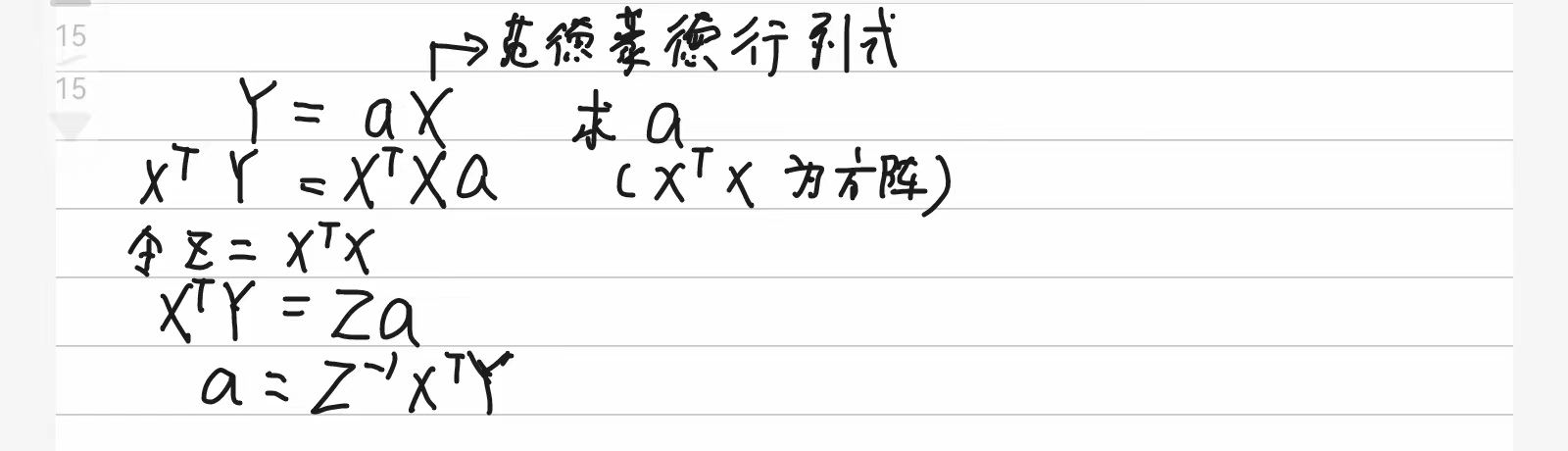
## 插值法小结

模型0为多项式插值，模型1为内维尔插值，模型2为有理函数插值

## 编程和学习过程中的小结

### 多项式插值

多项式插值的关键在于解出系数矩阵，以及迭代出计算式子



### 内维尔插值

内维尔插值的关键在于建立矩阵进行相关的迭代运算

### 有理函数插值

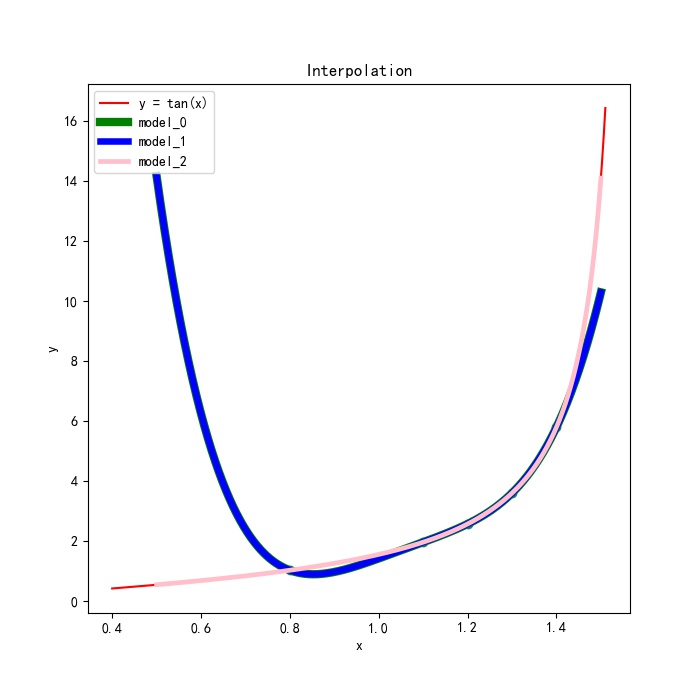
有理函数插值有点坑,但是对于tan（x）类型的函数是真的优秀。但有时候会分母为0，对插值点要求比较多，比如插值的时候y不能有0，就是在任何一步运算中都不能有分母为0的情况

小经验，python的-1可以代表最后一行（列），刚好在初始化时设定为了0，就不用做多余的增加维度的操作了

PS:我的有理函数插值虽然在某些时刻会碰到分母为0而不能正确输出，但在部分函数，精度优于官方，详见word最后一点

## 时间效率与基准度测试

## EXAMPLE1：tan（x），主要在0.8-1.5之间



三种模型与精准值比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input | Value | Test Input | True Value | Polynomial | Neville | Rational |
| 0.8 | 1.0296385570503641 | 0.75 | 0.9315964599440725 | 1.482744944320558 | 1.4827457593706665 | 0.931709855332858 |
| 1.1 | 1.9647596572486523 | 0.95 | 1.398382589287699 | 1.1690688598696681 | 1.169068976989774 | 1.3983244193652424 |
| 1.2 | 2.5721516221263188 | 1.15 | 2.2344969487553255 | 2.2470265651460295 | 2.2470266687890663 | 2.2345013630700996 |
| 1.3 | 3.6021024479679786 | 1.35 | 4.455221759562705 | 4.49274631863841 | 4.4927462156497 | 4.455245685735335 |
| 1.4 | 5.797883715482887 | 1.55 | 48.07848247921907 | 13.85239980414292 | 13.852400263245773 | 47.85780672977233 |
| time |  |  |  | 0.0 | 0.0 | 0.0 |

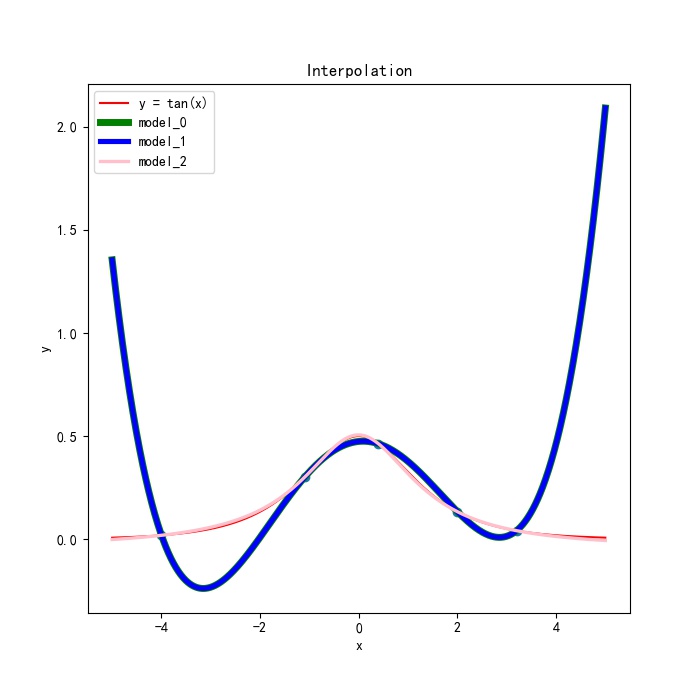
额，都挺快的，难以比较啊

误差值

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input | Value | Test Input | True Value | Polynomial | Neville | Rational |
| 0.8 | 1.0296385570503641 | 0.75 | 0.9315964599440725 | 0.5511484843764856 | 0.5511492994265941 | 0.00011339538878551991 |
| 1.1 | 1.9647596572486523 | 0.95 | 1.398382589287699 | -0.22931372941803096 | -0.22931361229792513 | -5.8169922456663414e-05 |
| 1.2 | 2.5721516221263188 | 1.15 | 2.2344969487553255 | 0.012529616390704046 | 0.012529720033740865 | 4.4143147741593225e-06 |
| 1.3 | 3.6021024479679786 | 1.35 | 4.455221759562705 | 0.03752455907570518 | 0.03752445608699517 | 2.3926172629806786e-05 |
| 1.4 | 5.797883715482887 | 1.55 | 48.07848247921907 | -34.22608267507615 | -34.22608221597329 | -0.22067574944674107 |

可以看到，外推精度有明显下降，对于突变型函数，有理函数插值会比较好

## Example 2 y=1/(np.exp(-x)+np.exp(x))



三种模型与精准值比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input | Value | Test Input | True Value | Polynomial | Neville | Rational |
| -4 | 0.018309496736843265 | -4.5 | 0.011107625748324837 | 0.5073727640498218 | 0.5073727640498548 | 0.006529948950152589 |
| -1.1 | 0.29966703028539643 | -1.6 | 0.19398909493724478 | 0.14436376407487567 | 0.14436376407486537 | 0.19845850431995898 |
| 0.4 | 0.4625037259528775 | 0 | 0.5 | 0.47372386553159346 | 0.47372386553160006 | 0.5054588830986309 |
| 2 | 0.13290111441703986 | 2.7 | 0.06690333839655072 | 0.013381963017715881 | 0.013381963017712552 | 0.06869882531362481 |
| 3.2 | 0.040694587590376606 | 3.8 | 0.02235958197131692 | 0.2995897414594264 | 0.29958974145944584 | 0.018291797474255075 |
| time |  |  |  | 0.0 | 0.0 | 0.0 |

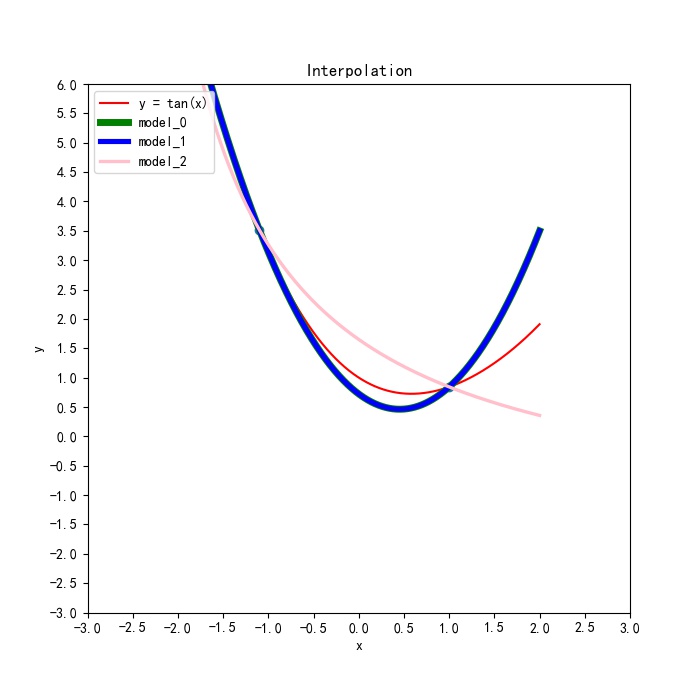
额，都挺快的，难以比较啊

误差值

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input | Value | Test Input | True Value | Polynomial | Neville | Rational |
| -4 | 0.018309496736843265 | -4.5 | 0.011107625748324837 | 0.49626513830149693 | 0.4962651383015299 | -0.004577676798172248 |
| -1.1 | 0.29966703028539643 | -1.6 | 0.19398909493724478 | -0.049625330862369116 | -0.04962533086237941 | 0.004469409382714201 |
| 0.4 | 0.4625037259528775 | 0 | 0.5 | -0.026276134468406542 | -0.026276134468399936 | 0.005458883098630851 |
| 2 | 0.13290111441703986 | 2.7 | 0.06690333839655072 | -0.053521375378834835 | -0.053521375378838165 | 0.0017954869170740961 |
| 3.2 | 0.040694587590376606 | 3.8 | 0.02235958197131692 | 0.2772301594881095 | 0.2772301594881289 | -0.0040677844970618444 |

可以看到，外推精度有明显下降，显然对于需要外推的炒股，插值方法不太实用

## Example 3 y=（x-1)^2+sin(x)



三种模型与精准值比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input | Value | Test Input | True Value | Polynomial | Neville | Rational |
| -2 | 8.090702573174319 | -3 | 15.858879991940134 | 15.580426238557468 | 15.580426238557472 | 1871.6879744828743 |
| -1.1 | 3.5187926399385647 | -1.2 | 3.9079609140327745 | 3.9253163698017532 | 3.9253163698017612 | 3.801658820291125 |
| 1 | 0.8414709848078965 | 1.8 | 1.6138476308781953 | 2.764060544548542 | 2.7640605445485527 | 0.4370819010142195 |
| time |  |  |  | 0.0 | 0.0 | 0.0 |

额，都挺快的，难以比较啊

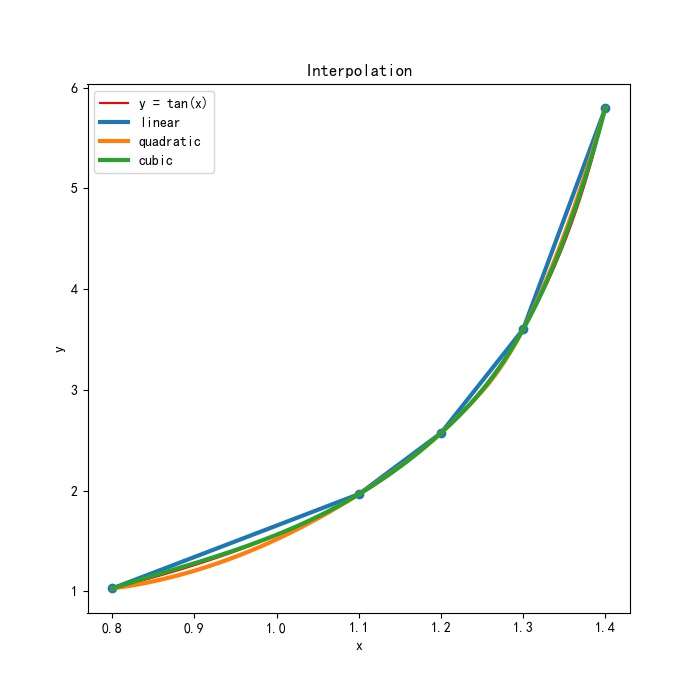
误差值

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input | Value | Test Input | True Value | Polynomial | Neville | Rational |
| -2 | 8.090702573174319 | -3 | 15.858879991940134 | -0.2784537533826654 | -0.2784537533826619 | 1855.8290944909343 |
| -1.1 | 3.5187926399385647 | -1.2 | 3.9079609140327745 | 0.01735545576897879 | 0.017355455768986783 | -0.10630209374164945 |
| 1 | 0.8414709848078965 | 1.8 | 1.6138476308781953 | 1.1502129136703467 | 1.1502129136703574 | -1.1767657298639758 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

可以看到，外推精度有明显下降,但在这种模型中，多项式模型效果更好

# 接下来就是官方库了

PYTHON 库主要是scipy.interpolate，对于这种分段插值，只支持内插，但由于分段的原因，精度还是蛮高的



三种模型与精准值比较

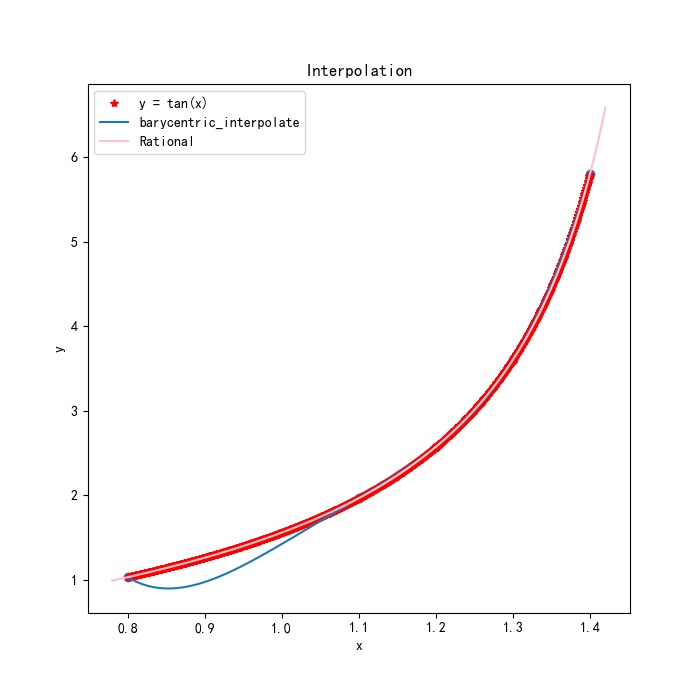
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input | Value | Test Input | True Value | Linear | Quadratic | Cubic |
| 0.8 | 1.0296385570503641 | 0.83 | 1.0934329172409998 | 1.1231506670701927 | 1.068097308580181 | 1.1062072015153737 |
| 1.1 | 1.9647596572486523 | 0.95 | 1.398382589287699 | 1.4971991071495079 | 1.34427311134392 | 1.4114557871523161 |
| 1.2 | 2.5721516221263188 | 1.15 | 2.2344969487553255 | 2.2684556396874846 | 2.23955561513049 | 2.23744074974309 |
| 1.3 | 3.6021024479679786 | 1.21 | 2.650324594970601 | 2.675146704710485 | 2.6484648087580265 | 2.6476142998153724 |
| 1.4 | 5.797883715482887 | 1.38 | 5.177437388630407 | 5.358727461979904 | 5.255868292163129 | 5.223473717205734 |

误差值

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input | Value | Test Input | True Value | Polynomial | Neville | Rational |
| 0.8 | 1.0296385570503641 | 0.83 | 1.0934329172409998 | 0.029717749829192952 | -0.025335608660818743 | 0.012774284274373882 |
| 1.1 | 1.9647596572486523 | 0.95 | 1.398382589287699 | 0.09881651786180878 | -0.05410947794377918 | 0.013073197864617025 |
| 1.2 | 2.5721516221263188 | 1.15 | 2.2344969487553255 | 0.0339586909321592 | 0.005058666375164389 | 0.0029438009877647 |
| 1.3 | 3.6021024479679786 | 1.21 | 2.650324594970601 | 0.024822109739883746 | -0.0018597862125746545 | -0.0027102951552286925 |
| 1.4 | 5.797883715482887 | 1.38 | 5.177437388630407 | 0.18129007334949687 | 0.07843090353272153 | 0.04603632857532691 |

可以看到，通过分段插值，大大提高了整体插值的精度

当然，python官方库也可以进行有理函数插值



至少可以看到，在这段区间内，我编写的插值比官方好！

至少可以看到，在这段区间内，我编写的插值比官方好！

至少可以看到，在这段区间内，我编写的插值比官方好！

三种模型与精准值比较

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | Value | Test Input | True Value | Neville | Rational |
| 0.8 | 1.0296385570503641 | 0.83 | 1.0934329172409998 | 0.9192771788776956 | 1.0933943565605864 |
| 1.1 | 1.9647596572486523 | 0.95 | 1.398382589287699 | 1.169068976989775 | 1.3983244193652424 |
| 1.2 | 2.5721516221263188 | 1.15 | 2.2344969487553255 | 2.2470266687890663 | 2.2345013630700996 |
| 1.3 | 3.6021024479679786 | 1.21 | 2.650324594970601 | 2.6468899329955784 | 2.65032320661139 |
|  |  |  |  |  |  |

误差值

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | Value | Test Input | True Value | Neville | Rational |
| 0.8 | 1.0296385570503641 | 0.83 | 1.0934329172409998 | -0.17415573836330422 | -3.8560680413413095e-05 |
| 1.1 | 1.9647596572486523 | 0.95 | 1.398382589287699 | -0.22931361229792402 | -5.8169922456663414e-05 |
| 1.2 | 2.5721516221263188 | 1.15 | 2.2344969487553255 | 0.012529720033740865 | 4.4143147741593225e-06 |
| 1.3 | 3.6021024479679786 | 1.21 | 2.650324594970601 | -0.0034346619750227347 | -1.3883592111518794e-06 |
|  |  |  |  |  |  |

可以看到，就这个函数来说，我的精度更高

# 接下来关于C++库

闭着眼睛都知道，C++的编译速度高于python的，但对于这种速度已经非常快的插值方法来说，这点小小的差异简直可以忽略不计

C++ 使用 boost 库实现插值算法,但不能直接用

按照官方文档给出的示例教程，在main函数中添加相应的插值函数的头文件，如：

#include <boost/math/interpolators/pchip.hpp>

修改一系列项目

拥有许多种插值函数

Cardinal Cubic B-spline interpolation 在等间距点采样的函数提供精准快速的插值方法

Barycentric Rational Interpolation 为采样点非均匀分布的数据提供插值方法

Modified Akima interpolation 采用cubic Hermite polynomials方法对非等间隔采样的数据进行插值，多项式斜率由 Akima 提出的修正后的几何重构法确定

PCHIP interpolation 与Modified Akima interpolation类似，PCHIP interpolation也采用cubic Hermite polynomials方法对非等间隔采样的数据进行插值，但斜率由另一种方法确定。