浙江工业大学

本科毕业设计论文

设计(论文)

题　　目　非线性演化方程的精确波解自动推导研究

指导教师
姓　　名　周明华

学　　生
姓　　名　余江涛

学　　生
学　　号　201210790427

院　　系　理学院

专　　业　信息与计算科学

班　　级　1201

非线性演化方程的精确波解自动推导研究

学生姓名：余江涛　　指导老师：周明华

浙江工业大学理学院

摘　　要

本文针对非线性演化方程的精确波解自动化求解问题，采用了改进的双曲正切函数方法进行求解，在 Maple 上实现了自动求解的软件包 IRATH。本文从三个方面对传统双曲正切函数方法进行了改进：(1)对阶数平衡的条件进行了完整的分析；(2)在处理非正整数阶数时，对前人的阶数变换方法进行了严格论证，并讨论了可以使用阶数变换的条件；(3)对于多项式方程组的求解问题，本文提出了求解效率更高、效果更好的实现方法。利用 IRATH 本文已成功求解大量的非线性演化方程，不仅求出了已知的解，对于某些方程还得到了新的解以及形式更一般的解。

关键词：非线性演化方程 孤立波解 双曲正切函数方法 Maple

The studies on automatically finding exact wave solutions of nonlinear evolution equations

Student:JiangTao YU Advisor: MingHua Zhou

College of Science
Zhejiang University of Technology

Abstract

We use the improved tanh function method to find exact wave solutions of nonlinear evolution equations automatically, and implement a Maple package for this method. We improved the classic tanh function method in three aspects: (1) We completely considered the of order balance condition. (2) We proved the correctness of the order transform method which deal with the non-positive integer orders, and proposed the service condition of this method. (3) We proposed a more effective and fast algorithm to solve the polynomial equations. We solved a lot of nonlinear evolution equations utilizing the package IRATH. It successfully recovered all previously known solutions. And more, for some equations, we have found new solutions and more general form of solutions.

Keywords：Nonlinear Evolution Equations; Exact Wave Solution; Tanh Function Method; Maple

目　录

第一章 引言

1.1 求解非线性演化方程的背景和意义

非线性演化方程（Nonlinear Evolution Equations）是一类偏微分方程，它描述了随时间而演变的过程。很多意义重大的自然科学和工程技术问题都可以归结为非线性演化方程的研究。寻找非线性演化方程的精确解能够帮助我们更加深入的洞察物理现象的本质。

寻找非线性演化方程的精确解，一直以来是数学家和物理学家所关注的问题。求得一些具有非常重要的物理意义的非线性演化方程的精确解，能够对一些重要的物理现象作出科学的解释，使得人们能够更加深入的洞悉物理现象的本质。例如大气旋转波的方程孤子解，可用来解释木星的红斑和其它特征。

寻找非线性演化方程孤子解的方法多种多样，如著名的反射方法以及 Hirota 变换、Darboux 变换、Backlund 变换等各种函数变换方法[1-4]。但是这些方法的应用范围有限，并且涉及复杂的数学技巧，也不方便进行推广应用。在 Maple、Mathematica 等计算机符号计算系统出现之后，采用计算机代数的方法来构造非线性演化方程的精确解越来越受到人们的青睐。

采用符号计算的方法，将繁琐的代数计算交给计算机来完成，能够帮助我们寻找更多非线性演化方程的精确解。对与已知的方程，也可以发现形式更一般的解，或是发现新的解。

1.2 双曲正切函数方法的国内外研究现状

一般来说，求解偏微分方程的解析解（也称精确解），是十分困难的。只有某一类方程在经过多年的研究之后，人们找到了一系列构造精确解的方法。这类方程是非线性演化方程。

在对非线性演化方程进行求解的时候往往涉及复杂而繁琐的代数计算，为了减轻求解方程的负担以及确保解的正确性，采用计算机代数的方法进行方程求解正变的越来越受欢迎。

伴随着 Maple、Mathematica 等计算代数系统出现和发展，近年来采用符号计算的方法进行非线性演化方程求解的方法也出现了很多。

90 年代初 Malfliet[5]提出的“双曲正切方法”就是最著名的方法之一。双曲正切方法在对非线性演化方程进行行波变换，转化为常微分方程的基础上，假设方程具有双曲正切函数（tanh 函数）的多项式形式的解，利用双曲正切函数的导数是其本身的多项式的性质，将常微分方程转化为多项式方程进行求解。许多重要的非线性演化方程都能通过这个方法得出重要的解，同时获得了这些方程的一些重要性质。

Li[7]在 1997 年首次编写了基于双曲正切方法求解的非线性演化方程的程序，并在代数方程的求解中引入了吴文俊消元法，但是只能在交互的方式下完成方程的而求解。

Parks[6]在 1998 年基于 Mathematica 编写了“双曲算术正切方法”程序 ATFM，对于许多方程能够自动给出解。但是很多时候，程序的运行还是需要人工干预。对于有的方程，其关键参数还需要人工计算。对于过于复杂的方程，还需要人工进行预处理。ATFM 并没有做到完全的自动化。

2002 年，Liu[8,9]等人克服了[6,7]在自动求解上的不足，在 Maple 上编写了 RATH 软件包，完全自动的实现了双曲正切函数方法。同时，Liu 还首次引入了一个变换，可以在求得的双曲正切函数多项式的阶数不是正整数的时候，对方程进行求解，获得关于双曲正切函数多项式的幂次形式的解。

1.3 本文工作

本文以参考文献[8,9]的工作为基础，继续深入研究双曲正切函数方法，对其进行拓展和深化。本文主要在以下三方面对原有方法进行了深化和改进：

- 对于方程的定阶问题，本文对原有方法进行了深化，引入非线性演化方程的一般形式，对方程的阶数平衡方程进行了详细的分析，深入探讨了阶数平衡的条件。
- 对于文献[8,9]提出的变换方法，本文给出了该方法可行性的证明，同时分析了该方法成立的条件。
- 多项式代数方程的求解是双曲正切函数函数方法中最为关键的一步，不同的求解方法会产生不同的结果。本文采用求解方法在求解复杂方程时，速度会比文献[8,9]所提出的方法更快，并且在有些时候能够得到更好的解。

第二章　求非线性演化方程的双曲正切方法

双曲正切函数方法是求解非线性演化方程孤波解的著名方法之一，它首先对非线性演化方程进行行波变换，再假设方程具有关于 tanh 的多项式形式的解，利用 tanh 的导数是其本身的多项式的特性，将原方程转化为多项式代数方程求解。

2.1 方法概述

考虑只包含一个未知函数$u\left(x,t\right)$的非线性演化方程

$$H\left(u,u_t,u_x,u_{xx},...\right)=0 \tag{2.1}$$

其中H是关于变元$u,u_t,u_x,u_{xx},...$的多项式。该方程描述了孤波$u\left(x,t\right)$的动态演化过程。

在未知函数$u\left(x,t\right)$中，变量t表示时间，变量x一般用于表示位置，从而$u\left(x,t\right)$一般用于表示t时刻孤立波u在x点的振幅。

假设孤波$u\left(x,t\right)$的形状不变，但它的位置随着时间平移，则称它被称为行波。

在假设非线性演化方程(2.1)具有行波解时，可以对他进行行波变换，即令

$$u=u\left(\xi\right),\xi=k\left(x-ct\right)+\xi_0 \tag{2.2}$$

其中k（波数）和c（波速）为待定常数，ξ_0为任意常数。

在进行上述代换之后，方程(2.1)中的偏导计算可以简化为导数计算。通过

$$\frac{\partial u}{\partial t}=\frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial t}=-ck\frac{\partial u}{\partial \xi}=-ck\frac{du}{d\xi} \tag{2.3}$$

$$\frac{\partial u}{\partial x}=\frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial x}=k\frac{\partial u}{\partial \xi}=k\frac{du}{d\xi} \tag{2.4}$$

我们可以将方程(2.1)化作关于变量ξ的常微分方程

$$F\left(u,u',u'',...\right)=0 \tag{2.5}$$

假设方程(2.5)具有双曲正切函数多项式形式的解，即

$$u\left(\xi\right)=\sum_{k=0}^{m}a_kT^k,T=\tanh\left(\xi\right) \tag{2.6}$$

其中$a_0,a_1,...,a_m$为待定系数。

由于双曲正切函数T满足

$$T'=1-T^2 \tag{2.7}$$

故$u\left(\xi\right)$的导数依然是T的多项式。从而方程(2.5)最终可以表示为关于T的多项式方程。

注意到

$$\frac{d}{d\xi}u(\xi)=\frac{d}{d\xi}\sum_{k=0}^{m}a_kT^k=\frac{d}{dT}\sum_{k=0}^{m}a_kT^k\frac{dT}{d\xi}=\left(1-T^2\right)\sum_{k=1}^{m}a_kkT^{k-1} \tag{2.8}$$

因此，$du/d\xi$ 中 T 的最高幂次，即 $du/d\xi$ 的阶数为

$$O\left(\frac{du}{d\xi}\right)=m+1 \tag{2.9}$$

通过递推可以得到，$du^p/d\xi^p$ 的阶数为

$$O\left(\frac{d^pu}{d\xi^p}\right)=m+p \tag{2.10}$$

根据多项式相乘的性质，还能得到

$$O\left(u^q\frac{d^pu}{d\xi^p}\right)=(q+1)m+p \tag{2.11}$$

在传统的双曲正切方法中，一般通过方程平衡(2.5)中线性最高阶幂次和非线性最高阶幂次来确定阶数 m 的取值。

将确定阶数的方程(2.6)代入(2.5)中，可以得到关于 T 的多项式方程。要使该多项式方程成立，只需令 T 的每一项系数为 0 即可。

从而我们可以得到一些列关于待定系数 $a_0,a_1,...,a_m,k,c$ 的多项式方程。求解这些多项式方程，确定待定系数的值，就可以得到方程(2.1)的孤立波解

$$u(x,t)=\sum_{k=0}^{m}a_k\tanh^k\left[k(x-ct)+\xi_0\right] \tag{2.12}$$

2.2 应用举例

考虑 KdV 方程

$$u_t+uu_x+\alpha u_{xxx}=0 \tag{2.13}$$

在行波变换下，该方程转化为

$$-cu'+uu'+\alpha k^2u'''=0 \tag{2.14}$$

要求解方程(2.14)首先需要确定阶数 m 的取值。

在方程(2.14)中，最高阶导数项为 u'''，最高阶非线性项为 uu'，平衡方程为 $m+3=2m+1$，可以得到 $m=2$。从而，可以设原方程的解为

$$u=a_0+a_1T+a_2T^2 \tag{2.15}$$

将(2.15)带入(2.14)，对 T 合并同类项，令多项式的各项系数为 0，可以得到多项式代数方程组：

$$\begin{cases}-24a_2\alpha k^2-2{a_2}^2=0\\-6a_1\alpha k^2-3a_1a_2=0\\40a_2\alpha k^2+2a_2c-2a_0a_2-{a_1}^2+2{a_2}^2=0\\8a_1\alpha k^2+a_1c-a_0a_1+3a_1a_2=0\\-16a_2\alpha k^2-2a_2c+2a_0a_2+{a_1}^2=0\\-2a_1\alpha k^2-a_1c+a_0a_1=0\end{cases} \tag{2.16}$$

求解上述方程组可以得到

$$a_0 = c + 8\alpha k^2, a_1 = 0, a_2 = -12\alpha k^2 \tag{2.17}$$

即原方程有解

$$u(x,t) = c + 8\alpha k^2 - 12\alpha k^2 \tanh^2\left[k(x-ct) + \xi_0\right] \tag{2.18}$$

第三章　双曲正切方法的改进

从上一节的描述可以看出，双曲正切方法的原理十分简单，且能够有效的求解非线性演化方程的孤波解。

但是，传统的双曲正切方法在以下三个方面存在着缺陷：

1) 方程定阶。传统的双曲正切方法通过平衡线性项和非线性项的最高项幂次来确定阶数的取值，没有考虑非线性项的阶数互相平衡的情况。本文将更加全面地分析阶数平衡的条件。
2) 对于方程的阶数 m 不是正整数的情况，双曲正切函数方法将无法求解。文献[8,9]提出了一种变换来处理这种情况，但并未对其作出证明，也没有讨论该变换成立条件。本文将对这种变换的正确性做出证明，并讨论变换成立的条件。
3) 多项式方程组的具体求解方法。求解多项式方程组是双曲正切方法和核心内容，不同的求解方法得到解可能会不同。文献[7-9]引入了吴文俊消元法来对多项式方程组进行求解，而本文将基于他们的结果，给出了效率更高的一种实现方法。

3.1 方程定阶

由于计算机只能对有限项的多项式进行计算，于是，要求解方程(2.5)，首先要确定阶数 m 的值，这一过程称为方程的定阶。

因为最终要求 T 的各项系数都为 0，所以我们可以首先要求 T 的最高项系数为 0，以此为出发点，来分析 m 的取值。

首先，我们引入常微分方程(2.5)的一般形式如下

$$\sum_{i=1}^{N} c_i \prod_{j=1}^{D_i} \frac{d^{p_{ij}}}{d\xi^{p_{ij}}} u(\xi) = 0 \tag{3.1}$$

该方程是关于 $u, u', u'', \ldots$ 的多项式方程。其中 N 表示多项式的项数，c_i 是每一项的系数，$\prod_{j=1}^{D_i} \frac{d^{p_{ij}}}{d\xi^{p_{ij}}} u(\xi)$ 是多项式每一项的变元，表示多项式的每一项由若干个 $u, u', u'', \ldots$ 相乘构成。D_i 是多项式每一项的次数，$D_i = 1$ 则称第 i 项为线性项，$D_i > 1$ 则称第 i 项为非线性项。

记 $u(\xi)$ 的最高项幂次为 $O(u(\xi)) = m$，$u(\xi)$ 的最高项系数为 $K(u(\xi)) = a_m$。

注意到

$$\frac{d}{d\xi} u(\xi) = \frac{d}{d\xi} \sum_{k=0}^{m} a_k T^k = \frac{d}{dT} \sum_{k=0}^{m} a_k T^k \frac{dT}{d\xi} = (1 - T^2) \sum_{k=1}^{m} a_k k T^{k-1} \tag{3.2}$$

因此，$du / d\xi$ 的最高项幂次为 $m+1$，$du / d\xi$ 的最高项系数为 $-a_m m$。经过递推可以得到

$$O\left(\frac{d^p u}{d\xi^p} \right) = m + p \tag{3.3}$$

$$K\left(\frac{d^p u}{d\xi^p}\right)=(-1)^p\, m\cdot(m+1)\cdots(m+p-1)a_m=(-1)^p\frac{(m+p-1)!}{(m-1)!}a_m=k_p a_m \tag{3.4}$$

其中 $K\left(d^p u/d\xi^p\right)$ 的表达式比较复杂，我们并不关心 $K\left(d^p u/d\xi^p\right)$ 的具体取值，只需发现它是和 a_m 成正比的即可。

从而可以得到方程(3.1)每一项的阶数和系数为：

$$O\left(c_i\prod_{j=1}^{D_i}\frac{d^{p_{ij}}}{d\xi^{p_{ij}}}u(\xi)\right)=\sum_{j=1}^{D_i}\left(m+p_{ij}\right)=D_i m+\sum_{j=1}^{D_i}p_{ij}=D_i m+P_i \tag{3.5}$$

$$K\left(c_i\prod_{j=1}^{D_i}\frac{d^{p_{ij}}}{d\xi^{p_{ij}}}u(\xi)\right)=c_i\left(\prod_{j=1}^{D_i}k_{p_{ij}}\right)a_m^{D_i}=K_i a_m^{D_i} \tag{3.6}$$

由于 $a_m\neq 0$（$a_m=0$ 则与 $u(\xi)$ 是关于 T 的 m 次多项式矛盾），$c_i\neq 0$（$c_i=0$ 则方程这一项不存在），以及 $k_{p_{ij}}\neq 0$（从方程(3.4)可以看出），我们可以得到

$$K\left(c_i\prod_{j=1}^{D_i}\frac{d^{p_{ij}}}{d\xi^{p_{ij}}}u(\xi)\right)\neq 0 \tag{3.7}$$

从而，要使原方程成立需要阶数相同的项的系数之和为 0。

在定阶阶段，我们首先考虑让方程的最高阶项的系数为 0。显然，方程的最高阶项需要有两项，因为只有一项的话，该项的系数不为 0。不妨设方程的最高阶项为第 i 项和第 j 项。则有

$$\begin{cases}D_i m+P_i=D_j m+P_j\\ K_i a_m^{D_i}+K_j a_m^{D_j}=0\end{cases} \tag{3.8}$$

该方程被称为是原方程的阶数平衡方程，满足该方程的 m 称作是原方程的平衡阶数。

在 $D_i\neq D_j$ 时，不妨设 $D_i>D_j$，m 有唯一解

$$m=\frac{P_i-P_j}{D_j-D_i} \tag{3.9}$$

此时，a_m 有非零解

$$a_m=\left(\frac{K_j}{K_i}\right)^{D_j-D_i} \tag{3.10}$$

在 $D_i=D_j$ 时，若 $P_i=P_j$，则 m 有无穷多组解，此时需要满足 $K_i+K_j=0$，原方程才能有解。

在之前的双曲正切方法中，一般只考虑平衡线性最高阶项和非线性最高项的幂次，即只考虑了 $D_i>D_j=1$ 的特殊情况，并没有考虑所有可能的情况。

要考虑所有可能的情况，只需考虑函数

$$f(m)=\max_{i\in\{1,..,N\}}\left\{D_i m+P_i\right\} \tag{3.11}$$

的平衡点即可。

设 m_0 是 $f(m)$ 的一个平衡点，则有：存在 $i\neq j$，满足 $D_i m+P_i=D_j m+P_j$，且对任意的 k，满足 $D_i m+P_i=D_j m+P_j\geq D_k m+P_k$，此时 $D_i m+P_i=D_j m+P_j$ 恰好为方程的两个最高阶项。

若 $D_i m+P_i$ 各不相等，则只会出现 $D_i \neq D_j$ 的平衡方程，因此原方程的平衡阶数 m 可能的取值是有限的，在此情况下，所有的平衡点都是 $f\left(m\right)$ 的拐点。

若存在 $D_i m+P_i$ 的相等的情况，则有可能出现 $D_i = D_j$ 的平衡方程，原方程的平衡阶数可能有无数个取值。

例如，对于

$$f\left(m\right)=\max\left\{3m+3,3m+3,4m+1\right\} \tag{3.12}$$

由 $3m+3=4m+1$ 可以得到 $m=2$；又任意的 m 都满足 $3m+3=3m+3$，在 $3m+3>4m+1$，且 m 是正整数的条件下，还可以得到 $m=1$ 这个解。此时，方程的平衡点集合为 $M=\left\{1,2\right\}$，尽管出现了阶数完全相同的情况，但可能的平衡阶数还是有限的。对于这种情况，我们可以先设定原方程的阶数为 $\max M$，在求解时也保留阶数小于 $\max M$ 的解，就能获得所有的解。

例如，对于

$$f\left(m\right)=\max\left\{2m+1,2m+1,m+2\right\} \tag{3.13}$$

其平衡点可以取遍所有正整数。对于这种情况，由于计算机不能计算任意阶数的多项式，所以如何求解该类方程还需进一步讨论。本文并没有解决这一类方程的求解，这一类方程的求解是本文的改进方向。

3.2 阶数变换

一般情况下，双曲正切函数方法只能处理阶数 m 是正整数的情况。文献[8,9]提出了一种变换来处理这种情况。

在 m 不是正整数的情况下，取

$$v\left(\xi\right)=u\left(\xi\right)^{\mathrm{sgn}\left(m\right)/d} \tag{3.14}$$

其中 $\mathrm{sgn}\left(m\right)$ 表示 m 的符号，d 表示 m 的分母。

将(3.14)代入原方程，可以将原方程化为关于 $v\left(\xi\right)$ 的方程，这是方程的阶数一定是 m 的分子，对变换后的方程能够采用双曲正切函数方法进行求解。

该变换实际上是假设原方程具有

$$u\left(\xi\right)=v^{\alpha}\left(\xi\right) \tag{3.15}$$

形式的解。其中

$$v\left(\xi\right)=\sum_{k=0}^{m}a_k T^k, T=\tanh\left(\xi\right) \tag{3.16}$$

将(3.15)带入原方程，可以将方程转化为

$$\sum_{i=1}^{N}c_i\prod_{j=1}^{D_i}\frac{d^{p_{ij}}}{d\xi^{p_{ij}}}v^{\alpha}\left(\xi\right)=0 \tag{3.17}$$

要分析该方程是否有解，首先要分析 $\dfrac{d^{p_{ij}}}{d\xi^{p_{ij}}}v^{\alpha}\left(\xi\right)$ 具有怎样的形式。

根据 Faà di Bruno 公式[10]，有

$$\frac{d^n}{dx^n}g\left(f\left(x\right)\right)=\sum\frac{n!}{\prod_{i=1}^{n}b_i!}g^{(k)}\left(f\left(x\right)\right)\prod_{i=1}^{n}\left(\frac{f^{(i)}\left(x\right)}{i!}\right)^{b_i} \tag{3.18}$$

其中$b_1,b_2,...,b_n$是$\sum_{i=1}^{n}ib_i=n$的非负整数解，且有$k=\sum_{i=1}^{n}b_i$，$g^{(k)}\left(x\right)$表示$g\left(x\right)$的k阶导数。

取$g\left(x\right)=x^{\alpha}$可以得到

$$\frac{d^n}{dx^n}f^{a}\left(x\right)=\sum\frac{n!}{\prod_{i=1}^{n}b_i!}k!\binom{\alpha}{k}f^{\alpha-k}\left(x\right)\prod_{i=1}^{n}\left(\frac{f^{(i)}\left(x\right)}{i!}\right)^{b_i} \tag{3.19}$$

其中

$$\binom{\alpha}{k}=\frac{\alpha\left(\alpha-1\right)\cdots\left(\alpha-k+1\right)}{k!} \tag{3.20}$$

因此

$$\frac{d^{p_{ij}}}{d\xi^{p_{ij}}}v^{\alpha}\left(\xi\right)=\sum_{l}K_{ijl}v^{\alpha-l}\left(\xi\right)\prod_{q=1}^{p_{ij}}\left(\frac{v^{(q)}\left(\xi\right)}{q!}\right)^{b_{ijq}} \tag{3.21}$$

其中

$$K_{ijl}=\frac{p_{ij}!}{\prod_{q=1}^{p_{ij}}b_{ijq}!}l!\binom{\alpha}{l} \tag{3.22}$$

$$\sum_{q=1}^{p_{ij}}qb_{ijq}=p_{ij} \tag{3.23}$$

$$\sum_{q=1}^{p_{ij}}b_{ijq}=l \tag{3.24}$$

从而，原方程可以变为

$$\sum_{i=1}^{N}c_i\prod_{j=1}^{D_i}\left[\sum_{l}K_{ijl}v^{\alpha-l}\left(\xi\right)\prod_{q=1}^{p_{ij}}\left(\frac{v^{(q)}\left(\xi\right)}{q!}\right)^{b_{ijq}}\right]=0 \tag{3.25}$$

要使方程的左端能够表示为T的多项式，需要方程中所有形如$v^{\beta}\left(\xi\right)$的项的幂次$\beta$都为非负整数。

可以对方程两端都乘上一个合适的$v^{\delta}\left(\xi\right)$来调节各项中$v\left(\xi\right)$的幂次。将$v^{\delta}\left(\xi\right)$合理的分配到方程的每一项，可以得到

$$
\begin{aligned}
0 &= v^{\delta}(\xi)\sum_{i=1}^{N} c_i \prod_{j=1}^{D_i}\left[\sum_{l} K_{ijl} v^{\alpha-l}(\xi)\prod_{q=1}^{p_{ij}}\left(\frac{v^{(q)}(\xi)}{q!}\right)^{b_{ijq}}\right] \\
&= \sum_{i=1}^{N} c_i v^{\bar{\delta}_i}(\xi)\prod_{j=1}^{D_i}\left[v^{\delta_{ij}}(\xi)\sum_{l} K_{ijl} v^{\alpha-l}(\xi)\prod_{q=1}^{p_{ij}}\left(\frac{v^{(q)}(\xi)}{q!}\right)^{b_{ijq}}\right] \\
&= \sum_{i=1}^{N} c_i v^{\bar{\delta}_i}(\xi)\prod_{j=1}^{D_i}\left[\sum_{l} K_{ijl} v^{\alpha-l+\delta_{ij}}(\xi)\prod_{q=1}^{p_{ij}}\left(\frac{v^{(q)}(\xi)}{q!}\right)^{b_{ijq}}\right]
\end{aligned}
\tag{3.26}
$$

满足

$$
\delta = \bar{\delta}_i + \sum_{j=1}^{D_i}\delta_{ij} \tag{3.27}
$$

当且仅当方程(3.26)中，每一个 $\bar{\delta}_i$ 和 $\alpha - l + \delta_{ij}$ 都为非负整数时，方程才是关于 T 的多项式方程。

在 $\bar{\delta}_i$ 和 $\alpha - l + \delta_{ij}$ 都为非负整数的条件下，方程每一项的阶数

$$
\begin{aligned}
&O\left(c_i v^{\bar{\delta}_i}(\xi)\prod_{j=1}^{D_i}\left[\sum_{l} K_{ijl} v^{\alpha-l+\delta_{ij}}(\xi)\prod_{q=1}^{p_{ij}}\left(\frac{v^{(q)}(\xi)}{q!}\right)^{b_{ijq}}\right]\right) \\
&= m\bar{\delta}_i + \sum_{j=1}^{D_i}\left[m\left(\alpha - l + \delta_{ij}\right) + \sum_{q=1}^{p_{ij}}(m+q)b_{ijq}\right] \\
&= m\bar{\delta}_i + \sum_{j=1}^{D_i}\left[m\left(\alpha - l + \delta_{ij}\right) + m\sum_{q=1}^{p_{ij}} b_{ijq} + \sum_{q=1}^{p_{ij}} q b_{ijq}\right] \\
&= m\bar{\delta}_i + \sum_{j=1}^{D_i}\left[m\left(\alpha - l + \delta_{ij}\right) + ml + p_{ij}\right] \\
&= m\left(\bar{\delta}_i + \sum_{j=1}^{D_i}\delta_{ij}\right) + mD_i\alpha + \sum_{j=1}^{D_i} p_{ij} \\
&= m\delta + mD_i\alpha + \sum_{j=1}^{D_i} p_{ij} \\
&= m\delta + mD_i\alpha + P_i
\end{aligned}
\tag{3.28}
$$

于是，变换后方程的平衡阶数满足

$$
m\delta + mD_i\alpha + P_i = m\delta + mD_j\alpha + P_j \tag{3.29}
$$

记变换后方程的平衡阶数为 m_v，显然

$$
m_v = \frac{1}{\alpha}\frac{P_i - P_j}{D_j - D_i} \tag{3.30}
$$

与原方程的平衡阶数

$$
m_u = \frac{P_i - P_j}{D_j - D_i} \tag{3.31}
$$

相比，满足

$$m_u = \alpha m_v \tag{3.32}$$

因此：

- 当原方程的阶数 m_u 是正整数时，因为 $v(\xi)$ 和 $v^n(\xi)$ 在形式上是等价的，都是关于 T 的多项式，只是最高项的表示形式不同，但意义是相同的，所以不需要进行变换。
- 当原方程的阶数 m_u 是负整数时，因为 $v^{-n}(\xi)$ 和 $v^{-1}(\xi)$ 是等价的，所以可以取 $u(\xi)=\left[v(\xi)\right]^{-1}$ 代入，变换后方程的阶数变为 $m_v = -m_u$，是正整数。
- 当原方程的阶数 m_u 是分数时，设 $m_u = n_u / d_u, (d_u > 0)$，可以取 $u(\xi)=\left[v(\xi)\right]^{\mathrm{sgn}(m_u)/d_u}$ 带入，则变换后方程的阶数 $m_v = \mathrm{sgn}(n_u) n_u = |n_u|$ 是正整数。

综合上述三点，我们可以得到，当原方程的阶数 m_u 不是正整数时，可以取

$$u(\xi)=\left[v(\xi)\right]^{\mathrm{sgn}(m_u)/d_u} \tag{3.33}$$

代入原方程，变换后方程的阶数一定是正整数。即我们证明了文献[8,9]提出的变换的可行性。

但是上述结论要在 $\overline{\delta_i}$ 和 $\alpha - l + \delta_{ij}$ 都为非负整数的条件下才能成立。若该条件不成立，则变换后的方程不能表示为 T 的多项式方程，就不能讨论方程的阶数，也不能用双曲正切函数方法进行求解。

例如，对于非线性演化方程：

$$u_{xx} + \alpha u u_t + \beta u^2 u_x = 0 \tag{3.34}$$

进行行波变换后得到

$$k u'' - \alpha c u u' + \beta u^2 u' = 0 \tag{3.35}$$

求 $\max\{m+2, 2m+1, 3m+1\}$ 的平衡点得到 $m = 1/2$，故取 $u(\xi)=\left[v(\xi)\right]^{1/2}$ 代入，得到

$$2\beta v^2 v' - 2\alpha c v^{3/2} v' + 2k v v'' - k(v')^2 = 0 \tag{3.36}$$

因为 $v^{3/2}$ 不是关于 T 的多项式，所以该方程不能用双曲正切函数方法进行求解，即变换(3.33)不适用于方程(3.34)。

因此，对方程进行变换并不适用于原方程的阶数不是正整数的情况，当变换后的方程不能表示为关于 tanh 函数的多项式时，不能采用双曲正切方法求解变换后的方程。

尽管变换(3.33)不总是正确的，但是从解方程的角度来讲，我们可以对所有的方程都进行该变换，只对变换后能够表示为 tanh 的多项式的方程进行求解即可。

因此，文献[8,9]提出的变换是可行的，但是在变换后需要检查方程是否能够表示为 tanh 函数的多项式，才能继续求解。

3.3 方程求解

将原方程化为关于双曲正切函数的多项式方程后，求解 tanh 的系数的多项式方程组是双曲正切函数方法的核心任务。

若直接采用 Maple 的 solve 命令进行求解，给出的解不够全面。因此，文献[7-9]提出了采用吴文俊消元法进行多项式方程进行求解。文献[8,9]基于王东明[11]编写的吴文俊消元法函数包 charsets 的求解命令 csolve 设计了求解算法。具体过程如下：

step 1. 不难证明，多项式方程组 PS 的前 $m+1$ 个方程是关于 $a_k\ (k=0,1,...,m)$ 的三角化组。先求解这些方程可以将 $a_k\ (k=0,1,...,m)$ 用待定系数 k,c 以及方程的参数表示出来，将前 $m+1$ 个方程的解集记为 SOL。

step 2. 对于 SOL 的每一项 SOL_i，执行 step3 至 step5。

step 3. 将 SOL_i 代入 PS 中的剩余方程得到 $PS2$， $PS2$ 是关于剩余未知参数的方程组。

step 4. 若 $PS2$ 仍是多项式方程组，则采用 csolve 进行求解；若 $PS2$ 中含有根式，则采用 solve 进行求解，解集记为 $SOL2$。

step 5. 将解集 $SOL2$ 中的每一项 $SOL2_j$ 和 SOL_i 进行合并，得到 SOL_k，将 SOL_k 添加到方程的解集 S 中去。

step 6. 消去解集 S 中所有平凡的解和复解。

step 7. 采用 csolve 和 solve 求得的解，其参数次数不能由用于完全决定，会存在一些不规则的解。例如，存在某个参数用 a_k 进行的表示的等式，对于这些等式，则继续求解 a_k。在最终得到的解中，$a_0,a_1,...,a_m,k,c$ 将用方程的参数进行表示。

本文主要出于以下几点考虑，对上述算法进行了修改：

1) 由于最终的结果应该是待定系数 $a_0,a_1,...,a_m,k,c$ 都用参数进行表示。因此，本文认为，先求解 $a_0,a_1,...,a_m,k,c$，再对剩余参数求解，顺序更加合理。
2) 非线性演化方程一般不需要复解，本文在需要使用利用 Maple 的 solve 命令进行求解时，采用 RealDomain[solve]命令进行求解，可以只在实数域内进行求解，速度较快，同时又能直接舍去之前产生的复解。
3) 在得到不规则的解之后，将其转化为规则的解是一个比较困难的任务。自己实现这个功能难免存在考虑不周的情况，遗漏掉一些解。本文利用 RealDomain[solve]能够指定参数顺序进行求解的特性，直接利用求解命令，将不规则的解转化为规则的解。
4) 本文考虑在求解过程中引入方程化简的过程，对方程进行因式分解消去非零项，通过简化方程来提高求解速度。

基于上述想法，本文对文献[8,9]中的求解算法进行了修改，提出了改进的求解算法。本文算法的具体步骤如下：

step 1. 对 PS 进行化简，消去 PS 中的每个方程的分母，并对分子进行因式分解，消去因式中的非零项。

step 2. 利用 csolve，对 PS 的前 $m+1$ 个方程进行求解，求解的变量为 $a_0,a_1,...,a_m,k,c$，在得到的结果中

$a_0, a_1, ..., a_m, k, c$ 将用方程的参数进行表示。得到的解集记为 SOL，消去其中平凡的解。

step 3. 对于 SOL 的每一项 SOL_i，执行 step4 至 step9。

step 4. 将 SOL_i 代入 PS 中的剩余方程中，得到关于参数的方程组 $PS2$，对 $PS2$ 采用和 step1 相同的方式进行化简。

step 8. 对化简后的 $PS2$，若 $PS2$ 仍是多项式方程组，则采用 csolve 进行求解；若 $PS2$ 中含有根式，则采用 Maple 的 RealDomain[solve]命令进行求解，解集记为 $SOL2$，消去其中平凡的解。

step 9. 对于 $SOL2$ 的每一项 $SOL2_j$，将其和 SOL_i 合并，构成新的方程组 $PS3$，对 $PS3$ 利用 Maple 的 RealDomain[solve]命令进行求解，可以规定求解变量为 $a_0, a_1, ..., a_m, k, c$，并且得到的结果将按照 $a_0, a_1, ..., a_m, k, c$ 的顺序进行显示。得到的解记为 SOL_k，将其并入到解集 S 中去。

step 10. 消去解集 S 中平凡的解，就可以得到原方程的所以 tanh 形式的解，且解能够按照 $a_0, a_1, ..., a_m, k, c$ 的顺序进行显示，且 $a_0, a_1, ..., a_m, k, c$ 将用方程的参数进行表示。

本文将在第五章中对本文的算法和文献[8,9]的算法进行对比，分析它们在求解效率和求解效果上的差异。

第四章　改进双曲正切方法的 Maple 实现

4.1 概述

本文在之前的章节中已经对双曲正切函数方法理论上的正确性和可行性进行了分析，并对传统的双曲正切方法作出了一定的改进。在本节中，将讨论如何将双曲正切函数方法程序化和自动化。

本文在计算机代数系统 Maple 编写了软件包 IRATH(Improved Real Automated TahH-function method)来实现改进的自动化双曲正切函数方法。

IRATH 主要由以下几个模块构成：

- findTanhSolutions()：主模块，是程序的入口，会依次调用下面的模块来完成非线性演化方程的求解工作，找到输入方程的 tanh 行驶的孤波解。
- getODE()：对方程进行行波变换，将原偏微分方程转换为常微分方程。
- findOrders()：确定转换后的常微分方程中的每一项的阶数关于 m 的表达式，为确定方程的平衡阶数做准备。
- findInflexion()：确定方程的平衡阶数，以及确定是否需要通过变换处理平衡阶数不是正整数的情况。
- solveAllOrder()：对于每一种可能的平衡阶数和方程变换的情况进行求解。
- printSolutions()：根据特定格式输出方程的解。

我们在 Maple2016 上实现了 IRATH 软件包，并将其打包成了“IRATH.mla”库文件，只需将该文件放在当前目录下，执行 whit(IRATH)命令，就可以用 findTanhSolutions()求解非线性演化方程了。

IRATH 的源代码以在附录一中给出。需要注意的是，本文的代码中保留了中文提示，而 Maple 对中文代码的读取和保存的支持并不好。本文编写了一个 Java 程序，将代码中的中文先转化为 Unicode 字节表示，再将其转化为 Maple 的转义字符，再将转化后的代码打包为“IRATH.mla”,从而保留了中文提示。

4.2 行波变换：getODE()

对非线性演化方程进行行波变换，将其转化为常微分方程，是双曲正切方法的第一步。在这一步中，需要对方程中的偏导符号进行替换：

$$\frac{\partial}{\partial t}\to -ck\frac{d}{d\xi},\frac{\partial}{\partial x}\to k\frac{d}{d\xi} \tag{4.1}$$

这对计算机而言并不是一个简单的任务。

在 Maple 中，任意一个符号表达式都被表示为一颗表达式树。这颗表达式树以操作符为根节点，以操作数为叶节点。

例如方程

$$u_t+uu_x+\alpha u_{xxx}=0 \tag{4.2}$$

的表达式树如下图所示。

表达式树以运算符优先级最低的“=”运算符为根节点。“=”用于表示表达式的相等关系，它有两个操

作数，分别为方程的两端。即“=”的左叶节点是方程的左端表达式，右叶节点是方程右端的表达式 0。

在方程的左端，优先级最低的运算符为“+”号，因此以“+”号为根节点。在 Maple 中，“+”可以有多个操作数，参与同一优先级的加法运算的元素都能作为“+”的操作数。在这个例子中，“+”的操作数是 u_t, uu_x 和 αu_{xxx}。

其中 uu_x 和 αu_{xxx} 还能根据“×”进行进一步划分。“×”的划分规则和“+”类似，同级别的乘法运算都能作为“×”的操作数。

而 Maple 中的 $x-y$ 是按照 $x+(-1)\cdot y$ 定义的，x/y 是按照 $x\cdot y^{-1}$ 定义的，其中 y^{-1} 的操作符是“^”，用于计算幂次。也就是说，Maple 进行代数计算的基本运算符是加号、乘号和幂次符号。

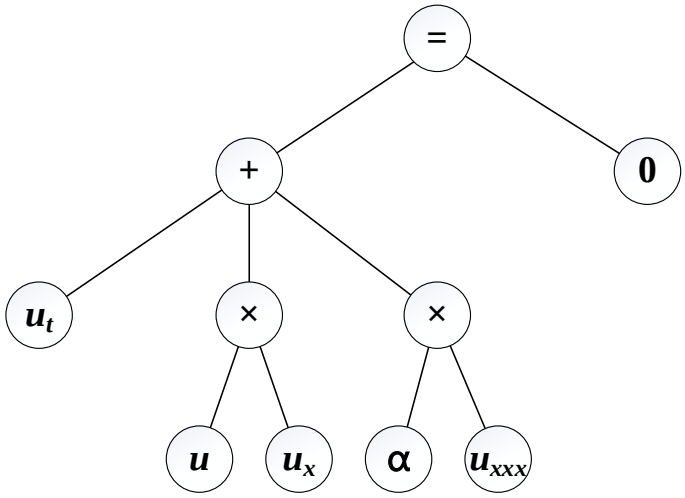

图 1 表达式树：$u_t + uu_x + \alpha u_{xxx} = 0$

在上图的表达式树中，节点 u, u_t, u_x, u_{xxx} 还能进行进一步展开。不失一般性，以 u_{xt} 进行举例，如下图所示。其中 u 表示函数 $u(x,t)$，函数的操作符为函数名 u，操作数为函数的变量 x,t，$u(x,t)$ 的表达式树为下图中左下角以 u 为根节点的子树。

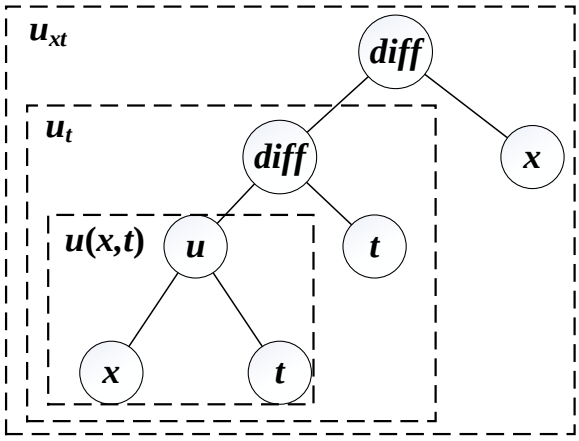

图 2 表达式树：u_{xt}

而微分计算的操作符为 diff 函数，他有两个操作数，左操作数是被求导的函数，由操作数为求导的变量。例如 $u_t = diff(u,t)$ 的表达式树为图中中间虚框所包含的子树。

Maple 的高阶微分计算是通过 diff 的层层迭代表示的，例如 $u_{xt} = diff(diff(u,x),t)$，

$u_{xxx} = diff\left(diff\left(diff\left(u,x\right),x\right)\right)$。因此，对于方程中的高阶导数项，需要层层递归才能访问，不利于导数替换的编程实现。

本文利用 Maple 对微分计算的另一种表现形式来实现行波变换。Maple 的 convert(expression, `D`) 命令可以将 expression 中的导数项替换为如下表达形式

$$\frac{\partial}{\partial x_{k_1}\partial x_{k_2}\cdots\partial x_{k_2}} f\left(x_1,x_2,...,x_m\right) = D\left[k_1,k_2,..,k_n\right]\left(f\right)\left(x_1,x_2,...,x_m\right) \tag{4.3}$$

在这种表达下，微分算子 D 是根节点，其子节点为求导顺序 $\left[k_1,k_2,..,k_n\right]$，$D\left[k_1,k_2,..,k_n\right]$ 构成了一个特定的微分算子，其参数为被求导的函数 f，$D\left[k_1,k_2,..,k_n\right]$ 作用到函数 f 上产生了求到后的函数 $D\left[k_1,k_2,..,k_n\right]\left(f\right)$，该函数的参数列表为 $\left(x_1,x_2,...,x_m\right)$。

以 u_{xxx} 为例，转换之后的表达式树变成了

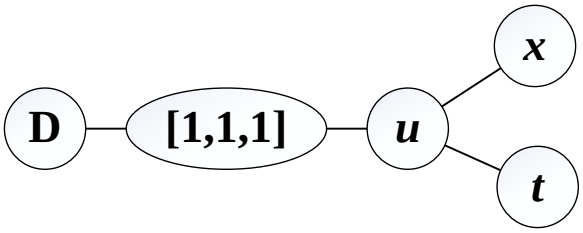

图 3 表达式树：$D\left[1,1,1\right]\left(u\right)\left(x,t\right)$

在这种表达方式下，我们可以很轻松的对导数进行替换。在本文的问题中，表达式为 $D\left[k_1,k_2,..,k_n\right]\left(u\right)\left(x,t\right)$，我们只需将 $\left(x,t\right)$ 替换为 $\left(\xi\right)$，就可以将 u 的变量变为 ξ。对于 $\left[k_1,k_2,..,k_n\right]$ 直接替换为 $[1,1,...,1]$ 即可，在处理过程中，如果遇到 $k_i=1$ 则表示对 x 进行求导，在将其替换为 1 的同时，把表达式乘上 k，即可表示 $\partial/\partial x \to k\cdot d/d\xi$；遇到 $k_i=2$ 则表示对 t 进行求导，在将其替换为 1 的同时，把表达式乘上 $-ck$，即可表示 $\partial/\partial t \to -ck\cdot d/d\xi$。

getODE()函数从表达式树的根节点对表达式的每个元素进行访问，如果遇到 $u\left(x,t\right)$ 则直接替换为 $u\left(\xi\right)$，如果遇到导数项，则将其进行转化之后再用上述的方式进行导数替换，最终将关于 $u\left(x,t\right)$ 的偏微分方程转化为了关于 $u\left(\xi\right)$ 的常微分方程，完成了行波变换。

4.3 确定方程各项阶数：findOrders()

进行行波变换之后的下一步是确定变换后方程中每一项的阶数。在上文中我们已经知道

$$O\left(c_i\prod_{j=1}^{D_i}\frac{d^{p_{ij}}}{d\xi^{p_{ij}}}u\left(\xi\right)\right) = \sum_{j=1}^{D_i}\left(m+p_{ij}\right) = D_i m + \sum_{j=1}^{D_i} p_{ij} = D_i m + P_i \tag{4.4}$$

要确定方程每一项的阶数，首先需要确定方程的每一项中，各导数项的导数阶数。通过对方程的每一项进行递归访问，将每一个导数项转化为微分算子的表达形式，统计求导顺序列表中的元素个数，即可确定求导阶数 p_{ij}。

最终，方程各项阶数都可以表示为 $D_i m + P_i$ 的形式，是关于 m 的线性表达式。本文将以二元组

(D_i, P_i)的形式记录方程的各项阶数，用作后续处理。

4.4 方程定阶：findInflexion()

在确定方程中各项的阶数关于 m 的表达式之后，本文设计了一种求函数 $f(m)=\max\{D_i m+P_i\}$ 拐点的算法来确定所有可能的阶数。该算法采用伪代码的形式可以表示如下：

输入：$S=\{(D_i,P_i)\}_{i=1}^{N}$

输出：拐点集合 M

$findInflexion(S)$

$M=\varnothing$

for every pair (i,j) subjects $i\neq j, 1\le i,j\le N$ do

if $D_i\neq D_j$ then

$$m=\frac{P_i-P_j}{D_j-D_i}$$

if $\forall 1\le k\le N, D_i m+P_i\ge D_k m+P_k$ then

$M=M\bigcup\{m\}$

end if

end if

end do

return M

end

该算法通过计算任意两条直线 $D_i m+P_i, D_j m+P_j$ 的交点，判断该点的函数值是否大于等于其它直线在该点的函数值来判断交点是否是 $f(m)=\max\{D_i m+P_i\}$ 的拐点。

在得到所有拐点之后，我们采用以下方法获取所有可能的平衡阶数：

step 1. 首先计算可能的变换阶数

$$TR=\{tr \mid tr=\mathrm{sgn}(m)/denom(m), m\in M\} \tag{4.5}$$

step 2. 对于每个 $tr\in TR$，对原方程做变换 $u(\xi)=[v(\xi)]^{tr}$，然后执行 step3 至 step5。

step 3. 对于变换后的方程进行化简，检查方程是否是关于 $v(\xi)$ 及其导数的多项式方程。如果不是多项式方程，则当前变换是不可行的。

step 4. 对于变换后的方程，重新计算拐点的集合 $M(tr)$，在 $M(tr)$ 中选取最大正整数作为变换 tr 下的阶数 m。

step 5. 对于每一组 (tr,m) 对应的方程，可以传递给 solveForOrder()函数进行求解，在求得解 $v(x,t)$ 之后，可以将 $u(x,t)=[v(x,t)]^{tr}$ 作为最终的解。

上述算法将正整数的阶数和非正整数的阶数进行统一处理，传递 solveForOrder()函数进行求解，可以让程序在结构上更加简洁。

本文 3.2 节的结论保证了在 step4 一定存在正整数的阶数。选取最大正整数最为最终阶数，可以涵盖所有可能的情况。

4.5 方程求解：solveForOrder()

在经过上一步阶数和变换的确定之后，solveForOrder()函数负责对指定阶数 m 的方程进行求解。

传递给 solveForOrder()函数的方程是关于 $v(\xi)$ 及其导数的多项式方程，并且阶数 m 是已知的。通过代换

$$v(\xi)=\sum_{k=0}^{m}a_k T^k=\sum_{k=0}^{m}a_k \tanh^k(\xi) \tag{4.6}$$

可以将原方程转化为关于 T 的多项式方程。然后再按照本文 3.3 节所述的方法进行求解即可。

4.6 解集输出：printSolutions()

本文的求解算法在求解过程中已经消去了所有重复的解和平凡的解，printSolutions 只负责将这些解按照特定的格式进行输出。

IRATH 的输出结果分为以下几个部分：

1) 输出方程。对输入方程进行标准化之后进行输出。
2) 变换后的方程。对于每个可以求解的阶数，输出变换后的常微分方程。
3) 输出解。在输出解时，会首先输出解的一般形式，再给出解中各参数的取值，以及解存在的条件。待定系数 $a_0,a_1,...,a_m,k,c$ 将按顺序进行输出，并用方程的参数进行表示，之后再输出方程的参数所满足的条件，即孤波解存在的条件。
4) 求解时间。最后以秒为单位输出求解当前所花费的总时间。

IRATH 的输出示例如下图所示。

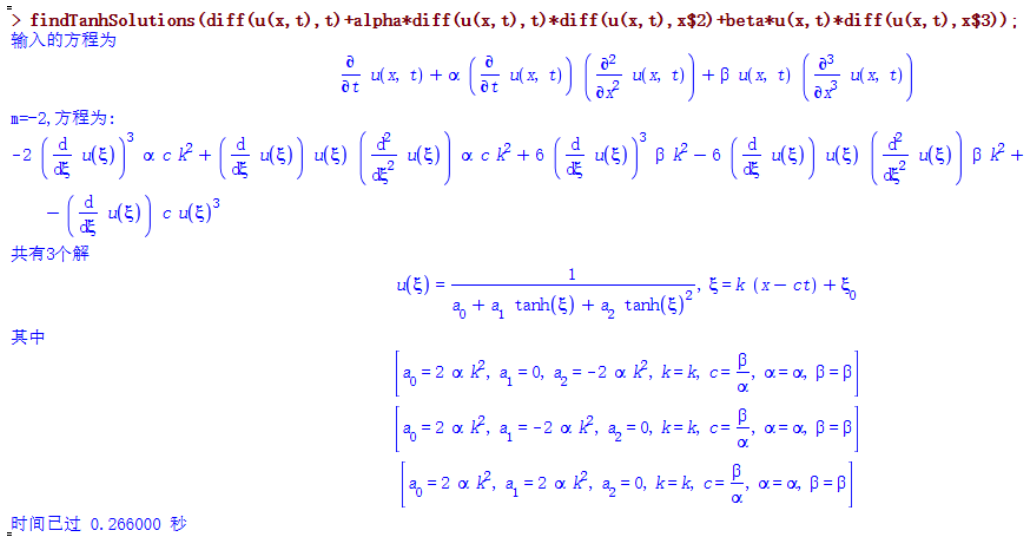

图 4 输出结果示例

第五章　改进双曲正切方法的实例分析

因为本文的是对文献[8,9]的改进，因此，本文将结合实例，用本文所实现的 IRAH 软件包和文献[8,9]实现的 RATH 软件包进行对比，验证本文算法的正确性，同时进行求解效率和求解效果的对比。在本章中，本文选取了若干个具有代表性的例子来对 IRATH 软件包的例子进行分析。

5.1 简单方程

对于方程

$$\frac{\partial}{\partial t}u\left(x,t\right)+u\left(x,t\right)\frac{\partial}{\partial x}u\left(x,t\right)+\alpha\frac{\partial^3}{\partial x^3}u\left(x,t\right)=0 \tag{5.1}$$

进行行波变换可以得到

$$\left(\frac{\mathrm{d}^3}{\mathrm{d}\xi^3}u\left(\xi\right)\right)\alpha k^2+u\left(\xi\right)\frac{\mathrm{d}}{\mathrm{d}\xi}u\left(\xi\right)-\left(\frac{\mathrm{d}}{\mathrm{d}\xi}u\left(\xi\right)\right)c=0 \tag{5.2}$$

定阶后可以确定 $m=2$。RATH 和 IRAH 都求得了

$$a_0=8\alpha k^2+c,a_1=0,a_2=-12\alpha k^2 \tag{5.3}$$

的解，k,c,α 均为自由变量。

5.2 更完整的符号取值分类

对于方程

$$\frac{\partial}{\partial t}u\left(x,t\right)+u\left(x,t\right)\frac{\partial}{\partial x}u\left(x,t\right)-p\frac{\partial^2}{\partial x^2}u\left(x,t\right)+q\frac{\partial^3}{\partial x^3}u\left(x,t\right)=0 \tag{5.4}$$

可以得到形如

$$u\left(\xi\right)=a_0+a_1\tanh\left(\xi\right)+a_2\left(\tanh\left(\xi\right)\right)^2,\xi=k\left(x-ct\right)+\xi_0 \tag{5.5}$$

的解。

RATH 得出了 1 个解

$$k=-\frac{1}{10}\frac{p}{q},a_0=\frac{1}{25}\frac{3p^2+25qw_1}{q},a_1=\frac{6p^2}{25q},a_2=-\frac{3p^2}{25q} \tag{5.6}$$

其中 c,p,q 为自由变量。

IRATH 得出了 2 个解

$$a_0=c+\frac{3p^2}{25q},a_1=-\frac{6p^2}{25q},a_2=-\frac{3p^2}{25q},k=\frac{1}{10}\frac{p}{q},c=c,p=p,q=q \tag{5.7}$$

$$a_0=c+\frac{3p^2}{25q},a_1=\frac{6p^2}{25q},a_2=-\frac{3p^2}{25q},k=-\frac{1}{10}\frac{p}{q},c=c,p=p,q=q \tag{5.8}$$

其中解(5.8)和 RATH 得到的解是同一个，而解(5.7)相当于和解(5.8)只相差了一个符号。当$\xi = k\left(x - ct\right)$时，只需将$k$乘上-1，就能得到从解(5.8)变为解(5.7)，此时解(5.7)和(5.8)是同一个解。但是，在$\xi = k\left(x - ct\right) + \xi_0$时，解(5.7)和(5.8)是不同的两个解。

在这个方程中，IRATH 的解比 RATH 更丰富的原因在于，IRATH 基于 Maple 自带 solve 对解的最终表达式进行求解，相比于 RATH，对符号的分类讨论更加完整。

5.3 更完整的阶数取值

对于方程

$$\frac{\partial}{\partial t}u\left(x,t\right) + \alpha\left(\frac{\partial}{\partial t}u\left(x,t\right)\right)\frac{\partial^2}{\partial x^2}u\left(x,t\right) + \beta u\left(x,t\right)\frac{\partial^3}{\partial x^3}u\left(x,t\right) = 0 \tag{5.9}$$

进行行波变换后得到

$$-\left(\frac{\mathrm{d}}{\mathrm{d}\xi}u\left(\xi\right)\right)\left(\frac{\mathrm{d}^2}{\mathrm{d}\xi^2}u\left(\xi\right)\right)\alpha ck^2 + u\left(\xi\right)\left(\frac{\mathrm{d}^3}{\mathrm{d}\xi^3}u\left(\xi\right)\right)\beta k^2 - \left(\frac{\mathrm{d}}{\mathrm{d}\xi}u\left(\xi\right)\right)c = 0 \tag{5.10}$$

求得阶数$m = -2$，进行变换$u\left(\xi\right) = \left[v\left(\xi\right)\right]^{-1}$后，新的方程为

$$\begin{aligned}&\left(\frac{d^2}{d\xi^2}v\left(\xi\right)\right)v\left(\xi\right)\left(\frac{d}{d\xi}v\left(\xi\right)\right)\alpha ck^2 - 2\left(\frac{d}{d\xi}v\left(\xi\right)\right)^3\alpha ck^2\\&-6\left(\frac{d^2}{d\xi^2}v\left(\xi\right)\right)v\left(\xi\right)\left(\frac{d}{d\xi}v\left(\xi\right)\right)\beta k^2 + \left(v\left(\xi\right)\right)^2\left(\frac{d^3}{d\xi^3}v\left(\xi\right)\right)\beta k^2\\&+6\left(\frac{d}{d\xi}v\left(\xi\right)\right)^3\beta k^2 - \left(\frac{d}{d\xi}v\left(\xi\right)\right)c\left(v\left(\xi\right)\right)^3 = 0\end{aligned} \tag{5.11}$$

求$\max\left\{3m+3, 3m+3, 3m+3, 3m+3, 3m+3, 4m+1\right\}$，RATH 只能得到$m = 2$，而 IRATH 能够得到$m = 1, 2$。

因此，RATH 只给出了 1 个解

$$a_0 = 2\alpha k^2, a_1 = 0, a_2 = -2\alpha k^2, c = \frac{\beta}{\alpha} \tag{5.12}$$

而 IRATH 能够给出 3 个解

$$a_0 = 2\alpha k^2, a_1 = 0, a_2 = -2\alpha k^2, k = k, c = \frac{\beta}{\alpha}, \alpha = \alpha, \beta = \beta \tag{5.13}$$

$$a_0 = 2\alpha k^2, a_1 = -2\alpha k^2, a_2 = 0, k = k, c = \frac{\beta}{\alpha}, \alpha = \alpha, \beta = \beta \tag{5.14}$$

$$a_0 = 2\alpha k^2, a_1 = 2\alpha k^2, a_2 = 0, k = k, c = \frac{\beta}{\alpha}, \alpha = \alpha, \beta = \beta \tag{5.15}$$

其中解(5.13)和解(5.12)相同，都是$m = 2$的解，并且多了(5.14)和(5.15)两个$m = 1$的解。

5.4 更一般的解、更简洁的表达式、更快的求解速度

对于方程

$$\begin{aligned}&\frac{\partial}{\partial t}u(x,t)+u(x,t)\frac{\partial^3}{\partial x^3}u(x,t)+p\left(\frac{\partial}{\partial x}u(x,t)\right)\frac{\partial^2}{\partial x^2}u(x,t)\\&+q\left(u(x,t)\right)^2\frac{\partial}{\partial x}u(x,t)+\frac{\partial^5}{\partial x^5}u(x,t)=0\end{aligned}\tag{5.16}$$

RATH 给出了 10 个解，这些解大多对参数 p,q 的取值都存在着约束，且表示式十分复杂。

其中没有参数约束的解有 2 个

$$\begin{aligned}a_0=&\frac{2k^2\left(p+2\pm\sqrt{p^2+4p-40q+4}\right)}{q},a_1=0,\\a_2=&\frac{3k^2\left(p+2\pm\sqrt{p^2+4p-40q+4}\right)}{q},\\c=&\frac{2k^4\left(p^2+2p\text{-}12q\pm p\sqrt{p^2+4p-40q+4}\right)}{q}\end{aligned}\tag{5.17}$$

其中 3 个 $\pm$ 符号同时取 $+$ 或者同时取 $-$。

而对于其中形式最为复杂的解，因为公式过于复杂，所以只给出 a_0 的解如下：

$$\begin{aligned}a_0=&\frac{846317429k^2\left(6109012+585\sqrt{13}\sqrt{644689}\right)^{4/3}\sqrt{13}\sqrt{644689}}{543267850109468129100}\\&-\frac{1463485097741k^2\left(6109012+585\sqrt{13}\sqrt{644689}\right)^{4/3}}{125369503871415722100}\\&-\frac{15399763k^2\sqrt[3]{6109012+585\sqrt{13}\sqrt{644689}}}{614987100}\\&+\frac{385k^2\left(6109012+585\sqrt{13}\sqrt{644689}\right)^{2/3}\sqrt{13}\sqrt{644689}}{488670702}\\&-\frac{26801k^2\sqrt[3]{6109012+585\sqrt{13}\sqrt{644689}}\sqrt{13}\sqrt{644689}}{204995700}\\&-\frac{1619701k^2\left(6109012+585\sqrt{13}\sqrt{644689}\right)^{2/3}}{1466012106}+\frac{577k^2}{9}\end{aligned}\tag{5.18}$$

RATH 给出的 10 个解中，8 个解的参数 p,q 都只在 p,q 是一个常数时成立，不具有一般性。

而 IRATH 只给出了 4 个解，且参数 p,q 均为自由变量。

$$
\begin{aligned}
&a_0=-\frac{2}{3}a_2, a_1=0, a_2=a_2, k=\pm\frac{\sqrt{30}\sqrt{-pa_2-2a_2\pm a_2\sqrt{p^2+4p-40q+4}}}{60},\\
&c=-\frac{\sqrt{p^2+4p-40q+4}\,pa_2{}^2}{450}+\frac{p^2a_2{}^2}{450}+\frac{\sqrt{p^2+4p-40q+4}a_2{}^2}{150}\\
&-\frac{pa_2{}^2}{450}+\frac{1}{15}qa_2{}^2-\frac{a_2{}^2}{75}, p=p, q=q
\end{aligned}
\tag{5.19}
$$

其中 k 的两个 ± 符号共有 4 种可能的取值。

并且 RATH 求解这个方程耗时 5.625 秒，而 IRATH 只用了 0.375 秒。

对于更为复杂的方程

$$
\begin{aligned}
&\frac{\partial}{\partial t}u(x,t)+2bu(x,t)\frac{\partial}{\partial x}u(x,t)+3d\left(u(x,t)\right)^2\frac{\partial}{\partial x}u(x,t)\\
&+2p\left(\frac{\partial}{\partial x}u(x,t)\right)\frac{\partial^2}{\partial x^2}u(x,t)+q\frac{\partial^3}{\partial x^3}u(x,t)\\
&+r\left(\frac{\partial}{\partial x}u(x,t)\right)\frac{\partial^2}{\partial x^2}u(x,t)+ru(x,t)\frac{\partial^3}{\partial x^3}u(x,t)+s\frac{\partial^5}{\partial x^5}u(x,t)=0
\end{aligned}
\tag{5.20}
$$

RATH 耗时 16.219 秒，而 IRATH 仅用了 3.563 秒。IRATH 相比于 RATH 在求解大型方程时更具优势。

5.5 小结

相比于 RATH，IRATH 具有以下优点：

1) 因为 IRATH 在求解时利用 Maple 自带求解函数来进行解的转换，在讨论解的取值时，对符号的分类更加完整。
2) 因为本文算法对阶数的取值进行了详细分析，因此 IRATH 能够得出解的阶数更完整。
3) 因为本文算法在求解时伴随着化简，因此 IRATH 得出的解形式更简单，求解速度更快。
4) 因为本文算法对变量的求解顺序更加合理，因此 IRATH 得出的解更具一般性。

第六章　总结与结论

双曲正切函数方法是求解非线性演化方程孤波解的有效方法。本文在总结前人研究的基础上，分析了传统双曲正切方法的不足之处，并对其进行了改进。

在理论上，本文的算法具有以下优势：

1) 首次提出了行波变换后常微分方程的一般形式。在以往的文献中，对于行波变换后的常微分方程，只是笼统的架势它是关于$u(\xi)$及其导数的多项式方程。而本文给出了该方程的一般形式，并基于该方程，在严格的数学推导的基础上，对阶数平衡条件以及方程阶数变换进行了分析。
2) 对阶数平衡的条件的考虑更为全面。传统的双曲正切函数方法只考虑平衡线性最高阶项和非线性最高阶项的阶数，而本文的算法考虑了所有非齐次项进行平衡的情况。另外，本文所处理的阶数平衡条件还涵盖了一部分同阶最高项相平衡的情况。
3) 在处理非正整数的阶数时，本文对文献中的方程变换方法进行了分析，并首次证明了其可行性。同时，还分析了变换成立的条件。

基于改进的双曲正切函数方法，本文在计算机代数系统 Maple 上，编写了软件包 IRATH 来实现该算法。同时，我们将 IRATH 和文献[8,9]中的 RATH 软件包进行了对比，分析其正确性和效率。最终得出 IRATH 具有以下优点：

1) 由于理论上的优势，IRATH 在求解一些方程时，能够发现比 RATH 更多的解。
2) IRATH 的求解算法在求解某些方程时，能够得到形式更简单的解，或者能够得到形式更一般的解。
3) 在求解复杂方程时，IRATH 的求解速度更快。

同时，本文的算法还存在以下不足之处：

1) 如本文 3.2 节所述，本文在考虑阶数平衡调节时，没有考虑无穷多组平衡阶数的处理方法。
2) 本文只实现了非线性演化方程的求解方法，并没有实现非线性演化方程组的求解方法。

本文应用 IRATH 对大量非线性演化方程进行了求解。IRATH 不仅给出了其它方法得到的双曲正切多项式的解，更为重要的是，对于一些方程，IRATH 还得到了新的解，得到了形式更简单和形式更一般的解。并且对于大多数方程的求解，IRATH 都能在 5 秒内完成。因此，IRATH 是求解非线性演化方程双曲正切多项式孤波解的有力工具。

参考文献

[1] Dodd R K, Eilbeck J C, Gibbon J D, et al. Solitons and nonlinear wave equations[J]. 1982.

[2] Ablowitz M J, Clarkson P A. Solitons, nonlinear evolution equations and inverse scattering[M]. Cambridge university press, 1991.

[3] 谷超豪. 孤子理论及其应用[J]. 1990.

[4] 郭柏林, 庞小峰. 孤立子[J]. 1987.

[5] Malfliet W. Solitary wave solutions of nonlinear wave equations[J]. American Journal of Physics, 1992, 60(7): 650-654.

[6] Parkes E J, Zhu Z, Duffy B R, et al. Sech-polynomial travelling solitary-wave solutions of odd-order generalized KdV equations[J]. Physics Letters A, 1998, 248(2): 219-224.

[7] 李志斌, 张善卿. 非线性波方程准确孤立波解的符号计算[J]. 数学物理学报, 1997, 17(1): 81-89.

[8] 柳银萍. 基于吴方法的孤波自动求解软件包及其应用[D].华东师范大学,2001.

[9] Li Z, Liu Y. RATH: a Maple package for finding travelling solitary wave solutions to nonlinear evolution equations[J]. Computer Physics Communications, 2002, 148(2): 256-266.

[10] Johnson W P. The curious history of Faà di Bruno's formula[J]. The American mathematical monthly, 2002, 109(3): 217-234.

[11] Wang D. An implementation of the characteristic set method in Maple[M]. Springer Vienna, 1995.

致　谢

在完成学位论文之际，我首先要感谢我的导师周明华教授，本文的工作正是在他的悉心指导下完成的。周明华教授不仅在学术研究上对我给予了很多指导，更是在人生的发展方向上给了我很多建议。同时我还要感谢华东师范大学的柳银萍老师，本文的工作是在她之前工作的基础上进行的。柳银萍老师在本文的写作过程中，耐心地为我解答了许多问题，给了我许多专业的指导和建议。

附录 A IRATH 实现代码

```
IRATH:=module()
    option    package;
    export    findTanhSolutions;
    local     nomalizeEqn,
              getODE,
              subsDiff,
              subsElem,
              subsEqn,
              simplifyEqn,
              findDiffOrder,
              findElemOrder,
              findItemOrder,
              findOrders,
              findInflexion,
              solveAllOrder,
              solveForOrder,
              selectSolutions,
              printSolutions,
              solveForPS,
              transSolution,
              subsIeq,
              subsSol,
              checkSolution,
              isEquationOf;


    (*
     * 求解函数
     *)
    findTanhSolutions:=proc(eqi)
        local eq,ms,params,oeq,st;
        st:=time();

        printf("输入的方程为");
        eq:=nomalizeEqn(eqi);
        oeq:=eq;
        print(eq);

        # printf("方程中的参数为");
        params:=indets(eq,name) minus {t,x};
```

```
    # print(params);

    # printf("行波变换后的方程为");
    eq:=getODE(eq);
    # print(eq);

    # printf("方程各项阶数为");
    ms:=findOrders(eq);
    # print(ms);

    # printf("拐点为");
    ms:=findInflexion(ms);
    # print(ms);

    if evalb(ms={0}) then
        printf("无解\n");
        printf("时间已过 %f 秒\n",time()-st);
        return;
    end if;

    # printf("开始求解\n");
    solveAllOrder(eq,ms,params,oeq);

    printf("时间已过 %f 秒\n",time()-st);
end proc:

(*
 * 标准化方程
 * 方程变量是 u(x,t)
*)
nomalizeEqn:=proc(ee)
    local e;
    e:=expand(numer(ee));
    return e;
end proc:

(*
 * 进行行波变换，将偏微分方程转化为常微分方程
 * 要求函数为 u(x,t),替换为 u(xi),
 * xi=k*(x-ct)
*)
getODE:=proc(eqi)
    local T;
    T:=table();
```

```
    subsEqn(eqi,T);
    simplifyEqn(add(indices(eval(T),'nolist')));
end proc:

(*
 * 导数替换
 * diff(u(x,t),t) -> -c*k*diff(u(xi),xi)
 * diff(u(x,t),x) -> k*diff(u(xi),xi)
*)
subsDiff:=proc(ee)
    local e,vars,n,v,p;
    e:=convert(ee,'D');
    vars:=[op(op(0,op(0,e)))];
    n:=numelems(vars);
    p:=1;
    for v in vars do
        if evalb(v=1) then
            p:=p*k;
        else
            p:=p*(-c*k);
        end if;
    end do;
    return p*diff(u(xi),xi$n);
end proc:

(*
 * 元素替换
*)
subsElem:=proc(ee)
    if type(ee,`^`) then
        subsElem(op(1,ee))^op(2,ee);
    elif evalb(op(0,ee)='diff') then
        subsDiff(ee);
    elif evalb(op(0,ee)='u') then
        subs(u(x,t)=u(xi),ee);
    else
        ee;
    end if;
end proc:

(*
 * 方程替换
*)
subsEqn:=proc(ee,T)
    local e,p;
```

```
    if type(ee,`+`) then
        for e in ee do
            subsEqn(e,T);
        end do;
    elif type(ee,`*`) then
        p:=1;
        for e in ee do
            p:=p*subsElem(e);
        end do;
        T[p]:=1;
    else
        T[subsElem(ee)]:=1;
    end if;
end proc:

(*
 * 化简方程
*)
simplifyEqn:=proc(ee)
    local e,v,p;
    e:=numer(ee);
    e:=factor(e);
    if type(e,`*`) then
        p:=1;
        for v in e do
            if type(v,`+`) then
                p:=p*v;
             elif type(v,`^`) and type(expand(op(1,v)),`+`) then
                p:=p*(simplifyEqn(op(1,v))^op(2,v));
            end if;
        end do;
        p:=expand(p);
    else
        p:=expand(e);
    end if;
    return p;
end proc:

(*
 * 计算导数项阶数
*)
findDiffOrder:=proc(ee)
    local e;
    e:=convert(ee,'D');
    e:=op(0,op(0,e));
```

```
    if evalb(op(0,e)=`@@`) then
        op(2,e);
    else
        1;
    end if;
end proc:

(*
 * 计算元素的阶数
*)
findElemOrder:=proc(ee)
    local e;
    if evalb(op(0,ee)='u') then
        e:=[1,0];
    elif evalb(op(0,ee)='diff') then
        e:=[1,findDiffOrder(ee)];
    elif type(ee,`^`) then
        e:=findElemOrder(op(1,ee))*op(2,ee);
    else
        e:=[0,0];
    end if;
    return e;
end proc:

(*
 * 计算方程某一项的阶数
*)
findItemOrder:=proc(ee)
    local e,p;
    if type(ee,`*`) then
        p:=[0,0];
        for e in ee do
            p:=p+findElemOrder(e);
        end do;
    else
        p:=findElemOrder(ee);
    end if;
    return p;
end proc:

(*
 * 计算方程每一项的阶数
 * 假设至少有两项
 * 假设指数上不含参数
*)
```

```
findOrders:=proc(ee)
    findItemOrder~([op(ee)]);
end proc:

(*
 * 计算拐点
*)
findInflexion:=proc(mss::list(list(integer)))
    local ms,i,j,n,mm,m,fun;
    fun:=(x,m)->x[1]*m+x[2];
    mm:={};
    ms:={op(mss)};
    n:=numelems(ms);
    for i from 1 to n-1 do
        for j from i+1 to n do
            if evalb(ms[i][1]<>ms[j][1]) then
                m:=-(ms[i][2]-ms[j][2])/(ms[i][1]-ms[j][1]);
                if evalb(fun(ms[i],m)=max(map(fun,ms,m))) then
                    mm:=mm union {m};
                end if;
            end if;
        end do;
    end do;
    return mm;
end proc:

(*
 * 对所有阶数进行求解
*)
solveAllOrder:=proc(eqi,mms::set(rational),params::set(name),oeq)
    local trans,tr,eq,ms,m;
    # 计算全部变换
    trans:=map(x->sign(x)/denom(x),mms);
    for tr in trans do
        eq:=eval(subs(u(xi)=u(xi)^tr,eqi));
        eq:=simplifyEqn(eq);
        ms:=findOrders(eq);
        m:=select(type,findInflexion(ms),posint);
        m:=max(m);
        printf("m=%a,方程为:",tr*m);
        print(eq);
        if not type(ms,list(list(integer))) then
            printf("方程不是多项式，无解\n");
            next;
        end if;
```

```
            solveForOrder(eq,m,params,tr,oeq);
        end do;
end proc:

(*
 * 对特定阶数进行求解
*)
solveForOrder:=proc(eqi,m::posint,params::set(name),tr::rational,oeq)
        local eq,f,PS,vars,sols;
        f:=add(seq(a[i]*tanh(xi)^i,i=0..m));
        vars:=[seq(a[i],i=0..m)];
        eq:=eval(subs(u(xi)=f,eqi));
        eq:=subs(tanh(xi)=T,eq);
        eq:=collect(eq,T);
        PS:=PolynomialTools[CoefficientList](eq,T,termorder=reverse);
        sols:=solveForPS(PS,m,params,vars);
        printSolutions(sols,m,tr);
end proc:

(*
 * 筛选解
 * 删去 k,c=0 的解
 * 删除参数为 0 的解
 * 删除 a[1]..a[m]都为 0 的解
*)
selectSolutions:=proc(solss::{set,list},
                                params::set(name),m::posint)
        local zs,ps,sols,as;
        as:={seq(a[i],i=1..m)};
        sols:=convert(solss,set);
        sols:=select(type,sols,equation);
        ps:=params union {k,c};
        zs:=lhs~(select(_x->type(_x,`=`(name,0)),sols));
        if evalb( (zs intersect ps) <> {} ) then
            return false;
        else
            return not evalb(as subset zs);
        end if;
end proc:

(*
 * 输出解
*)
printSolutions:=proc(sols::set({list,set}),
```

```
                        m::posint,tr::rational)
    local sol;
    if evalb(sols={}) then
        printf("无解\n");
        return;
    end if;
    printf("共有%d 个解",numelems(sols));
    print(u(xi)=add(seq(a[i]*tanh(xi)^i,i=0..m))^tr,xi=k*(x-ct)+xi[0]);
    printf("其中");
    for sol in sols do
        print(expand(simplify(sol)));
    end do;
end proc:

(*
 * 求解 PS
 * 基于 csolve 的版本
 * 只有剩余方程不能用 csolve 求解时，才使用 solve 求解
*)
solveForPS:=proc(PS,m::posint,params::set(name),vars::list(name))
    local pa,pp,sola,sol,vs,res,np,nsol,ssol,nres,sssol;
    res:={};# 所有解
    pa:=PS[1..(m+1)];
    pp:=PS[(m+1)..-1];
    sola:=[csolve(pa,{vars[],k,c})];# 求解前 m+1 个方程
    sola:=select(selectSolutions,sola,params,m);# 除去平凡的解
    for sol in sola do
        # 代入化简
        np:=simplify(subs(op(sol),pp));
        np:=simplifyEqn~(np);# 化简方程，去除非零项
        np:=remove(type,np,0);

        # 如果剩余的方程全 0，则不需要继续求解
        if evalb(np=[]) then
            res:=res union {sol};
            next;
        end if;

        # 求解剩余方程
        try
            nsol:=[csolve(np,indets(np,name))];
        catch:
            nsol:=[RealDomain[solve](np,indets(np,name))];
```

```
            end try;

            nsol:=select(selectSolutions,nsol,params,m);

            for ssol in nsol do
                # 合并解，因为前面的解，在后面已经被带入
                # 所以，后面的解不会包含前面已有的符号
                # 因此，直接合并即可
                try # 有可能出现分母为 0 的情况
                    sssol:=simplify(subs(op(ssol),sol));
                    res:=res union {sssol union ssol};
                catch:
                    next;
                end try;
            end do;
        end do;

        res:=select(selectSolutions,res,params,m);

        # printf("未转换的解");
        # print~(res);

        nres:={};
        map(transSolution,res,'nres',params,vars);
        res:=select(selectSolutions,nres,params,m);

        return res;
    end proc:



    (*
     * 转化 csolve 的结果
    *)
    transSolution:=proc(res::set(equation),nres::evaln(set),
                        params::set(name),vars::list(name))
        local r,rv,rp,np,rrv,rrp,sp,newSol;
        # 先求解 a[0]..a[m],k,c
        sp:=select(isEquationOf,res,{vars[],k,c});
        sp:=map(_x->simplifyEqn(lhs(_x)-rhs(_x)),sp);# 化简方程，去除非零项
        rv:=RealDomain[solve](sp,[vars[],k,c]);

        # 如果求解失败则直接原样返回
        if evalb(rv=[]) then
```

```
            nres:=eval(nres) union {[res[]]};
            WARNING("solve 求解失败");
            return;
        end if;

        # 再求解参数
        for rrv in rv do
            try
                np:=simplify(subs(op(rrv),res));
            catch:
                next;
            end try;
            np:=map(_x->simplifyEqn(lhs(_x)-rhs(_x)),np);# 化简方程，去除非零项
            np:=select(isEquationOf,np,params);
            rp:=RealDomain[solve](np,[params[]]);
            for rrp in rp do
                rrp:=subsIeq~(rrp);
                try
                    newSol:=[op(subsSol(rrv,rrp)),rrp[]];
                    newSol:=simplify(rationalize(newSol));
                    nres:=eval(nres) union {newSol};
                catch:
                    next;
                end try;
            end do;
        end do;
        return;
    end proc:

    (*
     * 消去不等式范围约束
    *)
    subsIeq:=proc(ee)
        local v;
        if type(ee,equation) then
            return ee;
        else
            v:=indets(ee);
            if evalb(nops(v)=1) then
                v:=v[1];
                return (v=v);
            else
                return ee;
            end if;
        end if;
```

```
    end proc:

    (*
     * 解集带入
     * sol2 代入 sol1
    *)
    subsSol:=proc(sol1,sol2)
        local sol;
        sol:=select(type,sol2,equation);
        expand(subs(op(sol),sol1));
    end proc:

    (*
     * 代回原方程进行验证
    *)
    checkSolution:=proc(ssol,oeq,m,tr)
        local uxi,sol,r;
        sol:=select(type,ssol,equation);
        r:=subs(u(x,t)=add(seq(a[i]*tanh(xi)^i,i=0..m))^tr,oeq);
        r:=subs(xi=k*(x-c*t),r);
        r:=subs(op(sol),r);
        r:=simplify(r);
        evalb(r=0);
    end proc:

    (*
     * 选择关于给定参数的方程
    *)
    isEquationOf:=proc(eq,vars::set)
        local vs;
        vs:=indets(eq,name);
        evalb( (vs intersect vars) <> {} );
    end proc:

end module:
```

附录 B IRATH 和 RATH 的对比代码

```
restart;
with(charsets);
with(IRATH);
with(RATH1);

cmp:=proc(eq)
    printf("RATH\n");
    main1([eq=0]);
    printf("IRATH\n");
    findTanhSolutions(eq);
end proc:

cmp(diff(u(x,t),t)+u(x,t)*diff(u(x,t),x)+alpha*diff(u(x,t),x$3)):
cmp(diff(u(x,t),t)+u(x,t)*diff(u(x,t),x)-p*diff(u(x,t),x$2)+q*diff(u(x,t),x$3));
cmp(diff(u(x,t),t$2)+diff(u(x,t),x$2)+diff(u(x,t),x$4)-p*u(x,t)^3);
cmp(diff(u(x,t),t)+alpha*diff(u(x,t),t)*diff(u(x,t),x$2)+beta*u(x,t)*diff(u(x,t),x$3));
cmp(diff(u(x,t),t)+p*u(x,t)*diff(u(x,t),x)-diff(u(x,t),x$2)-q*u(x,t)*(1-u(x,t))*(u(x,t)-r));
cmp(diff(u(x,t),t)+2*alpha*diff(u(x,t),x)+3*u(x,t)*diff(u(x,t),x)+diff(u(x,t),x$2,t)+2*diff(u(x,t),x)*diff(u(x,t),x
$2)+u(x,t)*diff(u(x,t),x$3));
cmp(diff(u(x,t),t)+u(x,t)*diff(u(x,t),x)+diff(u(x,t),x$2)+p*diff(u(x,t),x$3)+diff(u(x,t),x$4)+q*diff((u(x,t)*diff(
u(x,t),x)),x));
cmp(diff(u(x,t),t)+diff(u(x,t),x)+u(x,t)*diff(u(x,t),x)+alpha*diff(u(x,t),x,x,t));
cmp(diff(u(x,t),t)+u(x,t)*diff(u(x,t),x$3)+p*diff(u(x,t),x)*diff(u(x,t),x$2)+q*u(x,t)^2*diff(u(x,t),x)+diff(u(x,t),
x$5));
cmp(diff(u(x,t),t)+diff((b*u(x,t)^2+d*u(x,t)^3+p*diff(u(x,t),x)^2+q*diff(u(x,t),x$2)+r*u(x,t)*diff(u(x,t),x$2)+
s*diff(u(x,t),x$4)),x));
cmp(alpha*u(x,t)*diff(u(x,t),x$3)-alpha*diff(u(x,t),x)*diff(u(x,t),x,t)+beta*u(x,t)^2*diff(u(x,t),x))
cmp(lambda*u(x,t)*diff(u(x,t),x$3)-alpha*diff(u(x,t),x)*diff(u(x,t),x,t)+beta*u(x,t)^2*diff(u(x,t),x))
```