Proceedings of the Risa Consortium
1997 pp. 131– pp. 139

2 変数代数方程式の実特異零点を含む区間の決定

齋藤 友克* 野田 松太郎†
上智大学理工学部 愛媛大学工学部

(RECEIVED 1997/7/15 REVISED 1997/10/20)

1. はじめに

2 変数代数方程式の実孤立零点の位置を代数的に求め、陰関数描画に適用する問題を考える。1 変数の場合は、零点の位置を Strum の方法を用いて代数的に求め、かつ描画アルゴリズムとする方法はすでに [3, 4] などで示されているが、2 変数の場合は同様の手法はいまだ知られていない。もちろん、代数方程式の解を数値計算によって求め、関数描画をしようという方法は一般的であるが、2 変数の場合は安定に数値解を得る方法はあるとはいえない。多くの数値的方法は Newton 法を基礎としているが、特異点計算のような場合は悪条件になり、計算速度、解の精度の両面で問題がある。

これに対して代数的解法は、計算速度の面では数値的方法に劣るものの、結果の信頼性においては優れており、本論で取り扱うような問題に対してはむしろ推奨されるべきものであると思われる。代数的解法という言葉から連想されるように、ここで与えられた 2 変数代数方程式に対して、

1. Gröbner 基底計算によって最小多項式を求める
2. 最小多項式より、Sturm 列を計算し、解が含まれる領域を決定する
3. この領域内には解はただ一つ含まれるものとする

というものである。Sturm 列の計算によって、解の存在する区間を判定する試みは [1, 2] 等においても取り扱われている。

*saito@mm.sophia.ac.jp
†noda@cs.ehime-u.ac.jp

さらに本論では、目的の実孤立零点を直接求めるのではなく、代数方程式の実特異零点を代数的に求める。与えられた方程式を $f(x,y)=0$ とすると、この問題は、次のような代数方程式系

$$\begin{cases} f(x,y)=0 \\ \dfrac{\partial f}{\partial x}=0 \\ \dfrac{\partial f}{\partial y}=0 \end{cases}$$

の解を求めることになる。

2. 目的と記号の準備

上で述べたように、実特異零点を求める問題は、2 変数代数方程式の実特異零点が与えられた矩形領域の内部に存在するか否かの判定をすることと同値なので、本論の目的は、以下の 2 つの条件を満足する代数的アルゴリズムを構築することにある。

1. **解の分離**　与えられた領域 D の中に個々の特異点に対して

$$\begin{cases} C=\{(x,y)|\ \ a\le x\le b,\ \ c\le y\le d\} \\ f(C) \text{ は } C \text{ 内に解を持つ} \\ \dfrac{\partial f}{\partial x} \text{は } C \text{ 内に解を持つ} \\ \dfrac{\partial f}{\partial y} \text{は } C \text{ 内に解を持つ} \end{cases}$$

となる領域 C の a,b,c,d を決定する。

2. **完全性**　$f=0$ が 定義域において特異点を持てば必ずその特異点をただし 1 個含む領域を決定する。

なお、本論で取り扱う代数方程式は有理係数とする。これは、本来問題が誤差の概念とは相容れない特異点を対象としている以上必然的であり、このために代数的解法を構築したともいえる。同様の議論は有理数体の有限次拡大体上においても成り立つことを付け加えておく。また、方程式は無平方であると仮定する。これは、無平方でない方程式は容易に無平方化することができるし、両者は同一実特異点を有するからである。

本論で使用する記号として、与えられた方程式を $f=0$ としその変数を x,y とあらわす。$f=0$ の実特異零点を $P_{1,1}=(s_1,t_1),\cdots,P_{r,r}=(s_r,t_r)$ とあらわす。与えられた方程式から生成されるイデアル $<f,\frac{\partial f}{\partial x},\frac{\partial f}{\partial x}>$ を$\mathcal{I}$とする。

3. アルゴリズム

3.1. 基本方針

解を含む領域を決定する方法は 1. でも述べたように、与えられた方程式系 (イデアル) の Gröbner 基底を全次数逆辞書式順序で求める。この基底を利用して x に関する最小多項式を基底変換により求める。ここで零次元の Gröbner 基底の最小多項式とは、新たに変数とそれを表す多項式 $t = f(x, y)$ を追加した方程式系における新たな変数の最小次数の底である。

まず始めに $t = x$ とおいた t の最小多項式にたいして Sturm 列を利用し、解区間求める、この区間は x に対する解区間である。ついで y に関しても同様に最小多項式より Sturm 列から y の解区間を求める。この区間により決定された矩形領域が解を含む目的の領域の候補である。最後に得られた個々の矩形領域が解を含むか否かの判定を行う。この判定の方法として、一般の位置にある直線を求め、再び Sturm 列を用いる。なお、一般の位置にある直線とは、その直線に対し方程式系の解の重複度が 1 である直線である。

3.2. 領域の候補の決定

イデアル $\mathcal{I}$ の x に関する最小多項式 ϕ_x の実数解の集合 $\{s'_1, \cdots, s'_u\}$ は、最小多項式の定義より、もとの方程式系の実特異点の x 座標からなる集合 $\{s_1, \cdots, s_r\}$ と一致する。よって ϕ_x に対し Sturm の定理を適用すれば、各々の s'_i に対し s'_i を 1 つだけ含む x の区間 $I_i = [a_i, b_i]$ を得ることができる。

同様にして変数 y に対する最小多項式から解を含む区間 $J_1 = [c_1, d_1], \cdots, J_v = [c_v, d_v]$ を求める。

これにより実特異点を含む可能性がある領域 $D_{l,m} = I_l \times J_m$ $(l = 1, \cdots, u \quad m = 1, \cdots, v)$ が全て求まったことになる。

3.2.1. 一般の位置にある直線の決定

一般の位置にある直線の決定方式としては、次の 2 つの方式がある。

1. **分割方式** 領域 $D_{l,m}$ ごとに適切な直線を決定する。この方式は直線を決定する条件が少ないためアルゴリズム的には軽便であるが各々の領域に対して毎回 strum 列を求める必要がある。
2. **一括方式** 全ての解に対して 1 本の直線を決定する。この方式は、直線の決定は困難が伴うが後の判定に利用する最小多項式を一度求めればよい。しかし、ほとんど全ての直線が一般の位置にあることを利用すれば、まづ任意の直線に対し解の判定を行いう。まく行かない場合直線を取りなおす手法がより現実的である。この場合一般の位置にある

直線を発見するまで試行する必要がある。この試行は解の個数が有限個であることより必ず有限回で終了する。

どの方式も必要に応じて x y の区間を縮小する。

3.2.2. 個々の候補ごとに決定する場合

判定をおこなう領域を $D_{l,m}$ とする。このとき領域の端点は (a_l, c_m), (b_l, c_m), (a_l, d_m), (b_l, d_m) である。このとき次の直線を考える。

$$y = \frac{1}{a_l - b_l}\{(d_{m-1} - c_m)x + a_l c_m - b_l d_{m-1}\}$$

この直線は、(b_l, c_m) と (a_l, d_{m-1}) を結ぶ直線である。このとき x に a_{l+1} を代入した値が d_v よりも小さい場合区間を縮小し値が d_v よりも大きくなるようにする。

また、

$$y = \frac{1}{a_l - b_l}\{(d_m - c_{m+1})x + a_l c_{m+1} - b_l d_m\}$$

に関しても x に b_{l-1} を代入した値が c_1 が大きければ区間を縮小し値が c_1 よりも小さくなるようにする。

以上の処理により

$$\begin{cases} y \geq \frac{1}{a_l - b_l}\{(d_{m-1} - c_m)x + a_l c_m - b_l d_{m-1}\} \\ y \leq \frac{1}{a_l - b_l}\{(d_m - c_{m+1})x + a_l c_{m+1} - b_l d_m\} \end{cases}$$

で定められた領域は $D_{l,m}$ 以外の $D_{i,j}$ と共通部分は存在しない。よって、この領域に対する一般の位置になる直線は

$$y = \frac{1}{a_l - b_l}x$$

となる。

イデアル $\mathcal{I}$の $z = (a_l - b_l)y - x$ に関する最小多項式 $\phi(z)$ を求める。得られた最小多項式 $\phi(z)$ に対し Sturm 列を求め $(a_l - bl)d_m - a_l$ と $(a_l - b_l)c_m - b_l$ の値を代入し差が 0 でなければ $D_{l,m}$ は実特異点を含む。

3.2.3. 全ての候補に対する直線の決定

まず原点を通る直線 $z = y - \alpha x$ に対し最小多項式を求める。この多項式が一般の位置にあれば以下のようにする。

$D_{l,m}$ の隅 4 点 (a_l, c_m), (b_l, c_m), (a_l, d_m), (b_l, d_m) を $z = y - \alpha x$ に代入し最大値を $U_{l,m}$ 最小値を $L_{l,m}$ とする。このとき全ての区間 $[U_{l,m}, L_{l,m}]$ が共通部分を持たなければこの直線を一般の位置の直線として使う。もし共通部分が存在すれば x と y の区間の幅を短くして繰り返す。このことにより直線上に uv 個の共通部分を持たない区間が決定される。

この直線に関する最小多項式より Sturm 列を求め決定した区間に対し解が含むか含まないかによって $D_{l,m}$ が実特異点を含むか否かが判定出来る。

4. Risa/Asir プログラム

上で述べたアルゴリズムを実際に Risa/Asir にインプリメントする。ここでは、解の存在の判定と存在する場合の領域を求めるために、関数 icheck を以下のように作成した。

```
def icheck(F, In, Ga, Di) {
/*****************************************************************
*   F:  テストする多項式
*   In: 調べる領域の大きさ リスト [[x の領域],[y の領域]]
*   Ga: 判定を停止する分割の大きさの最小値
*   Di: 判定をおこなう直線の方程式の傾き y = Di x に対して判定する
*   戻り値は実特異点を含む領域を表す x,y の数値のリスト
*****************************************************************/
   V = vars(F) $
   Fx = diff(F, V[0]) $
   Fy = diff(F, V[1]) $
   GB = hgr([F, Fx, Fy], V, 0) $
   P = ptozp(V[1] - Di * V[0]) $
   U1 = uc() $     U2 = uc() $ U3 = uc() $
   X = minipoly(GB, V, 0, V[0], U1) $
   Y = minipoly(GB, V, 0, V[1], U2) $
   Ix = chk(X, In, Ga) $
   Lx = length(Ix) $
   Iy = chk(Y, In, Ga) $
   Ly = length(Iy) $
   Rc = interchk(Ix, Iy, Lx, Ly, P, V) $
   while (Rc == -1) {
      Ga = Ga / 2 $
      Ix = chk(X, In, Ga) $
      Lx = length(Ix) $
      Iy = chk(Y, In, Ga) $
      Ly = length(Iy) $
      Rc = interchk(Ix, Iy, Lx, Ly, P, V) $ }
   Z = minipoly(GB, V, 0, P, U3) $
   Sturm = sturm(Z) $
   Size = size(Sturm) $
   Solv =[] $
   for (I = 0; I < Lx; I++) {
      Wx =[Ix[I][0], Ix[I][1]] $
      for (J = 0; J < Ly; J++) {
         Wy = [Iy[J][0], Iy[J][1]] $
         N = nchk(Sturm, Size, U3, Wx, Wy, P, V) $
         if (N != 0) Solv = append(Solv,[[Wx, Wy]]) $ } }
   return Solv$ }

def interchk(Ix, Iy, Lx, Ly, P, V) {
/*****************************************************************
*   Ix : x 軸に対し Sturm により解が分離された領域のリスト
```

```
*   Iy : y 軸に対し Sturm により解が分離された領域のリスト
*   Lx : Ix のリストの長さ
*   Ly : Iy のリストの長さ
*   P  : 与えられた多項式
*   V  : 直線の方程式
*   与えれた領域は Ix[i] x Iy[j] となるこの時の V に対する像が重複
*   しているかどうかの判定を行う関数。重複していれば戻り値は -1
*************************************************************/
   Int =[] $
   for (I = 0; I < Lx; I++) {
      for (J = 0; J < Ly; J++) {
         Iw = ichk([Ix[I][0], Ix[I][1]],[Iy[J][0], Iy[J][1]], P, V) $
         Int = append(Int,[[Iw[0], Iw[3]]]) $ } }
   L = Lx * Ly $
   for (I = 0; I < L; I++) {
      Dl = Int[I][0] $
      Du = Int[I][1] $
      if (Dl > Du) {
         Tm = Du$
         Du = Dl$
         Dl = Tm$ }
      for (J = I + 1; J < L; J++) {
         W =[Dl, Du, Int[J][0], Int[J][1]] $
         S = qsort(W) $
         if ((S[0] == Dl) && (S[1] != Du)) return -1 $
         if ((S[2] == Dl) && (S[3] != Du)) return -1 $ } } }

def nchk(Sturm, Size, Var, X, Y, P, V) {
/*********************************************************************
*  与えられた Sturm 列のリストより区間の中にある解の個数を計算する
*********************************************************************/
   S = ichk(X, Y, P, V) $
   E1 = evsturm(Sturm, Size, S[0], Var) $
   E2 = evsturm(Sturm, Size, S[3], Var) $
   return abs(E1 - E2) $ }

def chk(F, In, Ga) {
/*********************************************************************
*   再帰的に解を含む区間を求めるための始めの関数
*********************************************************************/
   Var = var(F) $
   Sturm = sturm(F) $
   Size = size(Sturm) $
   E0 = evsturm(Sturm, Size, In[0], Var) $
   E1 = evsturm(Sturm, Size, In[1], Var) $
   E =[E0, E1] $
   Sol = rchk(Sturm, Size, Var, In, Ga, E) $
   return Sol$ }

def rchk(Sturm, Size, V, In, Ga, Ei) {
/*********************************************************************
*   区間を分割して解を分離する関数の実体
*********************************************************************/
   Out =[] $
   Mp = (In[0] + In[1]) / 2 $
```

```
   Mv = evsturm(Sturm, Size, Mp, V) $
   if (Mp - In[0] <= Ga) {
      if (Ei[0] != Mv) Out = append(Out,[[In[0], Mp]]) $
      if (Mv != Ei[1]) Out = append(Out,[[Mp, In[1]]]) $
   } else {
      if (Ei[0] != Mv) {
         O = rchk(Sturm, Size, V,[In[0], Mp], Ga,[Ei[0], Mv]) $
         if (O !=[]) Out = append(Out, O) $
      }
      if (Mv != Ei[1]) {
         O = rchk(Sturm, Size, V,[Mp, In[1]], Ga,[Mv, Ei[1]]) $
         if (O !=[]) Out = append(Out, O) $ } }
   return Out$ }

def evsturm(Sturm, Size, Val, Var) {
/*****************************************************************
*   与えられた Sturm 列の Val における値を求める
*****************************************************************/
   Scount = 0 $
   if (subst(Sturm[0], Var, Val) > 0) Cs = 1 $ else Cs = -1 $
   for (I = 1; I < Size[0]; I++) {
      if (subst(Sturm[I], Var, Val) > 0) {
         if (Cs == -1) {
            Scount++ $
            Cs = 1 $ }
      } else {
         if (Cs == 1) {
            Scount++ $
            Cs = -1 $  } } }
   return Scount$ }

def ichk(X, Y, P, V) {
/*****************************************************************
*   与えれた領域の角の点を V に射影した像の位置をソートしたリスト
*   として返す
*****************************************************************/
   A1 = subst(P, V[0], X[0], V[1], Y[0])$
   A2 = subst(P, V[0], X[0], V[1], Y[1])$
   A3 = subst(P, V[0], X[1], V[1], Y[0])$
   A4 = subst(P, V[0], X[1], V[1], Y[1])$
   S = qsort([A1, A2, A3, A4])$
   return S$ }
```

4.1. 実行例

このアルゴリズムの実行例を示す。対象とする関数は

$$\begin{aligned}\mathrm{Heart}(x,y) = &\frac{93392896}{15625}x^6 + (\frac{94359552}{625}y^2 + \frac{91521024}{625}y - \frac{249088}{125})x^4 \\ &+ (\frac{1032192}{25}y^4 - 36864y^3 - \frac{7732224}{25}y^2 - 207360y + \frac{770048}{25})x^2 \\ &+ 65536y^6 + 49152y^5 - 135168y^4 - 72704y^3 \\ &+ 101376y^2 + 27648y - 27648\end{aligned}$$

を考える。

```
[100] load("gr")$
[101] In=[-201/100,203/101]$
[102] Ga=1/107$
[103] Di=1/2$
[104] icheck(Heart,In,Ga,Di);
[[[-2233119/1292800,-355675/206848],[-112069/206848,-690281/1292800]],
[[-121819/323200,-1908503/5171200],[-5724997/5171200,-1421099/1292800]],
[[-1/20200,8069/1034240],[-40603/40400,-5156583/5171200]],
[[-1/20200,8069/1034240],[3856839/5171200,24359/32320]],
[[18891/51200,121787/323200],[-5724997/5171200,-1421099/1292800]],
[[8891363/5171200,2232991/1292800],[-112069/206848,-690281/1292800]]]
0.4767sec + gc : 0.1958sec
```

となり Heart$(x, y) = 0$ の実特異点を求めることができる。このとき使用した計算機は Pentum Pro 200Mhz である。

5. むすび

多変数代数方程式の特異点は、数学的にその方程式の性質を知るために必要なものである。また、特異点を正確に求めることは、応用数学の多年の夢であり現在まで満足いく解法はなかったといえる。特異点近傍からの数値計算に頼る程度であったが、この様な逆に問題は数値計算技法が最も苦手とするものでもある。しかし、本論で述べた代数的アルゴリズムを用いることによって、特異点の個数と特異点を含む領域を必要な精度で求めることが可能になった。

もちろん、このアルゴリズムは万能ではない。数値計算と比較にならないほど、計算速度は遅く、記憶容量を消費する。また、当然ではあるが、代数方程式の次数が高くなる場合や係数が非常に長桁になる場合には、現在のままでは目的を達することが出来ない場合が多くなるであろう。特に、大規模な方程式を扱う場合に、Gröbner 基底を求める計算による制約が大きいことである。今後早急に検討すべき課題は、本論で導入した一般の位置にある直線の決定するアルゴリズムの効率化と、上で述べた Gröbner 基底の計算法の改良である。

参 考 文 献

[1] Davenport,J.H.,Siret,Y.,Tournier,E. :*Computer Algebra*, Academic Press, 1988

[2] Ochi,M.,Noda,M. T. and Sasaki,T. :*Approximate Greatest Common Divisor of Multivariate Polynomials and Its Application to Ill-Conditioned Systems of Algebraic Equations*, *J.Inf.Proc.*, Vol. 14 (1991), pp. 292–300.

[3] Saito,T.:*An extension of Sturm's theorem to two dimensions*, *Proc. Japan Academy*, 73, Ser. A (1997), pp. 18–19

[4] Takeshima,T.,Noro,M.Saito,T. :図形描画装置及びその方法, 特許出願番号 *H06-048717*, 1994