数理解析研究所講究録
第673巻 1988年 47-61

The interval arithmetic for the ill-conditioned polynomial equation

Matu-Tarow NODA* and Tateaki SASAKI**

* Department of Computer Science, Faculty of Engineering, Ehime University, Matsuyama-shi, Ehime 790, Japan

** The Institute of Physical and Chemical Research Wako-shi, Saitama 351-01, Japan

1. Introduction

A polynomial is said to be ill-conditioned if small changes in its coefficients result in large changes in its zeros (Bareiss, 1967). An ill-conditioned polynomial equation has at least one of the following properties:

1) The existence of several roots having ratios close to unity.

2) The existence of multiple roots.

The property 1) means the existence of close zeros in a polynomial equation. On multiple roots, let the polynomial equation be

$$P_n(x)=a_n x^n+a_{n-1}x^{n-1}+\ldots+a_1 x+a_0 ,$$

and let m_i be a multiplicity of a solution x_i of $P_n(n)=0$. If a coefficient, a_k is perturbed slightly to $a_k+\Delta a_k$, then the perturbation in x_i is

$$x_i = -\left\{ \frac{m_i!\, x_i x^k \Delta a_k}{(d/dx)^{m_i} P_n(x_i)} \right\}^{1/m_i}$$

where $\Delta a_k \ll a_k$.

Many numerical algorithms have been proposed to obtain zeros of

polynomial equations. Most of them, however, is not effective for ill-conditioned problems. It is well known that the Newton's method and some modified methods can not calculate roots accurately if root are multiple. As a result, authors of today's textbooks on numerical mathematics limit themselves to write "it is very difficult to compute equations having multiple roots and close roots" or "it is necessary to use double or quadruple precision arithmetics for obtaining accurate solutions".

In this note, two approaches for the ill-conditioned polynomial equation are discussed. One is by using the interval arithmetic and the other is by the hybrid computation, i.e. a combination of symbolic and numerical computations. In the way of the hybrid computation, algorithms which are established in algebraic computations are extended to validated or approximate algorithms.

2. Rough sketches of an algorithm

If the polynomial P(x) has integer coefficients and integer roots , multiple roots for the P(x)=0 is easily separated. In this case, P(x) has not square-free decomposition and P(x) is divided by dP(x)/dx exactly. For s-fold multiple roots, P(x) is divided by $d^{(s)}P(x)/d^{(s)}x$ with residual=0. It follows that the greatest common divisor (GCD) of two polynomials P(x) and its s-times differentiation by x, $d^{(s)}P(x)/d^{(s)}x$, is not primitive. The GCD of two polynomials P_1 and P_2 is written as $GCD(P_1,P_2)$. It is usually obtained by the Euclidean algorithm and is shown as the **algorithm 1**. Here, the coefficients of polynomials are ommited for the simpilicity.

Algorithm 1 Euclidean algorithm for the GCD

```
input   polynomial P1(x), P2(x)
output  GCD(P1(x),P2(x))
1.   F <- P1 ,  G <- P2
2.   obtain Q and R satisfies
     F = QG + R
3.   if  R = 0  then
         GCD(P1,P2) = R
     else
         F <- G,  G <- R
4.   goto step 2.
```

P_1, P_2 and successive Rs construct the polynomial remainder sequence (PRS). The algorithm is rewritten as

Algorithm 1' Euclidean algorithm for the GCD

```
input   polynomial P1(x), P2(x), deg(P1)>deg(P2)
output  GCD(P1(x),P2(x))
calculate a PRS
   (P1, P2, ..., Pk≠0, Pk+1=0)
by the formula
   Pi-1 = QiPi + Pi+1 ,   i=2,...,k
if Pk+1 = 0  then
   GCD(P1,P2) = Pk
else
   repeat the process
```

where $deg(P_i)$ is the degree of the polynomial P_i. The PRS is computed by the elimination of leading terms(the highest degree term) of two polynomials. The GCD-computation is valid only for two integers or two polynomials whose coefficients and zeros are limited to integers or rational numbers. Here, we consider an extension of the GCD-computation to polynomials whose coefficients or roots are floating point numbers. We call the GCD in above case as an approximate-GCD. The computation of the PRS is terminated by a given parameter ε in the approximate-GCD calculation. The parameter ε corresponds to the accuracy of the result. The approximate-GCD is computed by the method with validation and by the combination of symbolic and numerical computations. The former is done by the interval arithmetic and the latter is realized on the hybrid computation system. If the

approximate-GCD is obtained, close roots of the original polynomial equation behave like multiple roots. The multiple root is easily separated from the equation. The ill-conditioned problem changes to two well-conditioned problems, one contains an approximate multiple root and the other does not contain it. Zeros of the equation which does not have approximate multiple roots are easily computed by any traditional methods for numerical computations. There remains a problem to obtain solutions of the equation having approximate multiple roots. The detailed discussion on the problem is in 3.

2.1 Approximate-GCD by the interval arithmetic

Two types of interval arithmetic have been discussed extensively. One is the rectangular interval arithmetic and the other is the circular one (Alefeld and Herzberger, 1983). The former is well known because it is easy to implement on computers. The advantage of the circular interval arithmetic is that it preserves some mathematical properties of the problem. The circular interval arithmetic consists of a center, A, and a radius, r(A) of a circle. A and r(A) correspond to a floating point number and an error, respectively. Because of the correspondence, here, the circular interval arithmetic is used to obtain the approximate-GCD.

The approximate-GCD of polynomials P_1 and P_2 with accuracy ε, $GCD(P_1,P_2,\varepsilon)$, is an natural extension of usual GCD computation. It is also obtained by the **Algorithm 1** , but the relational operator R=0 in the step 3 must be rewritten as $|r(R)| <= \varepsilon$. There arises a difficulty on a division by an interval number.

The fact that the denominator must not include zero in the interval arithmetic, requires modifications of the Algorithm 1. It is important especially for the case of irregular polynomials. The irregular polynomial is the polynomial whose leading coefficient(lc) is very small in the process to eliminate the leading term of polynomials. If the lc is normalized to unity, the PRS is obtained without difficulty. The Euclidean algorithm is modified as follows in the interval arithmetic.

Algorithm 2 Euclidean algorithm for the GCD(interval arithmetic)

input polynomial $P_1(x)$, $P_2(x)$, $\deg(P_1)>\deg(P_2)$, cutoff value
output $GCD(P_1(x),P_2(x))$ with accuracy
calculate a PRS
$(P_1,P_2,\ldots,P_k\neq 0$(with accuracy ε), $P_{k+1}=0$(with accuracy ε))
by the formula

$$P_{i-1} = Q_i P_i + P_{i+1}, \quad i=2,\ldots,k$$

where $lc(P_{i+1})$ = interval number corresponds to unity

if $|r(R)| <= \varepsilon$ then
$GCD(P_1,P_2,\varepsilon) = P_k$
else
repeat the process

Many studies have been done to implement the interval arithmetic especially for the rectangular interval arithmetic. In our computation, however, the circular interval arithmetic is adequate to decide whether the result contains zero or not. Here, the package to compute the circular interval arithmetic is made and implemented on personal computers. Some results are in 5.

2.2 Approximate-GCD by the hybrid computation

The exact GCD is easily obtained by the symbolic computation as shown above. We try to modify the Algorithm 1 for the case coefficients or zeros of the polynomial are floating point numbers. Computations are done by the combination of the symbolic computation and the numerical computation. Results by the symbolic computation are used in the numerical computation and

vise versa. We call the symbolic-numerical computation as the **hybrid computation.** In the hybrid computation, the GCD algorithm for the symbolic computation is modified. Numbers computed in the algorithm are not the interval but the integer, the rational number or the floating point number. The decision whether a number is zero or not is measured by considering accuracy. Except for the use of interval numbers, the strategy is similar to the method mentioned in 2.1 for the circular interval arithmetic. The normalization of the remainder R is also considered and is made some changes. Here, for the purpose of the normalization, the absolute value of the maximum magnitude coefficient of the polynomial P is defined and is written as **mmc**(P). The step 2 of the Euclidean algorithm becomes clearly as shown below.

Algorithm 3 Euclidean algorithm for the GCD (hybrid computation)

input polynomial $P_1(x)$, $P_2(x)$, $\deg(P_1)>\deg(P_2)$, cutoff value
output $GCD(P_1(x),P_2(x))$ with accuracy ε
calculate a PRS
$(P_1,P_2,\ldots,P_k\neq 0$(with accuracy ε),$P_{k+1}=0$(with accuracy ε))
by the formula

$$P_{i-1} = Q_i P_i + \max(1, \mathrm{mmc}(Q_i))P_{i+1}, \quad i=2,\ldots,k$$

if all coefficients of $P_{k+1} <= \varepsilon$ then
$GCD(P_1,P_2,\varepsilon) = P_k$
else
repeat the process

The detailed discussion on the algorithm and the approximate-GCD by the hybrid computation is in ref.2. Hybrid computation systems used here are briefly mentioned in 4.

3. Root-finding algorithm.

The root-finding process for well conditioned equations is performed easily. If ill-conditioned parts are separated out

from the given equation, the residual becomes the well conditioned equation. There is no trouble to obtain numerical solutions of the residual equation. Then, we must solve the extracted equation that contains multiple or close roots. If multiple roots are contained in the extracted equation, it is easy to find the solution by computing the GCD. There remains a problem on the equation having close multiple roots. The position of an approximate multiple root which is obtained by an approximate-GCD is on the center of close multiple roots (Sasaki and Noda). The result is important for the following discussion.

The root-finding algorithm for the equation having close roots is constructed both by the interval arithmetic and by the hybrid computation. In the interval arithmetic, the basis of the algorithm is the Krawczyk operator (Alefeld and Herzberger) and the Moore-Jones method on existence region of a solution. It requires the great number of operations to find out the uniquely existence region of a solution. Though it is possible to obtain satisfactory results, it takes too much CPU times. It seems new idea should be introduced to overcome the difficulty.

On the other hand, the root finding algorithm for the close root part in the hybrid computation is as follows. It gives satisfactory results both in the accuracy and in the CPU times. The notation

$$P^{(m)}(u) = d^m P(x)/dx^m \Big|_{x=u}$$

is used in the algorithm. The input of the algorithm is P(x) which is a regular polynomial having m close roots around x=u.

Outputs of the algorithm are m close roots $u_1,\ldots,u_m$ around x=u. Following three steps consist of the algorithm as follows:

Step 1: Construct the following equation in variable δ :

[C] $P^{(m)}(u)\,\delta^m/m! + \ldots + P^{(1)}(u)\,\delta/1! + P(u) = 0$

Step 2: Solve equation [C] w.r.t. by regularizing the l.h.s. and let the roots obtained be $\delta_1, \ldots, \delta_m$

Step 3: For each δ_i, i=1,...,m, solve P(x)=0 by Newton's method with initial approximation $x_0 = u+\delta_i$.

The regularization in the step 2 corresponds to the normalization in the **algorithm 3**. Even if the input polynomial is regular, an irregular polynomial appears, sometimes, in the elimination process of the algorithm. The regularization process is then important to complete the algorithm. It is done by an operation multiplying constant to the variable. Small coefficients in the polynomial are changed to coefficients having usual magnitude.

4. Hybrid computation systems.

The hybrid computation system is essential to obtain the approximate-GCD by the **Algorithm 3**. The second author of this note uses the hybrid computation system **GAL**(Sasaki et al.). The system GAL is written in LISP and is designed for big computers. On the other hand, the first author uses the portable hybrid computation system **SYNC**. It is written mainly in PROLOG and partially in C. As a hybrid computation system, it is a new comer but it is capable to compute hybrid computations on personal computers. Details on SYNC is written in another article(Noda and Iwashita). Here we will limit ourselves to give an outline of the **SYNC**. Special features of SYNC are as

follows:

a) A variable occurs in a mathematical expression is assigned to a prime number.

b) The data structure is adequate to a portable system.

c) The format of functions of SYNC's symbolic manipulation parts is similar to that of usual symbolic computation system.

d) SYNC has a powerful interface between symbolic computations and numerical computations.

e) Symbolic results in SYNC are easily translated to FORTRAN program and numerical FORTRAN results are used in SYNC.

f) SYNC has its own programming language. The programming language is similar to Pascal and is easy to program.

Among above features, e) is mainly described in the following. In SYNC, the result in symbolic computation (symbolic result) is used in the numerical FORTRAN computation. The numerical result is used in the next symbolic computation. The process is done automatically in SYNC. The process is divided into five stages as follows: 1) the symbolic computation is done, 2) a program written in an intermediate language receives the symbolic result as its input, 3) the program is translated to a FORTRAN source program, 4) the FORTRAN compiler runs and the numerical result is generated and 5) the numerical result is used in the next symbolic computation. Stages 2) to 4) are managed by the SYNC's predicate fortran. The intermediate language in the stage 2) is a small extension to usual FORTRAN. It has a new statement JOINT which is similar to the block COMMON statement. JOINT statement has two blocks. One is called IN and the other is OUT.

In IN block, symbolic results are stored and in OUT block, numerical outputs are stored. The intermediate language whose file-attribute is **syn** is translated usual FORTRAN in the stage 3). An example of stages are shown as follows:

```
<Q> f:=x^4-10.4*x^3-70.96*x^2+29.6*x-3.
<A> x ^ 4 - 10.4 * x ^ 3 - 70.96 * x ^ 2 + 29.6 * x - 3.
<Q> df:=dif(f,x).
<A> 4 * x ^ 3 - 31.2 * x ^ 2 - 141.92 * x + 29.6.
<Q> x0:=20.
<A> 20.
<Q> eps:=1.0e-5.
<A> 0.00001.
<Q> fortran newton(f,df,x0,eps,out).
      *** compiler & linker messages ***
 approximate sol. =  14.999999965778487
<Q> g:=x-out.
<A> x - 14.99999966.
<Q> poldiv(f,g).
    residual = -0.00149918
    quotient = x^3 + 4.59999966*x^2 -1.96000671*x + 0.19990006
<A> done.
```

In above, a line with <Q> accepts user's input statement and a line with <A> returns results of evaluation by SYNC. An input statement must terminate a symbol ".". in the first input line, a polynomial is defined. The derivative of it with the variable x is computed symbolically in the next <Q> line. The initial value and the stopping criterion of the Newton iteration are defined in the following two inputs. User must prepare a program for the hybrid Newton's method in the intermediate language and store it in the file with the name "newton.hyb". The program "newton.hyb" is the same as usual FORTRAN program except for one statement. In the program, a statement

```
JOINT /IN/ F,DF,x0,eps /OUT/ y
```

must be added. The order of these parameters and the name of the program correspond to arguments of the predicate **fortran**. The program in the intermediate language, "newton.hyb", is

automatically translated to the program in usual FORTRAN, "newton.f77", by the predicate **fortran** in SYNC. The FORTRAN program generated here is stored in file and is accepted by FORTRAN compiler. Results of the numerical computation is put on /OUT/ block. As shown in the figure, the numerical result is easily used in the next symbolic computation. The user defined predicate "poldiv" in the final input line divide a polynomial f by the other g. The residual and the quotient are shown symbolically.

Some studies have been proposed on the connection of symbolic computations and numerical computations by using FORTRAN. In famous symbolic computation systems REDUCE and MACSYMA, the FORTRAN source program is also generated automatically. In these systems, however, the symbolic computation must be terminated and the FORTRAN processor must be started as a different computation. Symbolic and Numerical results are, then, exchanged through files and two systems run independently.

5. Examples for the approximate-GCD.

5.1 Multiple root(double root)

We consider the polynomial treated in 4,

$$P(x) = x^4 - 10.4\,x^3 - 70.96\,x^2 + 29.6\,x - 3\ .$$

The equation $P(x) = 0$ has roots $x = -5$, 15 and double root at $x=0.2$. It is an ill-conditioned polynomial equation because of its double roots. Two methods described in 2 are applied to the equation. In the circular interval number with the value A and the radius r(A) is represented as $< A, r(A) >$. We show the PRS generated by the **algorithm 2** as follows.

F_1 = < 1.0 , 2.2E-16> x^4 + <-10.4, 1.8E-15> x^3
+ <-70.96, 1.4E-14> x^2 + < 29.6, 3.6E-15> x
+ < -3.0 , 4.4E-16>
F_2 = < 4.0E 00, 1.8E-15> x^3 + <-3.1E 01, 8.9E-15> x^2
+ <-1.4E 02, 5.7E-04> x + < 3.0E 01, 7.1E-15>
F_3 = <-8.9E 02, 3.8E-12> x^2 + <-1.1E 03, 9.3E-12> x
+ < 2.6E 02, 1.8E-12>
F_4 = <-4.7E 06, 1.1E-07> x + < 9.5E 05, 2.2E-08>
F_5 = < 1.3E-06, 7.8E-04> ∋ 0

Starting r(A)s in F_1 are automatically computed according to the floating point format in the computation. Above computations are performed on the IBM-PC with the Intel's 8087 floating point co-processor. The IEEE format is used in the computation. Then, the GCD of P(x) and its derivative is f_4. It means the equation P(x)=0 has the double root. If we put the lc of the GCD to unity, we obtain the GCD as

GCD(P(x), dP(x)/dx) = x - < 0.2, 4.7E-10 >.

On the other hand, the hybrid computation gives the following PRS by the algorithm 3.

F_1 = x^4 - 10.4 x^3 - 70.96 x^2 + 29.6 x - 3
F_2 = - 4 x^3 - 31.2 x^2 - 141.92 x + 29.6
F_3 = - 85.78 x^2 - 107.76614385 x + 24.98461538
F_4 = - 0.46563934 x + 0.09312787
F_5 = 2.77555756E-17

Then, we obtain the GCD = x - 0.2, because the F_5 is approximately zero. Above results for the double root show that two algorithms are also valid for the ill-conditioned polynomial equation with close roots.

5.2 Close roots

We consider an equation with close root such as

P(x) = (x + 5)(x - 15)(x - 0.2)(x - (0.2 - η)).

The algorithm 2 generates the PRS for several η. Final F_5s of the PRS are shown in the Table 1. When $\eta > 10^{-6}$, the absolute

value of A is greater than r(A) and the obtained interval does not contain zero. On the other hand, $\eta \leq 10^{-7}$, the interval contains zero and F_4 should be taken as the approximate-GCD. If we make r(A) large, then F_5 with greater η will be zero in the interval arithmetic. An example of the PRS for $\eta = 10^{-5}$ and a starting $r(A) = 10^{-10}$ is computed as

$$
\begin{aligned}
F_1 &= \langle\ 1.0\ ,\ 1.0E\text{-}10\rangle\, x^4 + \langle -10.4\ ,\ 1.0E\text{-}10\rangle\, x^3 \\
&\quad + \langle -70.96\ ,\ 1.0E\text{-}10\rangle\, x^2 + \langle\ 29.6\ ,\ 1.0E\text{-}10\rangle\, x \\
&\quad + \langle\ -3.0\ ,\ 1.0E\text{-}10\rangle \\
F_2 &= \langle\ 4.0E\ 00,\ 4.0E\text{-}10\rangle\, x^3 + \langle -3.1E\ 01,\ 3.0E\text{-}10\rangle\, x^2 \\
&\quad + \langle -1.4E\ 02,\ 2.0E\text{-}10\rangle\, x_2 + \langle\ 3.0E\ 01,\ 1.0E\text{-}10\rangle \\
F_3 &= \langle -8.9E\ 02,\ 4.8E\text{-}07\rangle\, x^2 + \langle -1.1E\ 03,\ 1.2E\text{-}06\rangle\, x \\
&\quad + \langle\ 2.6E\ 02,\ 2.5E\text{-}07\rangle \\
F_4 &= \langle -4.7E\ 06,\ 1.3E\text{-}02\rangle\, x + \langle\ 9.5E\ 05,\ 2.6E\text{-}03\rangle \\
F_5 &= \langle\ 1.4E\ 00,\ 9.3E\ 00\rangle\ \ 0
\end{aligned}
$$

Then the approximate-GCD is obtained from F_4 as

approximate-GCD = < 1.0, 5.3E-09 > x + <-0.200005, 1.1E-09 >.

It is possible to extract the approximate GCD from a given ill-conditioned polynomial equation with close roots. The starting r(A) behaves like a cutoff value ε in the algorithm 3.

The hybrid computation by using algorithm 3 is also easy to extract the approximate-GCD. The computation is performed with no trouble. The normalization process is important in whole computation. Results for any ηs are shown in Table 2.

η	F_5
10^{-1}	< 1.4E 08, 7.1E-04 >
10^{-2}	< 1.4E 06, 5.9E-04 >
10^{-3}	< 1.4E 04, 5.8E-04 >
10^{-4}	< 1.4E 02, 5.8E-04 >
10^{-5}	< 1.4E 00, 5.8E-04 >
10^{-6}	< 1.4E-02, 5.8E-04 >
10^{-7}	< 1.4E-04, 5.8E-04 >
10^{-8}	< 0.0E 00, 5.8E-04 >

Table 1

η	F_5	F_4
10^{-1}	-0.0094543	x-0.23943771
10^{-2}	-0.0001167	x-0.20487204
10^{-3}	-0.0000012	x-0.20049869
10^{-4}	-1.0E-09	x-0.20004999
10^{-5}	-1.197E-10	x-0.20000499
10^{-6}	0.0	x-0.20000000

Table 2

6. Conclusions.

A method to solve the ill-conditioned polynomial equations are considered. Ill-conditioned parts, multiple or close roots, are extracted from the given equation. The method of the extraction is based on the Euclidean algorithm that is defined as the 'exact'-algebraic method. The algorithm is extended to be able to treat floating point coefficients and zeros. The first extension is done by using the interval arithmetic and the second is by the hybrid computation. Examples by both methods are shown and results are satisfiable.

Extracted parts which contain close roots are also computed very carefully. In the hybrid computation, the Taylor series expansion is effectively used. In the interval arithmetic, accurate root finding methods are based on a modified Newton's method but it takes too many CPU times. An efficient method for the root finding should be necessary.

The hybrid computation discussed here should be extended to new and ultimate hybrid computation. In the 'ultimate' computation, it is expected the combination of symbolic and validated numerical computations . To better use of the 'ultimate'-hybrid computation, the hybrid computation system which allows both the validated numerical computation and the symbolic computation seems to be important. The portable hybrid computation system SYNC and the validated PASCAL-SC (Rall, 1985) will be candidates for the system.

Computations reported here were done by M .Ochi, Ehime Univ., (hybrid computation) and by Y. Nishio, Ehime Univ., (circular

interval arithmetic). H. Asagawa, Ehime Univ., read the manuscript carefully.

References

Alefeld,G. and Herzberger,J. : 'Introduction to Interval Computation', Academic Press, 1983.

Bareiss,E.H.: The numerical solution of polynomial equations and the resultant procedure , in 'Mathematical Method for Digital Computers Vol.2', John Wiley, 1967.

Moore,R.E. and Jones,S.T. : Self starting regions for iterative methods, SIAM J. Numer. Anal., 14, 1977, pp.1051-1065.

Noda,M.T. and Iwashita,H. : Portable hybrid computation system SYNC, J. Inf. Proc. Japan, to be submitted(in Japanese).

Rall,L.B. : An introduction to the scientific computing language Pascal-SC, trans. Second Conf. on Applied Math. and Computing, U.S. Army Research Office, Research Triangle Park, NC, 1985, pp.117-148.

Sasaki,T., Fukui,Y., Suzuki,M. and Sato,M.(1988): Proposal of a scheme for linking different computer languages, Jour. Inf. Proc., submitted.

Sasaki,T. and Noda,M.T., Approximate square-free decomposition and root-finding of ill-conditioned algebraic equations", Jour. Inf. Proc., submitted.