91-401

сообщения
объединенного
института
ядерных
исследований
дубна

E5-91-401

V.P.Gerdt, P.Tiller

A REDUCE PROGRAM FOR SYMBOLIC COMPUTATION OF PUISEUX EXPANSIONS

1991

1. Introduction

The Puiseux expansion of a function $x(\lambda)$, defined as a solution of the given algebraic equation

$$F(x, \lambda) = 0$$

, with the small parameter λ is the following series

$$x(\lambda) = x_{\varepsilon_1}\lambda^{\varepsilon_1} + x_{\varepsilon_2}\lambda^{\varepsilon_2} + x_{\varepsilon_3}\lambda^{\varepsilon_3} + \cdots, \tag{1}$$

where $\varepsilon_1 < \varepsilon_2 < \varepsilon_3 < \ldots$ are rational numbers with the same denominator. The expansion (1) is a convergent series in the neighborhood of the point $\lambda = 0$ with this point deleted.

The theory of Puiseux expansions is a major tool in algebraic geometry where they act as Laurent expansions in ordinary function theory. In addition to it the Puiseux expansions are a powerful tool for solving many computational problems. Among them are:

- Calculation of the genus of the Riemann surface, construction of differentials of the first kind, or general construction of meromorphic functions on the Riemann surface (Coates' algorithm [1, 2]).
- Determination and analysis of branching the solutions of nonlinear equations [3, 4].
- Symbolic integration of algebraic functions [2, 5].
- Finding of the complete topological invariants of algebraic curves [6].
- Fast numeric evaluation of the solutions of linear differential equations containing algebraic functions [7].
- Analysis of the elliptic curves determining the meromorphic solutions of linear differential equations of Halphen type [8].

The construction of the Puiseux expansions (1) is a very tedious procedure. Therefore a computer implementation is very useful for applications. A number of computer programs have been created recently for computation of Puiseux expansions. They are written in different languages: `Reduce` [9, 10], `Scratchpad II` [11], `Lisp` [12], `Pascal` [13]. All these programs except [12] are based on the classical algorithm of Newton and

Puiseux [15]. The program [9] is written in Rlisp, i.e. in symbolic mode of the Reduce language, but has not been widely distributed. Another previous Reduce program [10] was written in algebraic mode of the Reduce language and by this reason is not very efficient.

In this paper our new implementation of the classical Newton-Puiseux method (Sect.2) in the symbolic (Rlisp) mode of Reduce 3.3 [14] is described (Sect.3). The program is running under MS-DOS on 286 and 386-based IBM PCs. Its efficiency is demonstrated by different examples (Sect.4).

2. Description of Algorithm

In this section we give a brief description of the method for constructing the Puiseux expansions of the algebraic function $x(\lambda)$ defined by the equation $F(x,\lambda)=0$, where $F(x,y) \in Q[x,y]$, i.e. F is a polynomial in x, y with rational coefficients, vanishing at point (0.0).

We use the classical Newton and Puiseux algorithm, based on the well-known Newton polygon method [15, 4] for successive construction of the terms of the Puiseux expansions (1). It should be noted that this method is a very useful tool for computer algebraic analysis and solving of linear algebraic equations at a singular point [16, 17].

Using the Newton and Puiseux method one can compute all possible values of x_ε and ε. Let the equation $F(x,\lambda)=0$ be given in the form

$$\sum_{s=0}^{n} f_s(\lambda)x^s = 0. \qquad (2)$$

We assume that in neighborhood of the point $\lambda = 0$ the coefficients $f_s(\lambda)$ are presented by the convergent series

$$f_s(\lambda) = f_{0s}\lambda^{\varrho_s} + \sum_{r=1}^{\infty} f_{rs}\lambda^{\varrho_s+\frac{r}{p}} \qquad (3)$$

where ϱ_s is a rational number. After the substitution of the first terms of (3) and (1) into (2) in the limit $\lambda \to 0$ we have the following equation

$$\sum_{s=0}^{n} f_{0s}\lambda^{\varrho_s+s\varepsilon}x_\varepsilon^s = 0. \qquad (4)$$

Now we may choose the value of ε in such a way as to have at least two equal exponents $(\varrho_j + j\varepsilon, \varrho_l + l\varepsilon)$ with the others satisfying the inequality $\varrho_s + s\varepsilon \geq s_j + j\varepsilon$. Hence from eq. (4) we obtain

$$\lambda^{\varrho_j+j\varepsilon}\sum_{s=0}^{n} f_{0s}x_\varepsilon^s = 0, \qquad (5)$$

where the summation is over all values s such that $\varrho_s + s\varepsilon = \varrho_j + j\varepsilon$. There is only one possibility for vanishing the left hand side of (5) if $\lambda \neq 0$. It means that the condition

$$\sum_{s=0}^{n} f_{0s}x_\varepsilon^s = 0 \qquad (6)$$

must be satisfied.

For the computation of the possible values of ε we use the Newton polygon method. It may be split into the following successive steps:

1. Plotting the points (s, ϱ_s), $s = 1, \ldots, n$ in a cartesian coordinate system.
2. For each point (k, ϱ_k), $k = 1, \ldots, n$ computing the value of the variable
$$\varepsilon_k = \frac{\varrho_0 - \varrho_k}{k},$$
that is the value of $\tan^{-1}\varphi$, where φ is the angle between the negative direction of the ϱ-axis and the line connecting two points $(0, \varrho_0)$ and (k, ϱ_k).
3. Finding the point (l, ϱ_l) which satisfies the conditions
   - $\varepsilon_l \geq \varepsilon_k$ for all $k = 1, \ldots, n$,
   - if $\varepsilon_l = \varepsilon_k$ then $l > k$.
4. Omitting all the points (k, ϱ_k) with $k < l$ and repeating steps 2-4 until the last point (n, ϱ_n).

After this algorithm we get a set of points called Newton set. If we connect the neighboring points by straight lines we obtain the Newton polygon. All the points (s, ϱ_s), $s = 0, \ldots, n$ lie either on or above this polygon. The possible values of ε are determined by the slopes of the segments of the Newton polygon. For a given value of ε x_ε must satisfy (6).

Therefore, to obtain the coefficient x_ε we have to solve eq.(6), which is generally a polynomial equation in one variable. Solving (6) is the most nontrivial step of the whole procedure. In many cases, however, eq.(6) is quite simple (see, for instance, the examples of Sect.4) to be solvable by a standard built-in Reduce package Solve [14] used in the given program.

As a result we obtain the first Puiseux terms in (1)

$$x = x_\varepsilon\lambda^\varepsilon + O(\lambda^\varepsilon) \qquad (7)$$

for all possible values of ε, i.e. branches. Then one can substitute (7) in eq. (2) and, having denoted the second term $O(\lambda^\varepsilon)$ by x, repeat the above procedure. This gives the second Puiseux terms. In such a way one can compute the explicit form of the first k terms of the Puiseux expansions $x(\lambda) = x_{\varepsilon_1}\lambda^{\varepsilon_1} + \cdots + x_{\varepsilon_k}\lambda^{\varepsilon_k} + O(\lambda^{\varepsilon_k})$ for any

k. Proofs of the fact that all n solutions of (2) may be presented in the form (1) and constructed by the above algorithm are given in [3, 15, 18].

3. Implementation in Reduce

We have implemented the algorithm of Sect.2 in the computer algebra system `Reduce 3.3` [14]. The program is written in the symbolic mode (Rlisp) of the `Reduce` programming language. Input data for the program are a polynomial $F(x, \lambda) = 0$ with a given singular point (0,0). A nonlinear function of general form must be reduced to a polynomial to be used in our program. It may be done, for example, by the standard package for computation of a Teylor expansion which is available in `Reduce`.

The main procedures of our package are

`PUISE(f,x,l,nt)` - the procedure for the computation of `nt` terms of the Puiseux expansion. This procedure returns a list of all solutions of the equation `f(x,l)=0` in the form of (1).

`PUISE1(f,x,l)` is similar to the procedure **`PUISE`**, but computes only the first terms. The result has the form $\{\{\epsilon, x_\epsilon\}, \ldots\}$ and presents all the list of branches (7).

`RESULTPUISE()` returns the intermediate result of the procedure **`PUISE`** in the case of incorrect break of the procedure, for example, because of the lack of computer memory (see example 3 of Sect.4).

There is a switch **`ALLPUISE`** in our program. This switch controls the set of values of ϵ which has to be computed. There are three possible situations:

- $\epsilon < 0$, then a solution satisfies the condition $\lim_{\lambda \to 0} x(\lambda) = 0$.
- $\epsilon = 0$, then $\lim_{\lambda \to 0} x(\lambda) = const$.
- $\epsilon > 0$, then a solution has divergent ("big") terms in the limit $\lambda \to 0$.

The last two cases are avoided when computing if **`ALLPUISE`** is turned off. By default **`ALLPUISE`** is off.

4. Examples

Below we give three examples of use of our program. All computations were done by an IBM PC AT (12 Mhz). In accordance with our assumption of Sect.2 all input equations vanish at the point (0.0) of computation of the Puiseux expansions. Each example is supplied with complete computer output including comments to explain the main aspects of the computational procedure.

Example 1 [18]

```
equ:=w**3-3*z*w+2*z**2$
% Computation of the Puiseux expansions

puise(equ,w,z,3);

             3         2
{W=32/729*Z  + 8/81*Z  + 2/3*Z,

 W= - 1/18*SQRT(Z)*SQRT(3)*Z + SQRT(Z)*SQRT(3) - 1/3*Z,

 W=1/18*SQRT(Z)*SQRT(3)*Z - SQRT(Z)*SQRT(3) - 1/3*Z}

Time: 16260 ms
```

Example 2 [4]

```
% sin(w)-w+w**2*sin(z)-z**4=0
% Computation of a Taylor expansion

equ:=w**2*z-w**2*z**3/6+w**2*z**5/120-w**2*z**7/5040-w**3/6+
w**5/120-w**7/5040-z**4$

% And now computation of the Puiseux expansions

puise(equ,w,z,3);

                     2                      2
{W=5/288*SQRT(Z)*Z  + SQRT(Z)*Z + 1/12*Z  ,

                       2                      2
 W= - 5/288*SQRT(Z)*Z  - SQRT(Z)*Z + 1/12*Z  ,

             3        2
 W=5287/540*Z  - 1/6*Z  + 6*Z}
```

Example 3 [18]

```
equ:=2*z**7-z**8-z**3*w+(4z**2+z**3)*w**2+(z**3-z**4)*w**3-
4z*w**4+7z**5*w**5+(1-z**2)*w**6+5*z**6*w**7+z**3*w**8$
```

```
% Computation of the Puiseux expansions

puise(equ,w,z,3);

No more free cons cell.
ERROR 96  NIL
Cont? (Y or N)
y

Time: 120880 ms

%There are no more memory cells for determining the third term.
%Intermediate results can be printed out by using the
%procedure RESULTPUISE

4: resultpuise();
        5      4
{W= - Z  + 2*Z  ,

             2
 W= - 3/64*Z  + 1/4*Z,

        3/4  1/4
 W=1/4*Z   *2    + SQRT(Z)*SQRT(2),

           3/4  1/4
 W= - 1/4*Z   *2    + SQRT(Z)*SQRT(2),

        3/4
 W=1/4*Z   *SQRT( - SQRT(2)) - SQRT(Z)*SQRT(2),

           3/4
 W= - 1/4*Z   *SQRT( - SQRT(2)) - SQRT(Z)*SQRT(2)}

Time: 2360 ms
```

5. Conclusion

As mentioned in Sect.2, the main restriction in the use of our program is connected with quite restrictive facilities of the package Solve of the Reduce system in solving nonlinear algebraic equations. The above assumption $F(x,y) \in Q[x,y]$ means that

the first term of (1) being computed at the first cycle of the algorithm is an algebraic number. Therefore, in general case, to repeat the procedure, that is, to compute the next terms of (1) it is necessary to manipulate with algebraic numbers and to solve polynomial equations in algebraic extensions of Q. The **Reduce**-facilities for the manipulation with algebraic numbers available in **Anum** package [14] are far from sufficient for these computations. A number of new ideas and their computer implementations given in [9, 11, 13] are very fruitful for further developments and generalizations.

In many applications one needs to manipulate with the Puiseux expansions as mathematical objects, i.e. to compute their sum, product, degree, inversion, decomposition, etc. There are useful **Reduce** packages [19, 20] which allow one to carry out these operations automatically by computer as well.

We would like to acknowledge N.A.Kostov and Z.T.Kostova for useful discussions and suggestions.

References

[1] Coates J. Construction of Rational Functions on a Curve, *Proc.Camb.Phyl.Soc.* 68, 1970, 105-123.

[2] Davenport J.H. On the Integration of Algebraic functions, *Lecture Notes in Computer Science*, 102, Springer-Verlag, 1981.

[3] Weinberg M.M., Trenogin B.A. Theory of Branching of Solutions of Nonlinear Equations, Nauka, Moscow, 1969 (in Russian).

[4] Trenogin B.A. Functional Analysis, Nauka, Moscow, 1980 (in Russian).

[5] Bronstein M. Integration of Elementary Functions, *J.Symb.Comp.* 9, 2, 1990, 117-173.

[6] Brieskorn E., Knörrer H. Ebene algebraische Kurven, Birkhaüser Verlag, Basel, 1981.

[7] Chudnovsky D., Chudnovsky G. Power Series and Fast Numeric Evaluation of Algebraic Functions and Linear Differential Equations, *The Scratchpad II Newsletter*, 1, No.3, 1986, 1-2.

[8] Gerdt V.P., Kostov N.A. Computer Algebra in the Theory of Ordinary Differential Equations of Halphen Type, in: Computers and Mathematics (eds. Kaltofen E. and Watt S.M.), Springer-Verlag, New-York, 1989, 279-288.

[9] Duval D. Diverses questions relatives au calcul formel avec des nombres algébriques, Thèse de doctorat d'état (Institut Fourier), 1987.

[10] Gerdt V.P., Kostov N.A., Kostova Z.T. Computer Algebra and Computation of Puiseux Expansions of Algebraic Functions, in "EUROCAL'87", *Lecture Notes in Computer Science* 378, Springer-Verlag, 1987, 206.

[11] Rybowicz M. Sur le calcul des places et de l'anneau des entiers d'un corps de fonctions algebriques. PhD thesis, Université de Limoges, France, 1990.

[12] Henry J.P.G., Merle M. Complexity of Computation of Embedded Resolution of Algebraic Curves, in: "EUROCAL'87", *Lecture Notes in Computer Science* 378, Springer-Verlag, 1987, 381-390.

[13] Baladi V. Guillement J.P. A Program for Computing Puiseux Expansions, *SIGSAM Bulletin* 24, 4, 1990, 33-41.

[14] Hearn A.C. REDUCE User's Manual, version 3.3, Rand Corporation, Santa Monica, 1987.

[15] Walker R.J. Algebraic Curves, Springer-Verlag, New-York, 1978.

[16] Hilali A. On the Algebraic and Differential Newton-Puseux Polygons, *J.Symb. Comp.* 4, 3, 1987, 335-350.

[17] Davenport J.H., Siret Y., Tournier E. Computer Algebra. Systems and Algorithms for Algebraic Computation, Academic Press, London, 1988.

[18] Chebotarev N. Theory of Algebraic Functions, Moscow, 1949 (in Russian).

[19] Feldmar E., Kölbig K.S. Reduce Procedures for the Manipulation of Generalized Power Series, *Comp.Phys.Comm.*. 39, 1986, 267-284.

[20] Padget J., Barnes A. Univariate Power Series Expansions in Reduce, in: "ISSAC'90" *(International Symposium on Symbolic and Algebraic Computation)*, ACM Press, Addison-Wesley Publishing Co., 1990, 82-87.

Received by Publishing Department
on September 2, 1991.