5. 自回归模型参数优化求解

自回归模型（Autoregressive Model）是用自身做回归变量的过程，即基于一定的模型公式，利用变量自身前期若干时刻的值描述当前时刻的变量值随，也可看作时间系列模型。

5.1 问题描述

案例模型公式及数据如下。两组时间系列变量 x 和 y，共有 5 个待求参数：k1, k2, a, b, 和 c，约束条件：k1 和 k2 范围均在[0,1]，a < 60*k1，b < 30*k2。

$$\begin{cases} x_{i+1} = (1-k_1)\cdot x_i + \frac{a}{1+\exp(c\cdot(x_i-y_i))} \\ y_{i+1} = (1-k_2)\cdot y_i + \frac{b}{1+\exp(c\cdot(x_i-y_i))} \end{cases} \quad (5\text{-}1)$$

表 5.1 数据

x	5.0,4.6,4.5,4.4,4.3,4.4,4.5,4.7,4.8,5.2,5.7,5.4,5.6,5.5,5.9,5.8,5.9,5.7,5.9,5.8,5.9,5.5,5.5,5.4,5.3
y	2.0,1.8,1.8,1.9,2,2.1,2.7,3.1,3.2,3.3,3.6,3.8,3.9,4.0,3.8,3.7,3.7,3.2,3.2,3.5,3.5,3.2,3.1,3.3,3.1

5.2 解决方案

1stOpt 的求解模式大概可分为二类，一是快捷模式：几乎就是将原模型表达式直接写入，非常简单易懂；二是编程模式：对于无法直接用简单表达式直接描述的模型，可嵌入 Pascal，Basic 或 Fortran 等高级语言对模型进行描述构建。

本案例时间序列自回归问题没有常规意义上的自变量和因变量，无法在快捷模式下实现，因此采用编程模式，分别在拟合及最优化两种计算模式下用 Pascal，Basic 和 Fortran 三种语言实现。计算时 x 和 y 的第一组数据用作各自的起始值。

5.3 拟合模式实现代码

Pascal 编程模式代码 5-1

```
Parameter [a,b,c]>0,[k1,k2]=[0,1];
Variable x[OutPut],y[OutPut];
StartProgram [Pascal];
Procedure MainModel;
var i: integer;
    temx, temy: double;
```

```
Begin
    temx := 5.0;
    temy := 2.0;
    for i := 0 to DataLength - 1 do begin
        temx := (1-k1)*temx+a/(1+exp(c*(temx-temy)));
        temy := (1-k2)*temy+b/(1+exp(c*(temx-temy)));
        x[i] := temx;
        y[i] := temy;
    end;
    ConstrainedResult := (a < 60*k1)and(b < 30*k2);
End;
EndProgram;
Data;
x=[4.6,4.5,4.4,4.3,4.4,4.5,4.7,4.8,5.2,5.7,5.4,5.6,5.5,5.9,5.8,5.9,5.7,5.9,5.8,5.9,5.5,5.5,5.4,5.3];
y=[1.8,1.8,1.9,2,2.1,2.7,3.1,3.2,3.3,3.6,3.8,3.9,4.0,3.8,3.7,3.7,3.2,3.2,3.5,3.5,3.2,3.1,3.3,3.1];
```

Basic 编程模式代码 5-2

```
Parameter [a,b,c]>0,[k1,k2]=[0,1];
Variable x[OutPut],y[OutPut];
StartProgram [Basic];
Sub MainModel
dim i as integer
dim as double temx, temy
    temx = 5.0
    temy = 2.0
    for i = 0 to DataLength - 1
        temx = (1-k1)*temx+a/(1+exp(c*(temx-temy)))
        temy = (1-k2)*temy+b/(1+exp(c*(temx-temy)))
        x(i) = temx
        y(i) = temy
    next
    ConstrainedResult = (a < 60*k1)and(b < 30*k2)
End Sub
EndProgram;
Data;
x=[4.6,4.5,4.4,4.3,4.4,4.5,4.7,4.8,5.2,5.7,5.4,5.6,5.5,5.9,5.8,5.9,5.7,5.9,5.8,5.9,5.5,5.5,5.4,5.3];
y=[1.8,1.8,1.9,2,2.1,2.7,3.1,3.2,3.3,3.6,3.8,3.9,4.0,3.8,3.7,3.7,3.2,3.2,3.5,3.5,3.2,3.1,3.3,3.1];
```

Fortran 编程模式代码 5-3

```
Parameter [a,b,c]>0,[k1,k2]=[0,1];
Variable x[OutPut],y[OutPut];
StartProgram [Fortran];
Subroutine MainModel
    integer i
    real(8) temx, temy
    temx = 5.0
    temy = 2.0
    do i = 0, DataLength - 1
        temx = (1-k1)*temx+a/(1+exp(c*(temx-temy)))
        temy = (1-k2)*temy+b/(1+exp(c*(temx-temy)))
```

```
            x(i) = temx
            y(i) = temy
        end do
        ConstrainedResult = (a < 60*k1)and(b < 30*k2)
End Subroutine
EndProgram;
Data;
x=[4.6,4.5,4.4,4.3,4.4,4.5,4.7,4.8,5.2,5.7,5.4,5.6,5.5,5.9,5.8,5.9,5.7,5.9,5.8,5.9,5.5,5.5,5.4,5.3];
y=[1.8,1.8,1.9,2,2.1,2.7,3.1,3.2,3.3,3.6,3.8,3.9,4.0,3.8,3.7,3.7,3.2,3.2,3.5,3.5,3.2,3.1,3.3,3.1];
```

上述三段不同高级语言编程模式下的代码均可获得以下相同的结果

均方差(RMSE): 0.1500799123397
残差平方和(SSR): 1.08115104421882
相关系数(R): 0.973609278788044
相关系数之平方(R^2): 0.947915027742176
修正 R 平方(Adj. R^2): 0.94297353022564
确定系数(DC): 0.939786592583878
F 统计(F-Statistic): 53.8912823645575

参数	最佳估算
a	3.9572575448015
b	10.1913718882801
c	1.00161329836361
k1	0.0822718528513837
k2	0.339712396276003

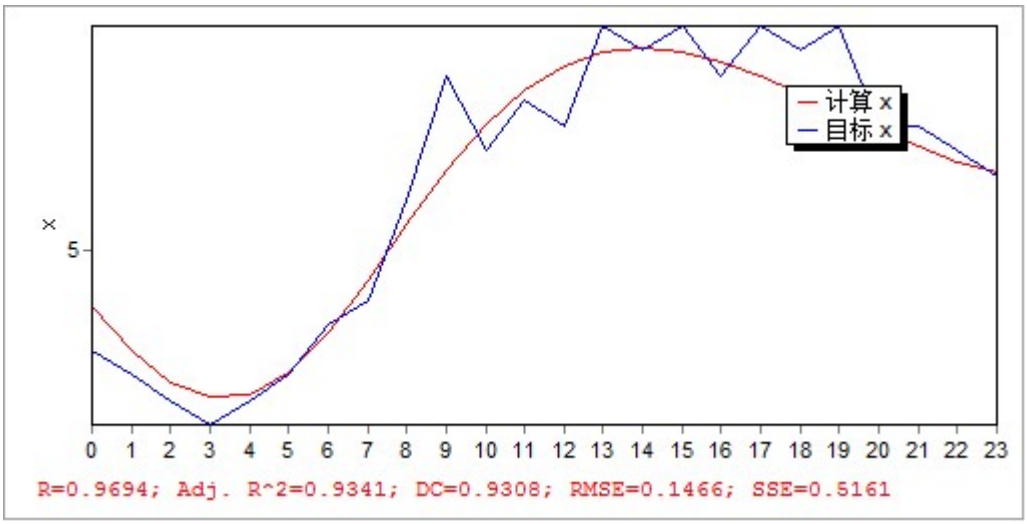

图 5-1 自回归模型 x 模拟计算与实际值对比

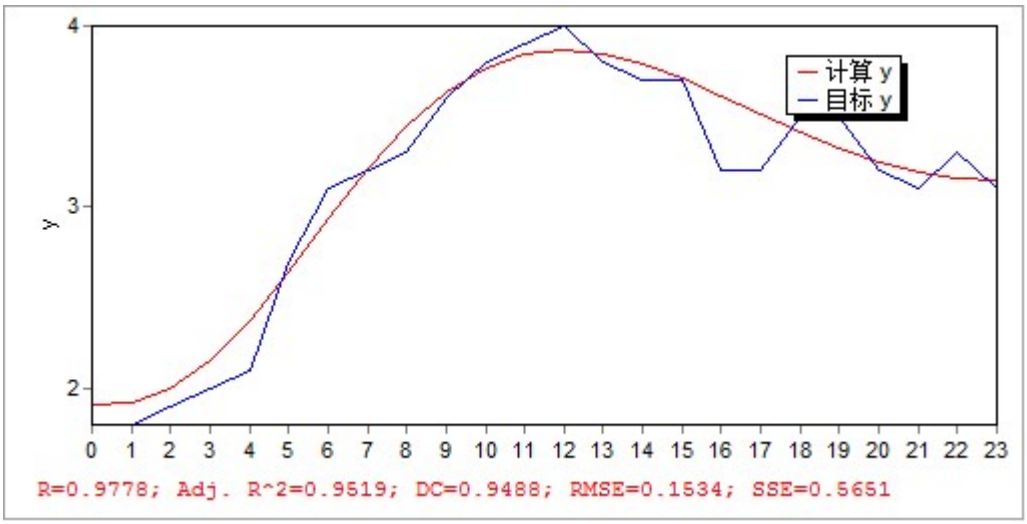

图 5-2 自回归模型 y 模拟计算与实际值对比

5.4 最优化模式实现代码

不同于上述拟合模式自动构建最小二乘拟合，这里通过代码自己构建优化求解模型，其目标函数即与最小二乘法一致。

Pascal 编程模式代码 5-4

```
Constant
x(0:23)=[4.6,4.5,4.4,4.3,4.4,4.5,4.7,4.8,5.2,5.7,5.4,5.6,5.5,5.9,5.8,5.9,5.7,5.9,5.8,5.9,5.5,5.5,5.4,5.3],
y(0:23)=[1.8,1.8,1.9,2,2.1,2.7,3.1,3.2,3.3,3.6,3.8,3.9,4.0,3.8,3.7,3.7,3.2,3.2,3.5,3.5,3.2,3.1,3.3,3.1];
Parameter [a,b,c]>0,[k1,k2]=[0,1];
PassParameter px(0:23), py(0:23);
Plot x,px,y,py;
StartProgram [Pascal];
Procedure MainModel;
var i: integer;
    temx, temy, temsum: double;
Begin
    temx := 5.0;
    temy := 2.0;
    temsum := 0;
    for i := 0 to 23 do begin
        temx := (1-k1)*temx+a/(1+exp(c*(temx-temy)));
        temy := (1-k2)*temy+b/(1+exp(c*(temx-temy)));
        px[i] := temx;
        py[i] := temy;
        temsum := temsum + sqr(x[i] - temx) + sqr(y[i] - temy);
    end;
    ObjectiveResult := temsum;
    ConstrainedResult := (a < 60*k1)and(b < 30*k2);
End;
```

```
EndProgram;
```

Basic 编程模式代码 5-5

```
Constant
x(0:23)=[4.6,4.5,4.4,4.3,4.4,4.5,4.7,4.8,5.2,5.7,5.4,5.6,5.5,5.9,5.8,5.9,5.7,5.9,5.8,5.9,5.5,5.5,5.4,5.3],
y(0:23)=[1.8,1.8,1.9,2,2.1,2.7,3.1,3.2,3.3,3.6,3.8,3.9,4.0,3.8,3.7,3.7,3.2,3.2,3.5,3.5,3.2,3.1,3.3,3.1];
Parameter [a,b,c]>0,[k1,k2]=[0,1];
PassParameter px(0:23), py(0:23);
Plot x,px,y,py;
StartProgram [Basic];
Sub MainModel
dim i as integer
dim as double temx, temy, temsum
    temx = 5.0
    temy = 2.0
    temsum = 0
    for i = 0 to 23
        temx = (1-k1)*temx+a/(1+exp(c*(temx-temy)))
        temy = (1-k2)*temy+b/(1+exp(c*(temx-temy)))
        px(i) = temx
        py(i) = temy
        temsum = temsum + (x(i) - temx)^2 + (y(i) - temy)^2
    next
    ObjectiveResult = temsum
    ConstrainedResult = (a < 60*k1)and(b < 30*k2)
End Sub
EndProgram;
```

Fortran 编程模式代码 5-6

```
Constant
x(0:23)=[4.6,4.5,4.4,4.3,4.4,4.5,4.7,4.8,5.2,5.7,5.4,5.6,5.5,5.9,5.8,5.9,5.7,5.9,5.8,5.9,5.5,5.5,5.4,5.3],
y(0:23)=[1.8,1.8,1.9,2,2.1,2.7,3.1,3.2,3.3,3.6,3.8,3.9,4.0,3.8,3.7,3.7,3.2,3.2,3.5,3.5,3.2,3.1,3.3,3.1];
Parameter [a,b,c]>0,[k1,k2]=[0,1];
PassParameter px(0:23), py(0:23);
Plot x,px,y,py;
StartProgram [Fortran];
Subroutine MainModel
    integer i
    real(8) temx, temy, temsum
    temx = 5.0
    temy = 2.0
    temsum = 0
    do i = 0, 23
        temx = (1-k1)*temx+a/(1+exp(c*(temx-temy)))
        temy = (1-k2)*temy+b/(1+exp(c*(temx-temy)))
        px(i) = temx
        py(i) = temy
        temsum = temsum + (x(i) - temx)^2 + (y(i) - temy)^2
    end do
```

```
    ObjectiveResult = temsum
    ConstrainedResult = (a < 60*k1)and(b < 30*k2)
End Subroutine
EndProgram;
```

上述三段不同高级语言编程模式下的代码也均可获得以下相同的结果

目标函数值(最小): 1.08115103378263 [1.08115103378263]
a: 3.95724687741913
b: 10.1913416342887
c: 1.00161187712782
k1: 0.0822718875679254
k2: 0.339712428798804

传递参数(PassParameter):
px0: 4.77545384922287
px1: 4.59608457004356
px2: 4.4708850180805
px3: 4.40956750481161
px4: 4.42103095775333
px5: 4.50939320900623
px6: 4.66808045400607
px7: 4.87622630487135
px8: 5.10305738141442
px9: 5.31864741588582
px10: 5.50214858723945
px11: 5.64317820194919
px12: 5.73924312892746
px13: 5.7927032589751
px14: 5.80856950407648
px15: 5.79314792886802
px16: 5.75324645064018
px17: 5.69570759542113
px18: 5.62712222936224
px19: 5.55364361284371
px20: 5.48085592864966
px21: 5.413666990165
px22: 5.35620167431564
px23: 5.31167935482585
py0: 1.91593845300965
py1: 1.91625852953201
py2: 1.99744022813992
py3: 2.15416295386134
py4: 2.37621470483881
py5: 2.64507309885022
py6: 2.93351891266379
py7: 3.21096331798237
py8: 3.45175047677343
py9: 3.64022793220947
py10: 3.77056600935454
py11: 3.84400112924928

py12: 3.8660917779754
py13: 3.84481014540719
py14: 3.78935760295053
py15: 3.70941170635408
py16: 3.61458483951451
py17: 3.51397793102961
py18: 3.41579074565974
py19: 3.32699438662912
py20: 3.25308460314123
py21: 3.19792614130769
py22: 3.16368375157698
py23: 3.15082857126226

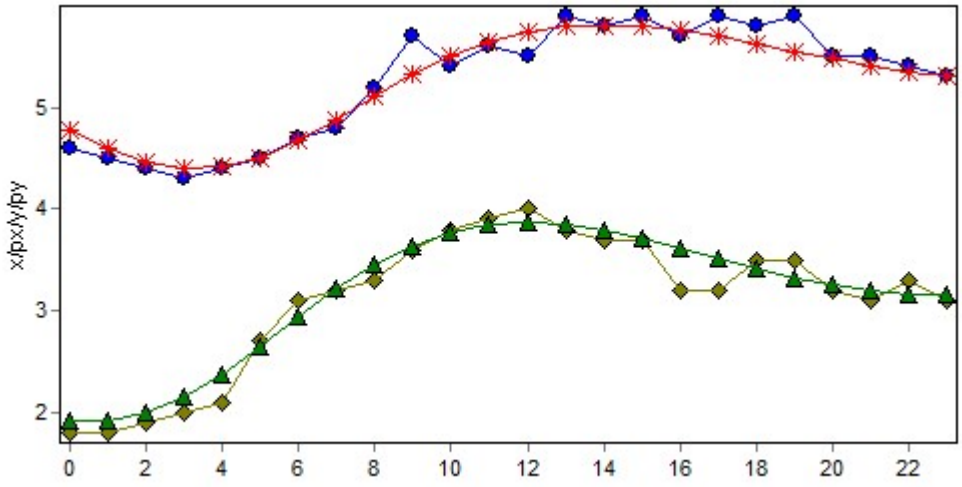

图 5-3 自回归模型模拟计算与实际值对比

5.5 小结

对于含约束条件的时间序列自回归参数求解问题，在编程模式下，1stOpt 可以很轻松实现，用户可以选择自己熟悉的高级语言描述并构筑模型目标函数。