doi: 10.12068/j.issn.1005-3026.2018.04.030

# 分数阶微积分的高精度递推算法

白 鹭[1,2], 薛定宇[1]
(1. 东北大学 信息科学与工程学院, 辽宁 沈阳 110819; 2. 沈阳大学 信息工程学院, 辽宁 沈阳 110044)

**摘 要**: 设计了一种计算分数阶微积分的高精度数值算法,提出了一种构造生成函数的简便方法. 分析了基于快速 Fourier 变换的算法,该算法误差较大的原因是应用了不准确的生成函数的系数,而且没有考虑原函数的非零初值条件对计算精度的影响. 新算法应用递推公式计算生成函数的系数,并将原函数分解成零初值条件和非零初值条件两部分,分别计算它们的分数阶微分和积分,这样可以减小计算误差. 误差分析和计算实例证明新算法具有很高的计算精度.

**关 键 词**: 分数阶;微积分;生成函数;高精度;递推算法
**中图分类号**: O 241.4 **文献标志码**: A **文章编号**: 1005-3026(2018)04-0604-05

# High Precision Recursive Algorithm for Computing Fractional-Order Derivative and Integral

*BAI Lu*[1,2], *XUE Ding-yu*[1]
(1. School of Information Science & Engineering, Northeastern University, Shenyang 110819, China; 2. School of Information Engineering, Shenyang University, Shenyang 110044, China. Corresponding author: XUE Ding-yu, professor, E-mail: xuedingyu@mail.neu.edu.cn)

**Abstract**: A high precision numerical algorithm was designed to compute fractional-order derivative and integral, and a simple method was proposed to construct the generating function. An algorithm based on fast Fourier transform was analyzed. It could be concluded that the reasons of its large computation error were using the inaccurate coefficient of the generating function and no considering the effect of nonzero initial condition of the original function on calculation precision. The recursive formula was used to compute the coefficient of the generating function in the new algorithm, what's more, the original function was decomposed into two parts, i.e., zero initial condition and nonzero initial condition, and their fractional-order derivative and integral were computed to decrease the computation error. The error analysis and the illustrative numerical examples showed that the computation accuracy of the new algorithm was very high.

**Key words**: fractional-order; derivative and integral; generating function; high-precision; recursive algorithm

分数阶微积分可以简洁地描述具有历史记忆的物理过程,被广泛应用在各工程领域中,例如在控制理论中出现的一些新算法[1],为描述和控制复杂系统提供了方便. 这些系统一般被描述成分数阶微分方程,并且出现了许多求解这种方程的数值算法[2-5]. 为了求解这些方程,首先要解决的问题是如何计算分数阶微分和积分,因此计算分数阶微积分的数值算法受到越来越多的关注,常用的算法有分数阶线性多步法[6]及基于快速 Fourier 变换(FFT)的算法[7]. 当原函数的初值条件不为零时,上述算法均不能得到高精度的结果. 文献[8]提出了一种高精度的数值算法,但此算法只能计算 Caputo 分数阶微分,而且微分阶次只能在(0,1)区间内. 本文提出一种计算分数阶微分和积分的高精度数值算法,该算法可以计算非零初值条件下任意阶次的分数阶微分和积分.

**收稿日期**: 2016-11-06
**基金项目**: 国家自然科学基金资助项目(61174145,61673094).
**作者简介**: 白 鹭(1982-),男,辽宁沈阳人,东北大学博士研究生; 薛定宇(1963-),男,辽宁沈阳人,东北大学教授,博士生导师.

## 1　生成函数

在分数阶微积分理论中,有多种分数阶微积分定义,文献[7]给出了这些定义的具体形式. 当原函数的初值条件为零时,RL 定义、GL 定义和 Caputo 定义等价,并可以应用式(1)近似计算[10].

$$ {}_0\mathscr{D}_t^{\alpha} y(t) \approx \frac{1}{h^{\alpha}} \sum_{k=0}^{[t/h]} w_k\, y(t-kh) . \tag{1} $$

其中:$\mathscr{D}$ 是分数阶微分或积分算子;$\alpha$ 是分数阶微分或积分的阶次;0 和 $t$ 分别是积分区间的下限和上限;$y(t)$ 是关于变量 $t$ 的函数;$h$ 是算法的步长;$k$ 是离散点的下标;$w$ 是生成函数的 Taylor 展开式的系数. 生成函数表达式为

$$ G_p^{\alpha}(z) = \left( \sum_{k=1}^{p} \frac{1}{k} (1-z)^k \right)^{\alpha} . \tag{2} $$

其中:$p$ 是生成函数的阶数. 如果原函数的初值条件为零,式(1)的计算精度为 $O(h^p)$[9]. 下面提出一种构造生成函数的新方法.

**定理 1**　当式(2)中的 $\alpha=1$ 时,生成函数为

$$ G_p(z) = \sum_{i=0}^{p} r_i z^i . \tag{3} $$

其中,系数 $r_i$ 满足

$$ \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & p+1 \\ 1 & 2^2 & 3^2 & \cdots & (p+1)^2 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 2^p & 3^p & \cdots & (p+1)^p \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ \vdots \\ r_p \end{bmatrix} = - \begin{bmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ p \end{bmatrix} . \tag{4} $$

证明　由式(2)和式(3)得

$$ \sum_{i=0}^{p} r_i z^i = \sum_{k=1}^{p} \frac{1}{k} (1-z)^k . \tag{5} $$

令式(5)中的 $z=1$,得到式(4)中的第一个方程:

$$ \sum_{i=0}^{p} r_i = 0 . \tag{6} $$

用 $z$ 乘以式(5)的两边,再求关于 $z$ 的一次微分,得

$$ \sum_{i=0}^{p} (i+1) r_i z^i = \sum_{k=1}^{p} \frac{1}{k} (1-z)^k - z \sum_{k=2}^{p} (1-z)^{k-1} - z . \tag{7} $$

令式(7)中的 $z=1$,得到式(4)中的第二个方程:

$$ \sum_{i=0}^{p} (i+1) r_i = -1 . \tag{8} $$

用 $z$ 乘以式(7)的两边,再求关于 $z$ 的一次微分,得

$$ \sum_{i=0}^{p} (i+1)^2 r_i z^i = \sum_{k=1}^{p} \frac{1}{k} (1-z)^k - 3z \sum_{k=2}^{p} (1-z)^{k-1} + z^2 \sum_{k=3}^{p} \frac{1}{k-1} (1-z)^{k-2} - 3z + z^2 . \tag{9} $$

令式(9)中的 $z=1$,得到式(4)中的第三个方程:

$$ \sum_{i=0}^{p} (i+1)^2 r_i = -2 . \tag{10} $$

重复以上过程可得矩阵方程(4),定理得证.

当计算一阶微分时,可以直接应用定理 1 构造生成函数. 当微分(积分)的阶次 $\alpha \neq 1$ 时,首先应用定理 1 构造 $\alpha=1$ 的生成函数,然后计算此生成函数的 $\alpha$ 次幂,这样可以得到计算 $\alpha$ 阶微分或积分的生成函数.

## 2　基于 FFT 算法的局限性

由于式(1)中的 $w$ 是生成函数的 Taylor 系数,所以可以计算生成函数在原点的 Taylor 展开式,求出对应的 Taylor 系数,然后应用式(1)计算分数阶微分或积分,显然,这种算法的效率很低. 为了提高算法的速度,文献[7]提出了一种基于 FFT 的算法,下面介绍此算法的计算过程.

将生成函数的 Taylor 展开式写成

$$ G_p^{\alpha}(z) = \sum_{k=0}^{+\infty} w_k z^k . \tag{11} $$

令式(11)中的 $z = \mathrm{e}^{-\mathrm{i}\varphi}$,则有

$$ G_p^{\alpha}(\mathrm{e}^{-\mathrm{i}\varphi}) = \sum_{k=0}^{\infty} w_k \mathrm{e}^{-\mathrm{i}k\varphi} . \tag{12} $$

如果将两个函数的内积定义为

$$ (\mathrm{e}^{\mathrm{i}n\varphi}, \mathrm{e}^{-\mathrm{i}m\varphi}) = \frac{1}{2\pi} \int_0^{2\pi} \mathrm{e}^{\mathrm{i}n\varphi} \mathrm{e}^{-\mathrm{i}m\varphi} \mathrm{d}\varphi = \begin{cases} 0, m \neq n; \\ 1, m = n . \end{cases} \tag{13} $$

则 $\mathrm{e}^{\mathrm{i}n\varphi}, \mathrm{e}^{-\mathrm{i}m\varphi}$ 是正交函数. 将下面的内积记为 $w_k$:

$$ (G_p^{\alpha}(\mathrm{e}^{-\mathrm{i}\varphi}), \mathrm{e}^{\mathrm{i}k\varphi}) = \frac{1}{2\pi} \int_0^{2\pi} G_p^{\alpha}(\mathrm{e}^{-\mathrm{i}\varphi}) \mathrm{e}^{\mathrm{i}k\varphi} \mathrm{d}\varphi = w_k . \tag{14} $$

可知 $w$ 是式(12)的 Fourier 系数,因此可用 FFT 算法计算系数 $w$,这样可以提高算法的速度. 以上的计算过程可以总结成下面的算法.

算法 1　基于 FFT 的算法

1) 选择算法的步长,计算原函数 $y(t)$ 在离散点上的函数值;

2) 应用定理 1 构造生成函数,应用 FFT 算法计算系数 $w$;

3) 应用式(1)计算分数阶微分或积分.

由于 FFT 算法是近似计算,得到的系数 $w$ 并不是准确值,因此会造成计算误差. 另外,算法 1 并没有考虑原函数的非零初值条件对计算精度的影响,非零初值条件也会造成很大的计算误差. 下面的例子将具体说明这一问题.

例 1 应用基于 FFT 的算法计算 $e^{-t}$ 的 0.6 阶 RL 微分,下面的解析解可以检验数值解的精度.

$$ {}_0^{\mathrm{RL}}\mathscr{D}_t^{0.6} e^{-t} = t^{-0.6}\varepsilon_{1,0.4}(-t). \tag{15} $$

其中,$\varepsilon(\cdot)$ 为 Mittag - Leffler 函数. 分别选择步长 $h=0.01$ 和 $h=0.001$,应用定理 1 构造 4 阶生成函数:

$$ G_4^{-0.6}(z) = (25/12 - 4z + 3z^2 - 4z^3/3 + z^4/4)^{-0.6}. \tag{16} $$

应用 FFT 算法计算式(16)的 Taylor 系数 $w$,然后应用式(1)计算 RL 微分的数值解. 将得到的数值解和解析解绘制成曲线,如图 1 所示,可见数值解和解析解相差很大. 正如之前的分析,造成误差的原因为:应用 FFT 算法计算的 $w$ 不准确;原函数 $e^{-t}$ 的非零初值条件也会造成很大的计算误差.

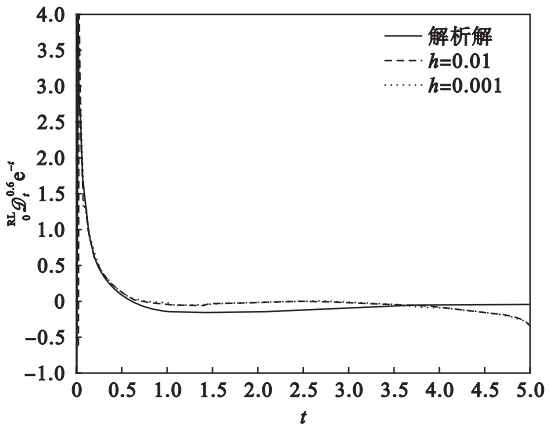


**图 1 数值解与解析解**

Fig. 1 Numerical solutions and analytical solutions

## 3 递推公式算法

为了解决系数 $w$ 不准确的问题,下面推导计算 $w$ 的递推公式.

**定理 2** 生成函数:

$$ G_p^{\alpha}(z) = \Big(\sum_{i=0}^{p} r_i z^i\Big)^{\alpha}, \tag{17} $$

其 Taylor 展开式为

$$ G_p^{\alpha}(z) = \sum_{k=0}^{\infty} w_k z^k. \tag{18} $$

当 $k=0$ 时,有 $w_0 = r_0^{\alpha}$;当 $k>0$ 时,计算 $w$ 的递推公式为

$$ w_k = -\frac{1}{r_0}\sum_{i=0}^{p} r_i\Big(1 - i\frac{1+\alpha}{k}\Big)w_{k-i}. \tag{19} $$

当 $k-i<0$ 时,有 $w_{k-i}=0$.

证明 由式(17)和式(18)可得

$$ \Big(\sum_{i=0}^{p} r_i z^i\Big)^{\alpha} = \sum_{j=0}^{\infty} w_j z^j. \tag{20} $$

令式(20)中的 $z=0$,可以得到 $w_0 = r_0^{\alpha}$.

对式(20)的两边求关于 $z$ 的一阶微分,然后乘以 $\sum_{i=0}^{p} r_i z^i$,得

$$ \alpha\sum_{i=1}^{p} i r_i z^{i-1}\cdot\Big(\sum_{i=0}^{p} r_i z^i\Big)^{\alpha} = \sum_{i=0}^{p} r_i z^i\cdot\sum_{j=1}^{\infty} j w_j z^{j-1}. \tag{21} $$

将式(20)代入式(21),得

$$ \alpha\sum_{j=0}^{\infty}\sum_{i=1}^{p} i r_i w_{j-i+1} z^j = \sum_{j=0}^{\infty}\sum_{i=0}^{p}(j-i+1) r_i w_{j-i+1} z^j. \tag{22} $$

在式(22)中,当 $j-i+1<0$ 时,有 $w_{j-i+1}=0$.

在式(22)的两边,$z^j$ 项的系数相等,所以有

$$ \alpha\sum_{i=1}^{p} i r_i w_{j-i+1} = \sum_{i=0}^{p}(j-i+1) r_i w_{j-i+1}. \tag{23} $$

将式(23)中的 $j+1$ 替换成 $k$,得

$$ \alpha\sum_{i=1}^{p} i r_i w_{k-i} = \sum_{i=0}^{p}(k-i) r_i w_{k-i}. \tag{24} $$

当 $k-i<0$ 时,有 $w_{k-i}=0$. 整理式(24),可得递推式(19),定理得证.

为了消除非零初值条件产生的计算误差,应用下面的方法将原函数转化成初值条件为零的函数. 在算法中如果选择 $p$ 阶生成函数,则构造辅助函数:

$$ u(t) = \sum_{i=0}^{p} c_i t^i. \tag{25} $$

为了使 $u(t)$ 和 $y(t)$ 在前 $p+1$ 个离散点上的函数值相等,式(25)中的系数 $c$ 需要满足

$$ \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & h & h^2 & \cdots & h^p \\ 1 & 2h & (2h)^2 & \cdots & (2h)^p \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & ph & (ph)^2 & \cdots & (ph)^p \end{bmatrix}\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_p \end{bmatrix} = \begin{bmatrix} y(0) \\ y(h) \\ y(2h) \\ \vdots \\ y(ph) \end{bmatrix}. \tag{26} $$

其中,$h$ 为算法的步长.

将 $y(t)$ 分解为 $u(t)+v(t)$,由于 $u(t)$ 和 $y(t)$ 在前 $p+1$ 个离散点上的函数值相等,可以认为 $u(t)$ 具有和 $y(t)$ 相同的初值条件,$v(t)$ 具有零初值条件,并且有

$$ {}_0^{\mathrm{RL}}\mathscr{D}_t^{\alpha} y(t) = {}_0^{\mathrm{RL}}\mathscr{D}_t^{\alpha} u(t) + {}_0^{\mathrm{RL}}\mathscr{D}_t^{\alpha} v(t), \tag{27} $$

$$ {}_0^{\mathrm{C}}\mathscr{D}_t^{\alpha} y(t) = {}_0^{\mathrm{C}}\mathscr{D}_t^{\alpha} u(t) + {}_0^{\mathrm{C}}\mathscr{D}_t^{\alpha} v(t) . \quad (28) $$

由于 $u(t)$ 是幂函数的和，可直接推导出 $u(t)$ 的分数阶微分(积分)的解析表达式：

$$ {}_0^{\mathrm{RL}}\mathscr{D}_t^{\alpha} u(t) = \sum_{i=0}^{p} \frac{c_i \Gamma(i+1)}{\Gamma(i+1-\alpha)} t^{i-\alpha} , \quad (29) $$

$$ {}_0^{\mathrm{C}}\mathscr{D}_t^{\alpha} u(t) = \sum_{i=\lceil \alpha \rceil}^{p} \frac{c_i \Gamma(i+1)}{\Gamma(i+1-\alpha)} t^{i-\alpha} . \quad (30) $$

这样就可以得到下面的递推公式算法.

算法 2　递推公式算法

1) 选择算法的步长，计算原函数 $y(t)$ 在离散点上的函数值；

2) 用定理 1 构造生成函数，用定理 2 计算系数 $w$；

3) 根据式(25)和式(26)构造 $u(t)$，将 $y(t)$ 分解为 $u(t)+v(t)$；

4) 应用式(1)计算 $v(t)$ 的分数阶微分或积分；

5) 如果计算 RL 微积分，应用式(29)计算 $u(t)$ 的 RL 微积分；如果计算 Caputo 微分，应用式(30)计算 $u(t)$ 的 Caputo 微分；

6) 应用式(27)计算 $y(t)$ 的 RL 微积分，或者应用式(28)计算 $y(t)$ 的 Caputo 微分.

例 2　应用本文提出的算法重新计算例 1 中的 RL 分数阶微分. 为了比较两种算法的计算效果，这里选择与例 1 相同的步长 $h=0.01$，构造 1 ~5 阶的生成函数，应用本文提出的算法计算数值解，得到的计算误差如表 1 所示.

由表 1 可见，本文算法的计算精度明显高于基于 FFT 算法的计算精度. 当 $h=0.01$ 时，本文算法的计算误差可以控制在 $10^{-12}$ 的级别. 另外选择大步长 $h=0.1$，构造 6 ~ 10 阶的生成函数，然后应用文本算法计算数值解，得到的计算误差如表 2 所示. 可见，即使选择非常大的步长，本文的算法仍然可以计算出精度很高的数值解.

表 1　当 $h=0.01$ 时的计算误差

Table 1　Computation errors when $h=0.01$

| $t$ | $p=1$ | $p=2$ | $p=3$ | $p=4$ | $p=5$ |
|---|---|---|---|---|---|
| 0.5 | 1.8E−3 | 1.2E−5 | 8.9E−8 | 7.1E−10 | 5.9E−12 |
| 1.0 | 1.7E−3 | 1.1E−5 | 8.6E−8 | 6.8E−10 | 5.7E−12 |
| 1.5 | 1.5E−3 | 1.0E−5 | 7.5E−8 | 6.0E−10 | 5.0E−12 |
| 2.0 | 1.3E−3 | 8.6E−6 | 6.4E−8 | 5.1E−10 | 4.3E−12 |
| 2.5 | 1.1E−3 | 7.4E−6 | 5.5E−8 | 4.4E−10 | 3.7E−12 |
| 3.0 | 9.6E−4 | 6.4E−6 | 4.8E−8 | 3.8E−10 | 3.2E−12 |
| 3.5 | 8.4E−4 | 5.6E−6 | 4.2E−8 | 3.4E−10 | 2.9E−12 |
| 4.0 | 7.5E−4 | 5.0E−6 | 3.7E−8 | 3.0E−10 | 2.8E−12 |
| 4.5 | 6.8E−4 | 4.5E−6 | 3.4E−8 | 2.7E−10 | 3.0E−12 |
| 5.0 | 6.2E−4 | 4.1E−6 | 3.1E−8 | 2.5E−10 | 3.4E−12 |

表 2　当 $h=0.1$ 时的计算误差

Table 2　Computation errors when $h=0.1$

| $t$ | $p=6$ | $p=7$ | $p=8$ | $p=9$ | $p=10$ |
|---|---|---|---|---|---|
| 0.5 | 3.1E−9 | 8.2E−11 | 3.0E−12 | 1.7E−13 | 4.4E−14 |
| 1.0 | 3.7E−8 | 2.9E−9 | 2.5E−10 | 2.0E−11 | 1.1E−12 |
| 1.5 | 3.7E−8 | 3.1E−9 | 2.5E−10 | 2.1E−11 | 1.5E−12 |
| 2.0 | 3.3E−8 | 2.8E−9 | 2.5E−10 | 1.7E−11 | 1.6E−12 |
| 2.5 | 2.8E−8 | 2.4E−9 | 2.5E−10 | 4.3E−11 | 3.9E−11 |
| 3.0 | 2.4E−8 | 2.1E−9 | 1.5E−10 | 2.5E−11 | 1.1E−10 |
| 3.5 | 2.1E−8 | 1.8E−9 | 4.5E−11 | 2.0E−10 | 6.9E−10 |
| 4.0 | 1.8E−8 | 1.5E−9 | 2.3E−10 | 5.9E−10 | 8.6E−9 |
| 4.5 | 1.6E−8 | 1.4E−9 | 4.9E−10 | 1.4E−9 | 2.5E−8 |
| 5.0 | 1.5E−8 | 1.3E−9 | 2.2E−10 | 1.0E−8 | 2.2E−7 |

为了比较本文算法与文献[8]中的算法，下面列举一个计算 Caputo 微分的实例.

例 3　应用本文算法计算函数 $e^{2t}$ 的 $\alpha$ 阶 Caputo 微分在 $t=1$ 处的数值解，数值解的精度可以用下面的解析解检验：

$$ {}_0^{\mathrm{C}}\mathscr{D}_t^{\alpha} e^{2t} = 2t^{1-\alpha} \varepsilon_{1,2-\alpha}(2t) . \quad (31) $$

此例引用自文献[8]，这里选择与文献[8]相同的 $\alpha$ 和步长，构造 6 阶生成函数，应用本文算法计算数值解，计算误差如表 3 所示.

表 3　两种算法的计算误差

Table 3　Computation errors of two algorithms

| $\alpha$ | $h$ | 本文算法 | 文献[8]算法 |
|---|---|---|---|
| 0.2 | 1/10 | 6.2124E−6 | 4.9025E−4 |
| | 1/20 | 1.5455E−7 | 4.4094E−5 |
| | 1/40 | 2.9006E−9 | 3.8638E−6 |
| | 1/80 | 4.9372E−11 | 3.4232E−7 |
| | 1/160 | 8.0647E−13 | 3.1440E−8 |
| 0.4 | 1/10 | 1.6234E−5 | 1.6156E−3 |
| | 1/20 | 3.7989E−7 | 1.5681E−4 |
| | 1/40 | 7.0048E−9 | 1.4478E−5 |
| | 1/80 | 1.1841E−10 | 1.3103E−6 |
| | 1/160 | 1.9256E−12 | 1.1851E−7 |
| 0.6 | 1/10 | 3.1264E−5 | 4.2309E−3 |
| | 1/20 | 6.9099E−7 | 4.5820E−4 |
| | 1/40 | 1.2550E−8 | 4.6839E−5 |
| | 1/80 | 2.1093E−10 | 4.6479E−6 |
| | 1/160 | 3.4515E−12 | 4.5469E−7 |
| 0.8 | 1/10 | 5.2502E−5 | 1.0190E−2 |
| | 1/20 | 1.1024E−6 | 1.2525E−3 |
| | 1/40 | 1.9780E−8 | 1.4521E−4 |
| | 1/80 | 3.3094E−10 | 1.6333E−5 |
| | 1/160 | 5.6612E−12 | 1.8089E−6 |

如果选择相同的步长，本文算法与文献[8]的算法相比，计算误差减小了至少两个数量级. 当

步长减小时,本文算法的误差减小得更为明显. 比较结果说明,与文献[8]的算法相比,本文提出的算法将计算精度提高了两个数量级以上.

文献[8]中的算法不能计算阶次在(0,1)区间以外的 Caputo 微分,但本文算法可以计算这样的 Caputo 微分. 另外,本文算法还可以计算分数阶积分,下面给出计算实例.

例 4 计算函数 $e^{-t}$ 的 0.6 阶 RL 积分在区间[0,5]内的数值解,可以应用下面的解析解检验数值解的精度:

$$ {}_0^{\mathrm{RL}}\mathscr{D}_t^{-0.6} e^{-t} = t^{0.6}\varepsilon_{1,1.6}(-t). \tag{32} $$

选择步长 $h=0.01$,构造 1 ~ 5 阶的生成函数,应用本文算法计算数值解,计算误差如表 4 所示,可见计算结果也很精确.

表 4 计算误差

Table 4 Computation errors

| $t$ | $p=1$ | $p=2$ | $p=3$ | $p=4$ | $p=5$ |
|---|---|---|---|---|---|
| 0.5 | 5.7E-4 | 3.6E-6 | 2.6E-8 | 2.0E-10 | 1.6E-12 |
| 1.0 | 1.5E-3 | 9.5E-6 | 7.0E-8 | 5.5E-10 | 4.5E-12 |
| 1.5 | 2.4E-3 | 1.6E-5 | 1.2E-7 | 9.2E-10 | 7.6E-12 |
| 2.0 | 3.3E-3 | 2.2E-5 | 1.6E-7 | 1.3E-9 | 1.1E-11 |
| 2.5 | 4.1E-3 | 2.7E-5 | 2.0E-7 | 1.6E-9 | 1.3E-11 |
| 3.0 | 4.9E-3 | 3.3E-5 | 2.4E-7 | 1.9E-9 | 1.6E-11 |
| 3.5 | 5.7E-3 | 3.7E-5 | 2.8E-7 | 2.2E-9 | 1.8E-11 |
| 4.0 | 6.4E-3 | 4.2E-5 | 3.1E-7 | 2.5E-9 | 2.1E-11 |
| 4.5 | 7.0E-3 | 4.6E-5 | 3.4E-7 | 2.7E-9 | 2.3E-11 |
| 5.0 | 7.6E-3 | 5.0E-5 | 3.7E-7 | 3.0E-9 | 2.5E-11 |

## 4 误差分析

对本文提出的算法进行误差分析,首先介绍下面的引理,此引理引自文献[6].

引理 1 用式(1)计算函数 $y(t)=t^{q-1}$ $(q>0)$ 的分数阶微积分,则有

$$ {}_0^{\mathrm{RL}}\mathscr{D}_t^{\alpha} y(t) = \frac{1}{h^{\alpha}}\sum_{k=0}^{[t/h]} w_k y(t-kh) + O(h^p) + O(h^q). \tag{33} $$

由引理 1 可知,式(1)的计算精度不仅与生成函数的阶次有关,而且与原函数的初值条件有关. 这里假设原函数 $y(t)$ 在原点处解析,可以写出 $y(t)$ 在原点的 Taylor 展开式为

$$ y(t)=\sum_{k=0}^{+\infty} c_k t^k. \tag{34} $$

本文算法将 $y(t)$ 分解为 $u(t)+v(t)$,由 $u(t)$ 和 $v(t)$ 的表达式可得

$$ y(t)=u(t)+v(t)=\sum_{k=0}^{p} c_k t^k + \sum_{k=p+1}^{+\infty} c_k t^k. \tag{35} $$

应用式(29)和式(30)可计算 $u(t)$ 的分数阶微积分的精确值. 当应用式(1)计算 $v(t)$ 的分数阶微积分时,$v(t)$ 具有零初值条件,即引理 1 中的 $p=q$,因此计算的精度只与生成函数的阶数有关,而与原函数的初值条件无关. 所以递推公式算法的计算精度为 $O(h^p)$.

## 5 结 论

1) 基于 FFT 算法误差较大的原因是应用了不准确的生成函数系数,而且在计算中未考虑原函数的非零初值条件对计算精度的影响.

2) 证明了计算生成函数系数的递推公式,并设计了递推公式算法,此算法可以计算非零初值条件下任意阶次的分数阶微分和积分.

3) 在相同的步长下,本文算法的计算精度高于文献[8]的算法两个数量级以上. 即使选择大步长 $h=0.1$,本文算法的计算误差仍然可以控制在 $10^{-10}$ 数量级.

**参考文献:**


[1] Wei Y H, Tse P W, Du B, et al. An innovative fixed-pole numerical approximation for fractional order systems[J]. *ISA Transactions*, 2016, 62: 94 – 102.

[2] Xue D Y, Bai L. Numerical algorithms for Caputo fractional-order differential equations [J]. *International Journal of Control*, 2017, 90(6): 1201 – 1211.

[3] Xue D Y, Bai L. Benchmark problems for Caputo fractional-order ordinary differential equations[J]. *Fractional Calculus and Applied Analysis*, 2017, 20(5): 1305 – 1312.

[4] Li M M, Wang J R. Finite time stability of fractional delay differential equations[J]. *Applied Mathematics Letters*, 2017, 64: 170 – 176.

[5] Li C P, Zeng F H. Numerical methods for fractional calculus [M]. Boca Raton: Chapman & Hall Crc, 2015: 50 – 100.

[6] Lubich C. Discretized fractional calculus[J]. *SIAM Journal on Mathematical Analysis*, 1986, 17(3): 704 – 719.

[7] Podlubny I. Fractional differential equations[M]. New York: Academic Press, 1999: 41 – 91.

[8] Cao J X, Li C P, Chen Y Q. High-order approximation to Caputo derivatives and Caputo-type advection-diffusion equations (Ⅱ)[J]. *Fractional Calculus and Applied Analysis*, 2015, 18(3): 735 – 761.