法で近似解を求める．次に，Smith の定理と区間演算法を用いることで，この近似解誤差評価を行なう．

実係数の n 次方程式を

$$P(z) = z^n + a_1 z\, n - 1 + \cdots + a_n = 0 \quad (a_n \neq 0) \ (8)$$

とする．DKA 法は，最初に Aberth の初期値

$$z_k^{(0)} = -\frac{a_1}{n} + r_0 exp\left[\sqrt{-1}\left(\frac{2(k-1)\pi}{n} + \frac{\pi}{2n}\right)\right]$$
$$(k = 1, 2, \cdots, n) \quad \cdots\cdots (9)$$

により，$-\frac{a_1}{n}$ を中心とする同一円周上に n 個の初期近似解 $\{z_k^{(0)}\}$ を均等に取る．ここで，r_0 は n 次方程式 (8) の全ての根が閉円版 $|z + a_1/n| \leq r_0$ に含まれるような十分大きな値とする．次に，Durand-Kerner 法

$$z_k^{(i+1)} = z_k^{(i)} - \frac{P(z_k^{(i)})}{\prod_{j \neq k}(z_k^{(i)} - z_j^{(i)})}$$
$$(k = 1, 2, \cdots, n) \quad \cdots\cdots (10)$$

により，近似解 $\{z_k^{(i)}\}$ $(i = 0, 1, 2, \cdots)$ を更新する．求めた近似解は，次の Smith の定理 [8] で事後誤差評価する．

定理 1 (Smith) $z_1, \cdots, z_n$ を相異なる複素数とする．すると，n 次方程式 (8) の全ての根は閉円板

$$|z - z_k| \leq n\left|\frac{P(z_k)}{\Pi_{j \neq k}(z_k - z_j)}\right| \quad (k = 1, 2, \cdots, n) \ (11)$$

の合併に含まれる．特に，全ての閉円板が分離している場合には，各円板に 1 つずつ根が含まれる．

この定理の後半では，式 (7) の事後誤差評価法で，近似逆行列を

$$R = \frac{1}{\prod_{j \neq k}(z_k - z_j)} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (12)$$

と置いた時，

$$\frac{1}{1-k} \leq n \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (13)$$

が成立していることを意味している．

実際に 5 次方程式

$$z^5 - 10z^4 + 43z^3 - 104z^2 + 15 - z - 100 = 0 \quad \cdots\cdots (14)$$

の近似解を DKA 法で求め，事後誤差評価を行なった．この方程式の真の解は，$1 \pm 2i, 2, 3 \pm i$ である．初めに，通常の数値計算で，DKA 法を用いて，9 回の反復を行い，近似解 $z_1, \cdots, z_5$ を得た (表 3 参照)．次に，Smith の不

表 3　近似解とその誤差の上限
Table3 Approximations and Upper Limits of the Error

解	近似解 $z^{(9)}$	誤差
$3+i$	$3.0000000030+0.9999999894i$	5.5×10^{-8}
$1+2i$	$1.0000000035+2.0000000076i$	4.2×10^{-8}
2	$1.9999999934+0.0000000030i$	3.6×10^{-8}
$1-2i$	$0.9999999999-2.0000000000i$	2.0×10^{-12}
$3-i$	$3.0000000000-1.0000000000i$	4.2×10^{-10}

等式 (11) で，区間演算プログラムを用いて，右辺の上限を求めた．表 3 は，真の値とその近似解及び Smith の方法で得られた誤差の上限を示している．この表から，誤差の評価が正しくできていることがわかる．また，図 3 は DKA 法による収束の様子を表している．赤い線は近似解の収束を示し，青い点が真の解である．

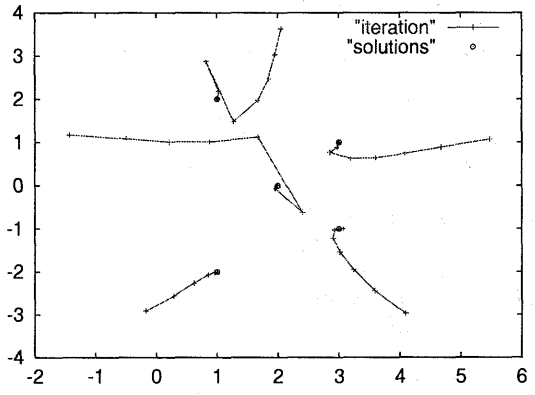

図 3　DKA 法の収束
Fig.3 Convergence of DKA Method

6. むすび

本論文では，浮動小数点数を区間で包むことで，精度を保証する数値計算法を紹介した．単純に区間演算を繰り返す，区間演算法ではしばしば，区間膨張が起こるため，実用的な精度が得られないことがある．これに対して，事後誤差評価法は，近似解の高い精度保証ができるだけでなく，解の存在をも数値計算で保証できる．本論文では，特に，C 言語の新しい規格である C99 を用いて，複素数の区間演算 (加減乗除) を行うプログラムを作成し，代数方程式の解の誤差解析を行ない，その有用性を確認した．

参考文献

[1] 大石進一，数値計算，裳華房，1999.

[2] 大石進一，“非線形現象の解析手法 [1]”，信学誌，Vol.79, No.2, pp.162–168, 1996.

[3] 大石進一，“非線形解析入門”，コロナ社，1997.

[4] 大石進一，“精度保証付き数値計算”，コロナ社，2000.

[5] 大石進一，“非線形問題を解く道具としての精度保証付き数値計算”，信学誌，Vol.84, No.1, pp.33–41, 2001.

[6] 林晴比古，新訂新 C 言語入門シニア編，ソフトバンク，2004.

[7] 伊理正夫，藤野和建，“数値解析の常識”，共立出版，pp.130–131，1985.

[8] 山本哲朗，数値解析入門 [増訂版]，サイエンス社，2003.

付録 加減乗除の区間演算プログラム

```
//--------------------------------------
//  Addition of Two Interval
//--------------------------------------
INTER cadd(INTER x, INTER y)
{
  INTER z;
  double a, b, c, d;

  fesetround(FE_DOWNWARD);
  a = creal(x.lower) + creal(y.lower);
  b = cimag(x.lower) + cimag(y.lower);
  z.l = a + b*I;

  fesetround(FE_UPWARD);
  c = creal(x.upper) + creal(y.upper);
  d = cimag(x.upper) + cimag(y.upper);
  z.u = c + d*I;

  fesetround(FE_TONEAREST);
  return z;
}

//--------------------------------------
//  Subtraction of Two Interval
//--------------------------------------
INTER csub(INTER x, INTER y)
{
  INTER z;
  double a, b, c, d;

  fesetround(FE_DOWNWARD);
  a = creal(x.lower) - creal(y.upper);
  b = cimag(x.lower) - cimag(y.upper);
  z.l = a + b*I;

  fesetround(FE_UPWARD);
  c = creal(x.upper) - creal(y.lower);
```

```
  d = cimag(x.upper) - cimag(y.lower);
  z.u = c + d*I;

  fesetround(FE_TONEAREST);
  return z;
}

//---------------------------------------
// Multiplication of Two Interval
//---------------------------------------
INTER cmul(INTER x, INTER y)
{
  INTER z;
  double _Complex comx[4];
  double _Complex comy[4];
  double _Complex comz[16];
  double a, b, c, d;
  int i, j, k;

  comx[0] = creal(x.lower) + cimag(x.lower)*I;
  comx[1] = creal(x.upper) + cimag(x.lower)*I;
  comx[2] = creal(x.lower) + cimag(x.upper)*I;
  comx[3] = creal(x.upper) + cimag(x.upper)*I;
  comy[0] = creal(y.lower) + cimag(y.lower)*I;
  comy[1] = creal(y.upper) + cimag(y.lower)*I;
  comy[2] = creal(y.lower) + cimag(y.upper)*I;
  comy[3] = creal(y.upper) + cimag(y.upper)*I;

  fesetround(FE_DOWNWARD);
  for(i=0, k=0; i<4; i++){
    for(j=0; j<4; j++){
      comz[k] = comx[i] * comy[j];
      k++;
    }
  }
  a = creal(comz[0]);
  b = cimag(comz[0]);
  for(k=1; k<16; k++){
    if(a > creal(comz[k]))
      a = creal(comz[k]);
    if(b > cimag(comz[k]))
      b = cimag(comz[k]);
  }
  z.l = a + b*I;

  fesetround(FE_UPWARD);
  for(i=0, k=0; i<4; i++){
    for(j=0; j<4; j++){
      comz[k] = comx[i] * comy[j];
      k++;
    }
  }
  c = creal(comz[0]);
  d = cimag(comz[0]);
  for(k=1; k<16; k++){
    if(c < creal(comz[k]))
      c = creal(comz[k]);
    if(d < cimag(comz[k]))
      d = cimag(comz[k]);
  }
  z.u = c + d*I;

  fesetround(FE_TONEAREST);
  return z;
}
```

```
//---------------------------------------
//  Division of Two Interval
//---------------------------------------
INTER cdiv(INTER x, INTER y)
{
  INTER z;
  double _Complex comx[4];
  double _Complex comy[4];
  double _Complex comz[16];
  double a, b, c, d;
  int i, j, k;

  if(creal(y.lower)<=0.0 && cimag(y.lower)<=0.0){
    if(creal(y.upper)>=0.0 && cimag(y.upper)>=0.0){
      printf("原点を含む区間で割ろうとしました。\n");
      exit(1);
    }
  }
  comx[0] = creal(x.lower) + cimag(x.lower)*I;
  comx[1] = creal(x.upper) + cimag(x.lower)*I;
  comx[2] = creal(x.lower) + cimag(x.upper)*I;
  comx[3] = creal(x.upper) + cimag(x.upper)*I;
  comy[0] = creal(y.lower) + cimag(y.lower)*I;
  comy[1] = creal(y.upper) + cimag(y.lower)*I;
  comy[2] = creal(y.lower) + cimag(y.upper)*I;
  comy[3] = creal(y.upper) + cimag(y.upper)*I;

  fesetround(FE_DOWNWARD);
  for(i=0, k=0; i<4; i++){
    for(j=0; j<4; j++){
      comz[k] = comx[i] / comy[j];
      k++;
    }
  }
  a = creal(comz[0]);
  b = cimag(comz[0]);
  for(k=1; k<16; k++){
    if(a > creal(comz[k]))
      a = creal(comz[k]);
    if(b > cimag(comz[k]))
      b = cimag(comz[k]);
  }
  z.l = a + b*I;

  fesetround(FE_UPWARD);
  for(i=0, k=0; i<4; i++){
    for(j=0; j<4; j++){
      comz[k] = comx[i] / comy[j];
      k++;
    }
  }
  c = creal(comz[0]);
  d = cimag(comz[0]);
  for(k=1; k<16; k++){
    if(c < creal(comz[k]))
      c = creal(comz[k]);
    if(d < cimag(comz[k]))
      d = cimag(comz[k]);
  }
  z.u = c + d*I;

  fesetround(FE_TONEAREST);
  return z;
}
```