$\caption' + stats + '$

StatsModels—Statistics in Python

Translated by Jge

Beginning on July 1, 2018

目录

欢迎来到Statsmodels的文档

`statsmodels`是一个Python模块，它提供了许多用于统计模型、统计检验和统计数据探索的类和函数。每个估计都有一个扩展的结果统计表，相关结果已根据现有的统计软件包进行了测试，确保了统计结果的正确性。该软件包是在开源修改的BSD（3-clause）许可下发布的。在线文档托管在statsmodels.org。

最小示例

从版本`0.5.0`开始`statsmodels`，您可以将R风格的公式与`pandas`数据框一起使用来拟合模型。这是一个使用普通最小二乘法的简单例子：

```
In [1]: import numpy as np
In [2]: import statsmodels.api as sm
In [3]: import statsmodels.formula.api as smf

# Load data
In [4]: dat = sm.datasets.get_rdataset("Guerry", "HistData").data

# Fit regression model (using the natural log of one of the regressors)
In [5]: results = smf.ols('Lottery ~ Literacy + np.log(Pop1831)', data=dat).fit()

# Inspect the results
In [6]: print(results.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                Lottery   R-squared:                       0.348
Model:                            OLS   Adj. R-squared:                  0.333
Method:                 Least Squares   F-statistic:                     22.20
Date:                Mon, 14 May 2018   Prob (F-statistic):           1.90e-08
Time:                        21:48:09   Log-Likelihood:                -379.82
No. Observations:                  86   AIC:                             765.6
Df Residuals:                      83   BIC:                             773.0
Df Model:                           2
Covariance Type:            nonrobust
================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept        246.4341     35.233      6.995      0.000     176.358     316.510
Literacy          -0.4889      0.128     -3.832      0.000      -0.743      -0.235
np.log(Pop1831)  -31.3114      5.977     -5.239      0.000     -43.199     -19.424
==============================================================================
Omnibus:                        3.713   Durbin-Watson:                   2.019
```

```
Prob(Omnibus):                  0.156   Jarque-Bera (JB):                3.394
Skew:                          -0.487   Prob(JB):                        0.183
Kurtosis:                       3.003   Cond. No.                         702.
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

您还可以使用`numpy`数组而不是公式：

```
In [7]: import numpy as np
In [8]: import statsmodels.api as sm

# Generate artificial data (2 regressors + constant)
In [9]: nobs = 100
In [10]: X = np.random.random((nobs, 2))
In [11]: X = sm.add_constant(X)
In [12]: beta = [1, .1, .5]
In [13]: e = np.random.random(nobs)
In [14]: y = np.dot(X, beta) + e

# Fit regression model
In [15]: results = sm.OLS(y, X).fit()

# Inspect the results
In [16]: print(results.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.183
Model:                            OLS   Adj. R-squared:                  0.166
Method:                 Least Squares   F-statistic:                     10.83
Date:                Mon, 14 May 2018   Prob (F-statistic):           5.68e-05
Time:                        21:48:10   Log-Likelihood:                -23.528
No. Observations:                 100   AIC:                             53.06
Df Residuals:                      97   BIC:                             60.87
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.4355      0.081     17.716      0.000       1.275       1.596
x1             0.2664      0.101      2.650      0.009       0.067       0.466
x2             0.4224      0.116      3.635      0.000       0.192       0.653
==============================================================================
Omnibus:                       75.567   Durbin-Watson:                   2.054
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                7.752
Skew:                           0.065   Prob(JB):                       0.0207
Kurtosis:                       1.642   Cond. No.                         5.32
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

可以通过`dir(results)`查看可用的统计结果。结果属性在`results.__doc__`中进行描述，结果方法也有他们自己的文档字符串（docstrings）。

引用

在科学出版物中使用statsmodels时，请考虑使用以下引文：

Seabold, Skipper, and Josef Perktold. "Statsmodels: Econometric and statistical modeling with python." *Proceedings of the 9th Python in Science Conference.* 2010.

Bibtex条目：

```
@inproceedings{seabold2010statsmodels,
  title={Statsmodels: Econometric and statistical modeling with python},
  author={Seabold, Skipper and Perktold, Josef},
  booktitle={9th Python in Science Conference},
  year={2010},
}
```

基本文档

Statsmodels 中的新功能

- 0.9版本
- 版本0.9.0
- 0.8版本
- 版本0.8.0
- 0.7版本
- 版本0.7.0
- 0.6版本
- 发布0.6.1
- 版本0.6.0
- 在0.6.0开发周期中结束的问题(Issues closed)
- 在0.6.0中结束的问题
- 0.5版本
- 版本0.5.0
- 在0.5.0开发周期中结束的问题

入门

这是一个非常简单的案例研究，旨在让您快速上手并运行`statsmodels`。我们将从原始数据开始，展示估计一个统计模型和绘制诊断图所需要的步骤。此处仅使用`statsmodels`或`pandas`与`patsy`依赖所提供的函数方法（functions）。

加载模块和方法

在安装statsmodels及其依赖项后，我们加载了一些模块和函数：

```
In [1]: from __future__ import print_function
In [2]: import statsmodels.api as sm
In [3]: import pandas
In [4]: from patsy import dmatrices
```

`pandas`建立在`numpy`数组上，提供了丰富的数据结构和数据分析工具。`pandas.DataFrame`函数提供了（可能是异质性的）数据的标签数组，类似于`R`的"数据框"(data.frame)。`pandas.read_csv`函数可用于将以逗号分隔值的文件转换为`DataFrame`对象。

`patsy`是一个用于描述统计模型和使用类似R公式建立设计矩阵(Design Matrices)的库。

数据

我们下载了Guerry数据集，这是用于支持安德列-米歇尔Guerry发表于1833年*法国道德统计*论文集的历史数据。该数据集由Rdatasets存储库以逗号分隔值格式（CSV）在线支持（hosted online）。我们可以本地下载数据文件然后使用`read_csv`加载，但是`pandas`会自动为我们进行处理：

```
In [5]: df = sm.datasets.get_rdataset("Guerry", "HistData").data
```

在输入/输出页面展示了如何导入其他各种格式的数据文件。

我们选择感兴趣的变量并查看最后的5行：

```
In [6]: vars = ['Department', 'Lottery', 'Literacy', 'Wealth', 'Region']
In [7]: df = df[vars]
In [8]: df[-5:]
Out[8]:
      Department  Lottery  Literacy  Wealth Region
81        Vienne       40        25      68      W
82  Haute-Vienne       55        13      67      C
83        Vosges       14        62      82      E
84         Yonne       51        47      30      C
85         Corse       83        49      37    NaN
```

请注意，"*Region*"列中有一处缺失值。我们使用`pandas`提供的`DataFrame`方法将其消除：

```
In [9]: df = df.dropna()
In [10]: df[-5:]
Out[10]:
      Department  Lottery  Literacy  Wealth Region
80        Vendee       68        28      56      W
81        Vienne       40        25      68      W
82  Haute-Vienne       55        13      67      C
83        Vosges       14        62      82      E
84         Yonne       51        47      30      C
```

实质性动机(Substantive motivation)和模型

我们想知道法国86个部门的识字率是否与19世纪20年代皇家彩票的人均赌注有关。我们需要控制每个部门的财富水平，并且我们还需要在回归方程的右边包含一系列哑变量，以控制由于区域效应而未观察到的异质性。该模型用普通最小二乘回归(OLS)估计。

- 设计矩阵（*endog*&*exog*）

  为了拟合`statsmodels`所覆盖的大多数模型，您需要创建两个设计矩阵。第一个是内生变量矩阵（即因变量，响应变量，回归子(regressand)等）。第二个是外生变量矩阵（即自变量，预测变量，回归元等）。OLS系数估计值通常计算如下：

  $\hat{\beta} = (X'X)^{-1}X'y$

  这里y是一个 关于人均彩票投注N×1的数据（*Lottery*）。X是 N×7带截距的数据矩阵， *Literacy* 和 *Wealth*变量，以及4个 region 二元变量。

  `patsy`模块为使用类似`R`公式准备设计矩阵提供了便利的功能。你可以在这里找到更多信息。

  我们使用`patsy`的`dmatrices`函数来创建设计矩阵：

  ```
  In [11]: y, X = dmatrices('Lottery ~ Literacy + Wealth + Region', data=df,
  return_type='dataframe')
  ```

  生成的矩阵/数据框如下所示：

  ```
  In [12]: y[:3]
  Out[12]:
     Lottery
  0     41.0
  1     38.0
  2     66.0

  In [13]: X[:3]
  Out[13]:
     Intercept  Region[T.E]  Region[T.N]  Region[T.S]  Region[T.W]  Literacy  \
  0        1.0          1.0          0.0          0.0          0.0      37.0
  1        1.0          0.0          1.0          0.0          0.0      51.0
  2        1.0          0.0          0.0          0.0          0.0      13.0
     Wealth
  0    73.0
  1    22.0
  2    61.0
  ```

  注意，`dmatrices`已经：

  - 将分类变量*Region*拆分为一组指标变量。
  - 给外生回归元矩阵添加了一个常数。
  - 返回`pandas`的DataFrame，而不是简单的numpy数组。这非常有用，因为数据框（DataFrames）允许`statsmodels`在报告结果时携带元数据（例如变量名称）。

  当然可以改变上述行为。查看patsy文档页面。

模型拟合和结果摘要(summary)

`statsmodels`拟合模型涉及3个典型的简单步骤：

1. 使用模型类来描述模型
2. 使用类方法拟合模型
3. 使用`summary`方法审查结果

对OLS，通过以下方式实现：

```
In [14]: mod = sm.OLS(y, X)    # Describe model
In [15]: res = mod.fit()       # Fit model
In [16]: print(res.summary())   # Summarize model
                            OLS Regression Results
==============================================================================
Dep. Variable:                Lottery   R-squared:                       0.338
Model:                            OLS   Adj. R-squared:                  0.287
Method:                 Least Squares   F-statistic:                     6.636
Date:                Mon, 14 May 2018   Prob (F-statistic):           1.07e-05
Time:                        21:48:07   Log-Likelihood:                -375.30
No. Observations:                  85   AIC:                             764.6
Df Residuals:                      78   BIC:                             781.7
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      38.6517      9.456      4.087      0.000      19.826      57.478
Region[T.E]   -15.4278      9.727     -1.586      0.117     -34.793       3.938
Region[T.N]   -10.0170      9.260     -1.082      0.283     -28.453       8.419
Region[T.S]    -4.5483      7.279     -0.625      0.534     -19.039       9.943
Region[T.W]   -10.0913      7.196     -1.402      0.165     -24.418       4.235
Literacy       -0.1858      0.210     -0.886      0.378      -0.603       0.232
Wealth          0.4515      0.103      4.390      0.000       0.247       0.656
==============================================================================
Omnibus:                        3.049   Durbin-Watson:                   1.785
Prob(Omnibus):                  0.218   Jarque-Bera (JB):                2.694
Skew:                          -0.340   Prob(JB):                        0.260
Kurtosis:                       2.454   Cond. No.                         371.
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

`res`对象具有许多有用的属性。例如，我们可以通过键入以下内容来提取参数估计值和r平方：

```
In [17]: res.params
Out[17]:
Intercept      38.651655
Region[T.E]   -15.427785
Region[T.N]   -10.016961
Region[T.S]    -4.548257
Region[T.W]   -10.091276
Literacy       -0.185819
Wealth          0.451475
dtype: float64

In [18]: res.rsquared
Out[18]: 0.337950869192882
```

通过`dir(res)`可查看完整的属性列表。

有关更多信息和示例，请参阅“ 回归文档页面

诊断和假设检验

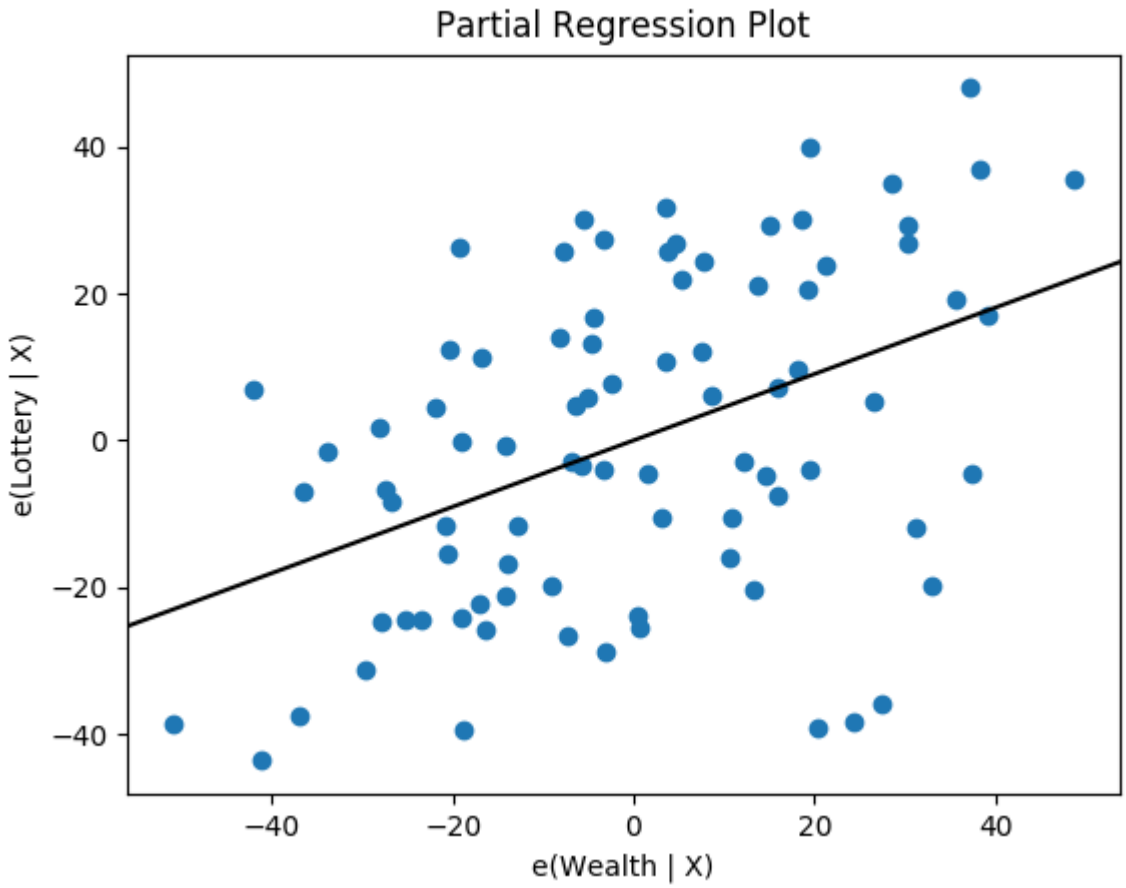
`statsmodels`允许您进行一系列有用的回归诊断和假设检验。例如，可以应用Rainbow检验模型是否线性（原假设是线性关系）：

```
In [19]: sm.stats.linear_rainbow(res)
Out[19]: (0.8472339976156913, 0.6997965543621643)
```

不可否认，上面生成的输出并不是非常详细，但是我们通过阅读文档字符串 （也就是`print(sm.stats.linear_rainbow.__doc__)`）知道第一个数字是F统计量，第二个数字是p值。

`statsmodels`还提供了图形功能。例如，我们可以通过以下方式绘制一组回归元的部分回归拟合图：

```
In [20]: sm.graphics.plot_partregress('Lottery', 'Wealth', ['Region', 'Literacy'],
   ....:                              data=df, obs_labels=False)
Out[20]: <Figure size 640x480 with 1 Axes>
```



更多

恭喜！您已准备好继续阅读目录中的其他主题

使用R型公式拟合模型

从版本0.5.0开始，`statsmodels`允许用户使用R语言风格的公式拟合统计模型。在`statsmodels`内部，使用`patsy`包将公式和数据转换为模型拟合所需的矩阵。公式框架非常强大; 本教程只触及皮毛。公式语言的完整描述可以在`patsy`文档中找到：

- Patsy公式语言描述

加载模块和功能

```
In [1]: import statsmodels.formula.api as smf
In [2]: import numpy as np
In [3]: import pandas
```

请注意，我们调用`statsmodels.formula.api`而不是通常的`statsmodels.api`。`formula.api`支持许多建立在`api`上的相同方法（例如OLS，GLM），但也包含大多数这些模型的小写字母形式。一般来说，小写模型(lower case models)接受`formula`和`df`参数，而大写模型(upper case ones)接受`endog`和`exog`设计矩阵。`formula`参数接受用`patsy`公式描述模型的字符串。`df`参数为一个`pandas`数据框。

`dir(smf)`将打印出可用模型列表 。

公式兼容模型(Formula-compatible models)具有以下通用调用签名(generic call signature): `(formula, data, subset=None, *args, **kwargs)`

使用公式进行OLS回归

首先，我们拟合入门页面上描述的线性模型。下载数据，提取子集列，并且以列表方式删除缺失值:

```
In [4]: df = sm.datasets.get_rdataset("Guerry", "HistData").data
In [5]: df = df[['Lottery', 'Literacy', 'Wealth', 'Region']].dropna()
In [6]: df.head()
Out[6]:
   Lottery  Literacy  Wealth Region
0       41        37      73      E
1       38        51      22      N
2       66        13      61      C
3       80        46      76      E
4       79        69      83      E
```

拟合模型:

```
In [7]: mod = smf.ols(formula='Lottery ~ Literacy + Wealth + Region', data=df)

In [8]: res = mod.fit()

In [9]: print(res.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                Lottery   R-squared:                       0.338
Model:                            OLS   Adj. R-squared:                  0.287
Method:                 Least Squares   F-statistic:                     6.636
Date:                Mon, 14 May 2018   Prob (F-statistic):           1.07e-05
Time:                        21:46:27   Log-Likelihood:                -375.30
No. Observations:                  85   AIC:                             764.6
Df Residuals:                      78   BIC:                             781.7
Df Model:                           6
Covariance Type:            nonrobust
===============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept      38.6517      9.456      4.087      0.000      19.826      57.478
Region[T.E]   -15.4278      9.727     -1.586      0.117     -34.793       3.938
Region[T.N]   -10.0170      9.260     -1.082      0.283     -28.453       8.419
Region[T.S]    -4.5483      7.279     -0.625      0.534     -19.039       9.943
Region[T.W]   -10.0913      7.196     -1.402      0.165     -24.418       4.235
Literacy       -0.1858      0.210     -0.886      0.378      -0.603       0.232
Wealth          0.4515      0.103      4.390      0.000       0.247       0.656
==============================================================================
Omnibus:                        3.049   Durbin-Watson:                   1.785
Prob(Omnibus):                  0.218   Jarque-Bera (JB):                2.694
Skew:                          -0.340   Prob(JB):                        0.260
Kurtosis:                       2.454   Cond. No.                         371.
==============================================================================
```

```
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

分类变量

查看上面打印的结果摘要，注意`patsy`确定*Region*变量的元素是文本字符串，因此它将*Region*处理为分类变量。`patsy`默认包含一个截距项，所以我们自动删除了一个*Region*类别。

如果*Region*是一个整型变量，而我们希望明确地将其处理为分类变量，那么我们可以使用`C()`运算符来完成：

```
In [10]: res = smf.ols(formula='Lottery ~ Literacy + Wealth + C(Region)', data=df).fit()
In [11]: print(res.params)
Intercept         38.651655
C(Region)[T.E]   -15.427785
C(Region)[T.N]   -10.016961
C(Region)[T.S]    -4.548257
C(Region)[T.W]   -10.091276
Literacy          -0.185819
Wealth             0.451475
dtype: float64
```

`patsy`分类变量更多高级功能的示例可以在这里找到：Patsy：分类变量的参照编码系统(Contrast Coding Systems)

运算符

我们已经看到“~”将模型的左侧与右侧分开，并且“+”将新列添加到设计矩阵中。

删除变量

“-”符号可用于删除列/变量。例如，我们可以通过以下方式从模型中删除截距项：

```
In [12]: res = smf.ols(formula='Lottery ~ Literacy + Wealth + C(Region) -1 ', data=df).fit()
In [13]: print(res.params)
C(Region)[C]    38.651655
C(Region)[E]    23.223870
C(Region)[N]    28.634694
C(Region)[S]    34.103399
C(Region)[W]    28.560379
Literacy        -0.185819
Wealth           0.451475
dtype: float64
```

交乘项

使用“:”将向设计矩阵中添加另外两列的乘积项。使用“*”不仅会添加乘积项，还将包括交乘的各列：

```
In [14]: res1 = smf.ols(formula='Lottery ~ Literacy : Wealth - 1', data=df).fit()
In [15]: res2 = smf.ols(formula='Lottery ~ Literacy * Wealth - 1', data=df).fit()
In [16]: print(res1.params)
Literacy:Wealth    0.018176
dtype: float64
In [17]: print(res2.params)
Literacy           0.427386
Wealth             1.080987
Literacy:Wealth   -0.013609
dtype: float64
```

运算符还可以做许多其他的事情。请参阅patsy文档以了解更多信息。

函数

您可以将矢量化函数应用于模型中的变量：

```
In [18]: res = smf.ols(formula='Lottery ~ np.log(Literacy)', data=df).fit()
In [19]: print(res.params)
Intercept           115.609119
np.log(Literacy)    -20.393959
dtype: float64
```

定义自定义函数：

```
In [20]: def log_plus_1(x):
   ....:     return np.log(x) + 1.
   ....:
In [21]: print(res.params)
Intercept           115.609119
np.log(Literacy)    -20.393959
dtype: float64
```

译注：`print(res.params)`输出结果一样，因为定义的函数应该而没有应用于模型，应该是官方文档失误。

命名空间

请注意，以上所有示例均使用调用命名空间来查找要所应用的函数。命名空间的使用可以通过`eval_env`关键字进行控制。例如，您可能想要使用`patsy.EvalEnvironment`自定义命名空间或者您可能想"清除"命名空间，我们提供了传递`eval_func=-1`参数的方法。默认是使用调用者命名空间，这可能会导致（非）预期的后果，例如，如果某人在其用户命名空间中有一个变量名为`C`，或在其数据结构中向`patsy`传递了`C`，并且`C`在公式中用来处理分类变量。请参阅Patsy API参考以获取更多信息。

使用尚未支持的模型公式

即使既定的`statsmodels`函数不支持公式，仍然可以使用`patsy`公式语言来生成设计矩阵。然后，这些矩阵可以作为endog和exog参数输入到拟合函数中。

生成`numpy`数组：

```
In [22]: import patsy
In [23]: f = 'Lottery ~ Literacy * Wealth'
In [24]: y, X = patsy.dmatrices(f, df, return_type='matrix')
In [25]: print(y[:5])
[[41.]
 [38.]
 [66.]
 [80.]
 [79.]]
In [26]: print(X[:5])
[[   1.   37.   73. 2701.]
 [   1.   51.   22. 1122.]
 [   1.   13.   61.  793.]
 [   1.   46.   76. 3496.]
 [   1.   69.   83. 5727.]]
```

`y` 和 `X` 将是 `numpy.ndarray` 的一个子类 `patsy.DesignMatrix` 的一个实例。

生成数据框：

```
In [27]: f = 'Lottery ~ Literacy * Wealth'

In [28]: y, X = patsy.dmatrices(f, df, return_type='dataframe')

In [29]: print(y[:5])
   Lottery
0     41.0
1     38.0
2     66.0
3     80.0
4     79.0

In [30]: print(X[:5])
   Intercept  Literacy  Wealth  Literacy:Wealth
0        1.0      37.0    73.0           2701.0
1        1.0      51.0    22.0           1122.0
2        1.0      13.0    61.0            793.0
3        1.0      46.0    76.0           3496.0
4        1.0      69.0    83.0           5727.0
```

```
In [31]: print(smf.OLS(y, X).fit().summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                Lottery   R-squared:                       0.309
Model:                            OLS   Adj. R-squared:                  0.283
Method:                 Least Squares   F-statistic:                     12.06
Date:                Mon, 14 May 2018   Prob (F-statistic):           1.32e-06
Time:                        21:46:27   Log-Likelihood:                -377.13
No. Observations:                  85   AIC:                             762.3
Df Residuals:                      81   BIC:                             772.0
Df Model:                           3
```

```
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept         38.6348     15.825      2.441      0.017       7.149      70.121
Literacy          -0.3522      0.334     -1.056      0.294      -1.016       0.312
Wealth             0.4364      0.283      1.544      0.126      -0.126       0.999
Literacy:Wealth   -0.0005      0.006     -0.085      0.933      -0.013       0.012
==============================================================================
Omnibus:                        4.447   Durbin-Watson:                   1.953
Prob(Omnibus):                  0.108   Jarque-Bera (JB):                3.228
Skew:                          -0.332   Prob(JB):                        0.199
Kurtosis:                       2.314   Cond. No.                     1.40e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.4e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

安装

- 使用setuptools

  使用pip安装最新版本的statsmodels

  > pip install -U statsmodels

  或者点击此链接到我们的PyPI页面，下载wheel或源代码进行安装。

  Statsmodels也可以通过Anaconda提供的conda获得 。最新版本可以使用如下命令安装

  > conda install -c conda-forge statsmodels

- 获取源代码

  我们并不经常发布，但我们源代码的主分支(master branch)通常满足日常的使用。您可以从我们的github存储库获取最新的源代码 。或者，如果你安装了git：

  ```
  git clone git://github.com/statsmodels/statsmodels.git
  ```

  如果您希望与github上的源代码保持同步，请定期这样做:

  ```
  git pull
  ```

- 从源代码安装

  您需要安装一个C语言编译器来构建statsmodels。如果您是从github源而不是源代码版本构建的，那么您也需要使用Cython。您可以按照以下说明获取Windows的C编译器。

  - Linux

  获得源码后，您可以（在具有适当权限下）：

```
python setup.py install
```

或者：

```
python setup.py build
python setup.py install
```

- window

如果可以，强烈建议使用64位Python。

- Python 2.7

  获取 用于Python 2.7的Microsoft Visual C ++编译器，然后使用如下安装

  > python setup.py install

- Python 3.5

  下载并安装最新版本的 Visual Studio Community Edition，然后使用如下安装

  > python setup.py install

- 32位或其他版本的Python

  您可以使用mingw32在Windows上构建32位版本的代码。

  首先，获取并安装mingw32。然后，您需要编辑distutils.cfg。这通常在像C：Python27Libdistutilsdistutils.cfg这样的地方找到。添加以下命令行：

  ```
  [build]
  compiler=mingw32
  ```

  然后在statsmodels目录下执行：

  ```
  python setup.py build
  python setup.py install
  ```

  或者

  您可以构建32位的Microsoft SDK。详细说明可以在Cython wiki上找到。这些说明的要点如下：您需要从Microsoft下载免费的Windows SDK C / C ++编译器。您必须使用**适用于Windows 7和.NET Framework 3.5 SP1的Microsoft Windows SDK**才能与Python 2.7,3.1和3.2兼容。3.5 SP1版本的链接是

  https://www.microsoft.com/en-us/download/details.aspx?id=18950

  对于Python 3.3和3.4，您需要使用的是适用于**Windows 7和.NET Framework 4的Microsoft Windows SDK**，从此链接可获得：

  https://www.microsoft.com/en-us/download/details.aspx?id=8279

  对于7.0，获取AMD64的ISO文件GRMSDKX_EN_DVD.iso。安装完成后，打开SDK Command Shell（开始 - > 所有程序 - > Microsoft Windows SDK v7.0 - > CMD Shell）。CD到statsmodels目录并输入：

  ```
  set DISTUTILS_USE_SDK=1
  ```

构建64位应用程序类型：

```
setenv /x64 /release
```

构建一个32位应用程序类型：

```
setenv /x86 /release
```

提示符颜色应该会改为绿色。然后照常进行安装:

```
python setup.py build
python setup.py install
```

对于7.1，说明完全相同，除非您使用上面提供的下载链接并确保您使用的是SDK 7.1。

如果要在不打开SDK CMD SHELL的情况下完成相同的操作，则可以在CMD Prompt或批处理文件中使用这些命令：

```
setlocal EnableDelayedExpansion
CALL "C:\Program Files\Microsoft SDKs\Windows\v7.0\Bin\SetEnv.cmd" /x64 /release
set DISTUTILS_USE_SDK=1
```

根据需要使用v7.1和/ x86替换v7.0和/ x64。

- 依赖
  - Python > = 2.7，包括Python 3.x
  - NumPy > = 1.8
  - SciPy > = 0.14
  - Pandas > = 0.14
  - Patsy > = 0.3.0
  - Cython > = 0.24才能从github而不是从源代码发行版构建代码。
- 可选的依赖关系
  - Matplotlib > = 1.4是绘制函数和运行很多示例所必需的。
  - 如果安装了X-12-ARIMA或 X-13ARIMA-SEATS可用于时间序列分析。
  - 需要Nose来运行测试套件。
  - 需要IPython > = 3.0才能在本地构建文档或使用notebook。
  - joblib > = 0.9可用于加速某些模型的分布式估计。
  - 运行notebook需要jupyter。

关于Statsmodels

背景

`scipy.stats` 的 `models` 模块最初由Jonathan Taylor编写，曾经是scipy的一部分，但后来被移除。在Google Summer of Code 2009中， `statsmodels` 经过更正，测试，改进并作为新软件包发布。从那以后，statsmodels开发团队继续增加新的模型，绘图工具和统计方法。

测试

大部分结果至少通过一个其他统计软件包进行验证：R，Stata或SAS。最初覆写和持续开发的指导原则是所有数字都必须经过验证。一些统计方法用蒙特卡罗研究进行测试。尽管我们努力遵循这种测试驱动的方法，但不能保证代码没有缺陷并且始终有效。一些辅助函数仍未得到充分测试，某些边缘情况可能也未被正确考虑，并且数值问题是许多统计模型所固有的。我们特别感谢有关这些问题的任何帮助和报告，以便我们可以不断改进现有模型。

代码稳定性

现有的模型大多在用户界面中解决，我们预计未来不会有很大的变化。对于现有代码，尽管还没有对API稳定性的保证，但我们对所有非常特殊的情形都有很长的弃用期，并且我们尽力将现有用户需要调整的更改保持在最低水平。对于一个较新的模型，当我们获得了更多的经验和反馈，可能会调整用户交互。这些更改始终会在我们文档的发布记录中进行说明。

经济支持

我们感谢为statsmodels的发展所获得的经济支持：

> Google www.google.com : Google Summer of Code (GSOC) 2009-2017.

> AQR www.aqr.com : financial sponsor for the work on Vector Autoregressive Models (VAR) by Wes McKinney

我们还要感谢为我们提供代码托管服务的github，托管我们文档的github.io，感谢python.org在PyPi上提供我们的下载。

我们也感谢我们的持续集成提供商， Travis CI 和 AppVeyor 进行单元测试，Codecov和 Coveralls 进行代码覆盖(code coverage)。

有关统计模型结构和开发的信息

什么是 endog 和 exog

Statsmodels正在使用 `endog` 和 `exog` 作为估计问题中的数据或者观察变量的名称。在不同的统计软件包或教科书中也经常使用其他名称，例如：

endog	exog
y	x
y variable	x variable
left hand side (LHS)	right hand side (RHS)
dependent variable	independent variable
regressand	regressors
outcome	design
response variable	explanatory variable

不同名称的使用通常是特定于域和模型的; 然而，我们几乎完全使用endog和exog。把这两个术语分开的一个重要提示是，exogenous在其名称中有一个“x”，正如x变量。

x和y是一个字母名，有时用于临时变量，本身不具有信息性。为了避免字母名我们决定用描述性的名称，并决定使用`endog`和`exog`。由于已受到批评，未来可能会改变。

背景

一些非正式的术语定义是

内生的：由系统内的因素引起的

外生的：由系统外因素引起的

*内生变量指经济或计量经济模型中被解释或者被预测的变量。*http://stats.oecd.org/glossary/detail.asp?ID=794

外生变量表示出现在经济或计量经济模型中的变量，但不由该模型解释（即它们由模型给出）。 http://stats.oecd.org/glossary/detail.asp?ID=890

在计量经济学和统计学中，术语的定义更加正式，并且根据模型而使用不同的外生性定义（弱，强，严格）。在statsmodels中，作为变量名来使用不能总是以正式的意义来解释，而是试图遵循相同的原则。

在最简单的形式中，模型以某种线性或非线性形式将观测变量y与另一组变量x相联系

```
y = f(x, beta) + noise
y = x * beta + noise
```

然而，为了得到一个统计模型，我们需要对解释变量x和噪声noise的属性进行额外的假设。对许多基本模型的一个标准假定是x与噪声noise不相关。更一般的定义，x是外生的，意味着我们不必考虑解释变量x是如何产生的，无论是通过设计还是从某些基本分布随机抽取，这时我们想估计x对y的效应或影响，或检验关于这种效应的假设。

换句话说，y 对于我们的模型是*内生的*，x 对于我们模型的估计是*外生*的。

作为一个例子，假设你进行一项实验，而第二个会话中有些实验对象不再可用。退出（drop-out）与你实验所得出的结论相关吗？换句话说，我们能否将退出决定视为我们问题的外生因素。【***译注：*** *这段话感觉翻译不好，不知道这个例子是个啥实验，原文为：As an example, suppose you run an experiment and for the second session some subjects are not available anymore. Is the drop-out relevant for the conclusions you draw for the experiment? In other words, can we treat the drop-out decision as exogenous for our problem.*】

这取决于用户知道（或参考教科书来找出）模型的基本统计假设。作为一个例子，在`OLS`中，如果误差或噪声项是随时间独立分布的（或随时间而不相关），`exog`就 可以有滞后因变量。然而，如果误差项自相关，那么OLS就不具有良好的统计特性（不具有一致性），并且正确的模型将是ARMAX。`statsmodels`具有回归诊断功能，用来检验某些假设是否合理。

导入路径和结构

我们提供了两种从statsmodels中导入函数和类的方法：

1. 用于交互式使用的API导入
    - 允许tab代码补全
2. 直接导入程序
    - 避免导入不必要的模块和命令

用于交互式使用的API导入

对于交互式使用，推荐的导入是：

```
import statsmodels.api as sm
```

导入statsmodels.api将加载statsmodels的大部分公共部分。这使得大多数函数和类可以方便地在一个或两个级别内使用，而不会使“sm”命名空间过于拥挤。

要查看可用的函数和类，可以键入以下内容（或使用IPython，Spyder，IDLE等的名称空间探索功能）：

```
>>> dir(sm)
['GLM', 'GLS', 'GLSAR', 'Logit', 'MNLogit', 'OLS', 'Poisson', 'Probit', 'RLM',
'WLS', '__builtins__', '__doc__', '__file__', '__name__', '__package__',
'add_constant', 'categorical', 'datasets', 'distributions', 'families',
'graphics', 'iolib', 'nonparametric', 'qqplot', 'regression', 'robust',
'stats', 'test', 'tools', 'tsa', 'version']
>>> dir(sm.graphics)
['__builtins__', '__doc__', '__file__', '__name__', '__package__',
'abline_plot', 'beanplot', 'fboxplot', 'interaction_plot', 'qqplot',
'rainbow', 'rainbowplot', 'violinplot']

>>> dir(sm.tsa)
['AR', 'ARMA', 'DynamicVAR', 'SVAR', 'VAR', '__builtins__', '__doc__',
'__file__', '__name__', '__package__', 'acf', 'acovf', 'add_lag',
'add_trend', 'adfuller', 'ccf', 'ccovf', 'datetools', 'detrend',
'filters', 'grangercausalitytests', 'interp', 'lagmat', 'lagmat2ds',
'pacf', 'pacf_ols', 'pacf_yw', 'periodogram', 'q_stat', 'stattools',
'tsatools', 'var']
```

注意

`api`模块可能不包括statsmodels的所有的公用功能。如果您发现应该添加到api的内容，请在github上提交问题或通过邮件列表告知。

statsmodels的子库包括api.py模块，主要用于收集这些子包的导入需求。例如，将subpackage / api.py 文件导入到statsmodels api中

```
from .nonparametric import api as nonparametric
```

用户无需直接加载subpackage / api.py模块。

直接导入程序

`statsmodels`子模块按主题进行排列（例如，*discrete*是离散选择模型，或*tsa*是时间序列分析）。我们的目录树（剥离下来的一部分）看起来像这样：

```
statsmodels/
    __init__.py
    api.py
    discrete/
        __init__.py
        discrete_model.py
```

```
        tests/
            results/
    tsa/
        __init__.py
        api.py
        tsatools.py
        stattools.py
        arima_model.py
        arima_process.py
        vector_ar/
            __init__.py
            var_model.py
            tests/
                results/
        tests/
            results/
    stats/
        __init__.py
        api.py
        stattools.py
        tests/
    tools/
        __init__.py
        tools.py
        decorators.py
        tests/
```

可以导入的子模块包含空的*__init__.py*，但用于运行子模块测试的一些测试代码除外。目的是在下一个版本中将所有目录更改为*api.py*和空*__init__.py*。

导入示例

函数和类：

```
from statsmodels.regression.linear_model import OLS, WLS
from statsmodels.tools.tools import rank, add_constant
```

模块

```
from statsmodels.datasets import macrodata
import statsmodels.stats import diagnostic
```

带有别名的模块

```
import statsmodels.regression.linear_model as lm
import statsmodels.stats.diagnostic as smsdia
import statsmodels.stats.outliers_influence as oi
```

我们目前没有关于子模块别名的约定。

陷阱(Pitfalls)

此页面列出了使用statsmodels时可能出现的问题。这些可能是数据相关或统计问题，也可能是软件设计，“非标准”使用模型或边缘事例的结果。

statsmodels提供了几个警示和辅助函数用于诊断检查（有关线性回归误设检查的示例，请参阅此博客文章）。报告当然不全面，但随着时间的推移会增加更多的警示和诊断功能。

虽然所有统计软件包的基本统计问题都是相同的，但软件实现方式在处理一些极端或边角情形(corner cases)方面有所不同。请报告模型可能无效的边角情形，以便我们可以适当地处理它们。

重复调用以适应不同的参数

结果实例通常需要访问相应模型实例的属性。用不同参数多次拟合模型可以改变模型属性。这意味着在模型重新拟合后，结果实例可能不再指向正确的模型属性。

因此，当我们想要使用不同的参数拟合模型时，最佳做法是创建单独的模型实例。

例如，这样做毫无问题，因为我们并不保留结果实例以供进一步使用

```
mod = AR(endog)
aic = []
for lag in range(1,11):
    res = mod.fit(maxlag=lag)
    aic.append(res.aic)
```

但是，当我们想要保留两个不同的估计结果时，建议创建两个单独的模型实例。

```
mod1 = RLM(endog, exog)
res1 = mod1.fit(scale_est='mad')
mod2 = RLM(endog, exog)
res2 = mod2.fit(scale_est=sm.robust.scale.HuberScale())
```

未识别的参数

exog非满秩(Rank deficient)，完全多重共线性

基于线性模型，GLS，RLM，GLM等的模型使用广义逆。这意味着：

- 秩不足的矩阵不会产生错误
- 几乎完全多重共线性或病态设计矩阵(ill-conditioned design matrices)的情况可能会产生不稳定的数值结果。如果这不是所期望的行为，用户需要手动检查矩阵的秩或条件数（condition number）。

注意：如果数据未重新调整，Statsmodels目前在针对Filip的NIST基准案例上是失败的，请参阅此博客

最大似然估计不完全收敛

在某些情况下，最大似然估计可能不存在，参数可能是无穷的或不唯一的（例如，在二元内生变量模型中的（准）分离((quasi-)separation)）。在默认设置下，如果优化算法没有达到收敛而停止，statsmodels将给出警告。然而，重要的是要知道收敛标准有时可能错误地指示收敛（例如，目标函数的值收敛但参数不收敛）。通常，用户需要验证收敛。

对二元Logit和Probit模型，如果检测到完美预测，statsmodels会抛出一个异常。然而，不能对准完美预测(quasi-perfect prediction)进行检查。

其他问题

数据变化不足

小数据集的数据或分类变量分组小的数据其变化可能不大。在这些情况下，结果可能无法识别或可能发生某些隐藏问题。

目前唯一已知的情况是稳健线性模型估计的完美拟合。对于RLM，如果残差等于零，那么它不会引起异常，但完美拟合可能使得某些结果产生NaN值（scale = 0和0/0 division）（问题＃55）。

开发者页面

本页面将介绍如何通过提交 补丁，统计测试，新模型或示例来为statsmodels的开发做出贡献。

statsmodels使用Git版本控制系统在Github上进行开发。

.....

内部类

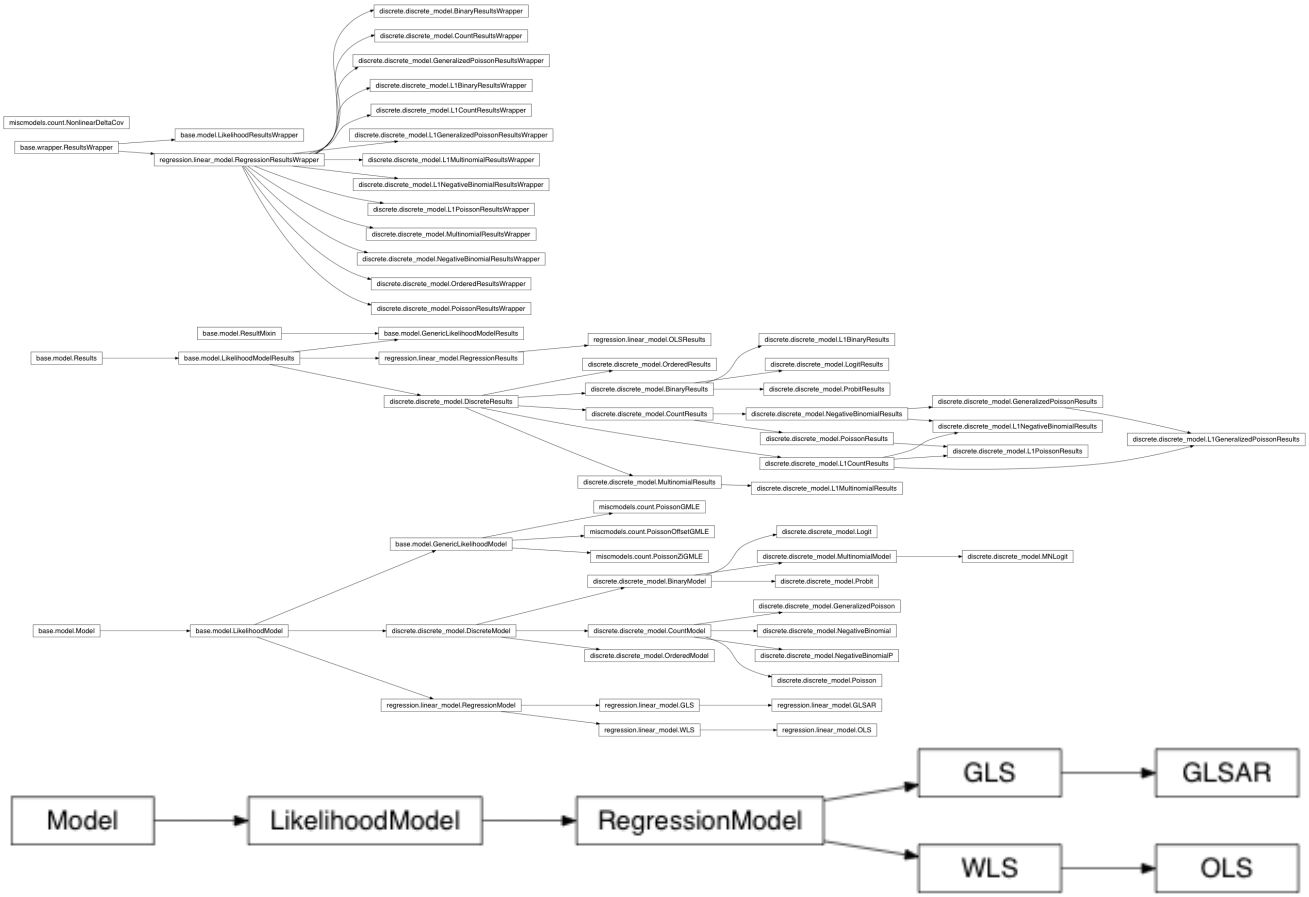
以下总结了不直接使用的类和函数，但主要向对内部使用感兴趣或想要扩展现有模型类的开发人员提供。

模块参考

模型和结果类

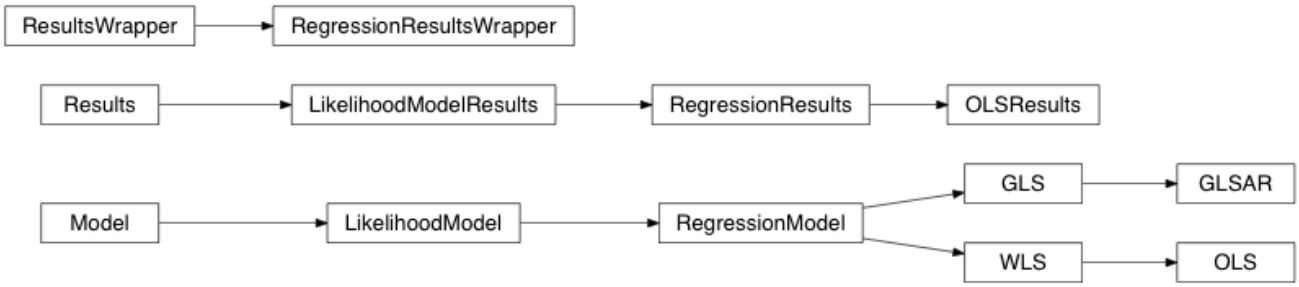
这些是估计模型和结果的基类。它们不是直接使用，而是用于布置子类的结构并定义一些常用方法。

Model（endog [，exog]）	一个（预测的）统计模型。
LikelihoodModel（endog [，exog]）	似然模型是Model的一个子类。
GenericLikelihoodModel（endog [，exog，...]）	允许通过最大似然拟合任何似然函数。
Results（model，params，** kwd）	包含模型结果的类
LikelihoodModelResults（model，params[，...]）	包含似然模型结果的类
ResultMixin	
GenericLikelihoodModelResults（model，mlefit）	离散因变量模型的结果类。
ContrastResults（[t，F，sd，effect，...]）	模型中系数线性约束检验的结果类。

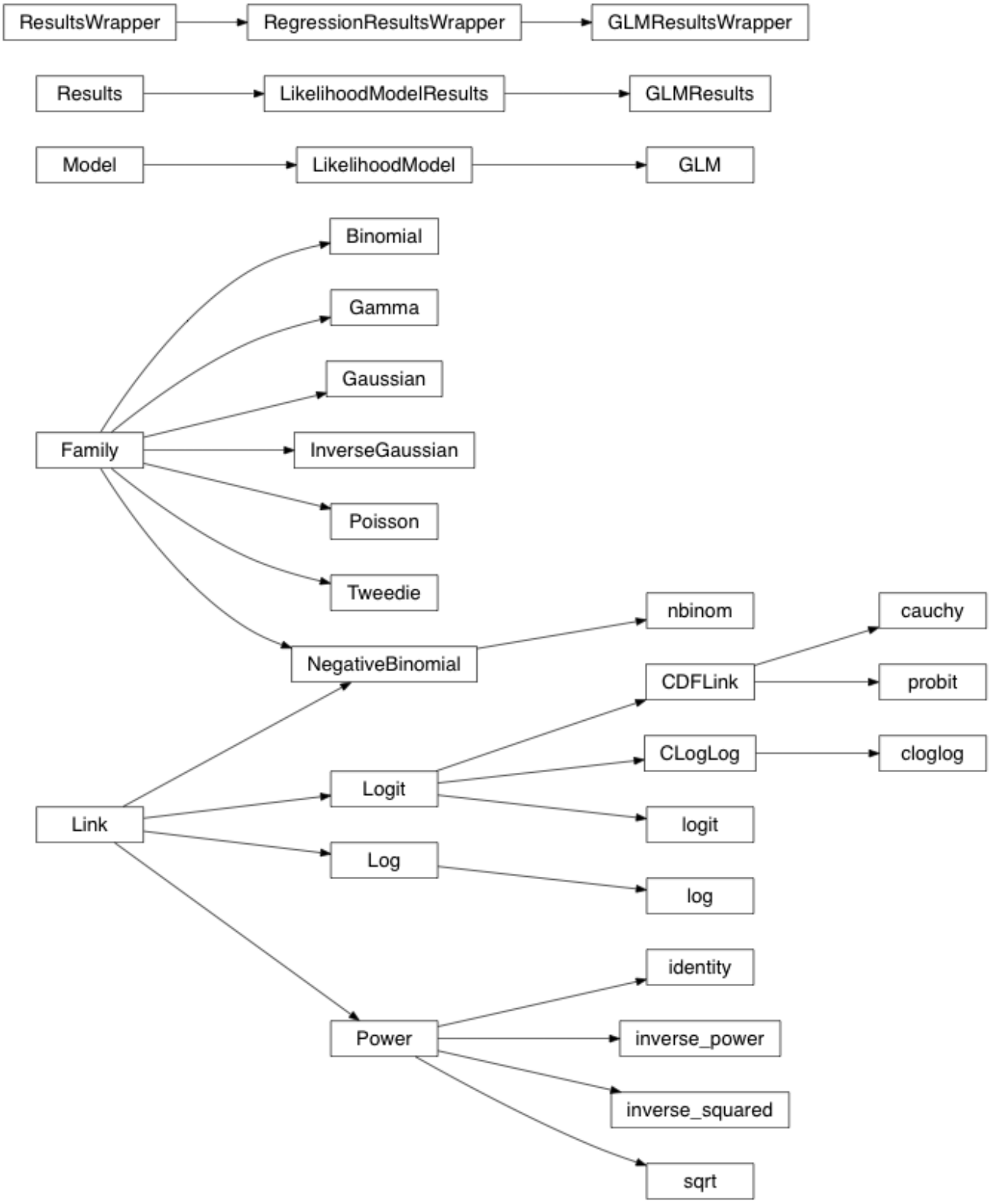

discrete.discrete_model.BinaryResultsWrapper
discrete.discrete_model.CountResultsWrapper
discrete.discrete_model.GeneralizedPoissonResultsWrapper
discrete.discrete_model.L1BinaryResultsWrapper
discrete.discrete_model.L1CountResultsWrapper
miscmodels.count.NonlinearDeltaCov
base.model.LikelihoodResultsWrapper
discrete.discrete_model.L1GeneralizedPoissonResultsWrapper
base.wrapper.ResultsWrapper
regression.linear_model.RegressionResultsWrapper
discrete.discrete_model.L1MultinomialResultsWrapper
discrete.discrete_model.L1NegativeBinomialResultsWrapper
discrete.discrete_model.L1PoissonResultsWrapper
discrete.discrete_model.MultinomialResultsWrapper
discrete.discrete_model.NegativeBinomialResultsWrapper
discrete.discrete_model.OrderedResultsWrapper
discrete.discrete_model.PoissonResultsWrapper
base.model.ResultMixin
base.model.GenericLikelihoodModelResults
base.model.Results
base.model.LikelihoodModelResults
regression.linear_model.RegressionResults
regression.linear_model.OLSResults
discrete.discrete_model.L1BinaryResults
discrete.discrete_model.OrderedResults
discrete.discrete_model.LogitResults
discrete.discrete_model.BinaryResults
discrete.discrete_model.ProbitResults
discrete.discrete_model.DiscreteResults
discrete.discrete_model.CountResults
discrete.discrete_model.NegativeBinomialResults
discrete.discrete_model.GeneralizedPoissonResults
discrete.discrete_model.L1NegativeBinomialResults
discrete.discrete_model.PoissonResults
discrete.discrete_model.L1GeneralizedPoissonResults
discrete.discrete_model.L1PoissonResults
discrete.discrete_model.L1CountResults
discrete.discrete_model.MultinomialResults
discrete.discrete_model.L1MultinomialResults
miscmodels.count.PoissonGMLE
miscmodels.count.PoissonOffsetGMLE
base.model.GenericLikelihoodModel
discrete.discrete_model.Logit
miscmodels.count.PoissonZIGMLE
discrete.discrete_model.MultinomialModel
discrete.discrete_model.MNLogit
discrete.discrete_model.BinaryModel
discrete.discrete_model.Probit
base.model.Model
base.model.LikelihoodModel
discrete.discrete_model.DiscreteModel
discrete.discrete_model.GeneralizedPoisson
discrete.discrete_model.CountModel
discrete.discrete_model.NegativeBinomial
discrete.discrete_model.OrderedModel
discrete.discrete_model.NegativeBinomialP
discrete.discrete_model.Poisson
regression.linear_model.RegressionModel
regression.linear_model.GLS
regression.linear_model.GLSAR
regression.linear_model.WLS
regression.linear_model.OLS

Model
LikelihoodModel
RegressionModel
GLS
GLSAR
WLS
OLS

线性模型

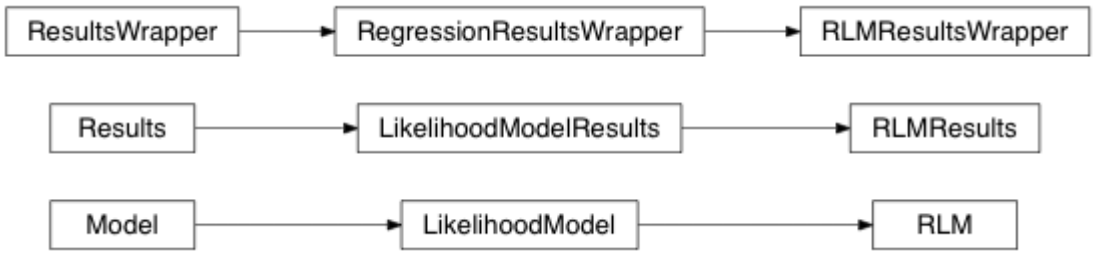

ResultsWrapper
RegressionResultsWrapper
Results
LikelihoodModelResults
RegressionResults
OLSResults
Model
LikelihoodModel
RegressionModel
GLS
GLSAR
WLS
OLS

广义线性模型

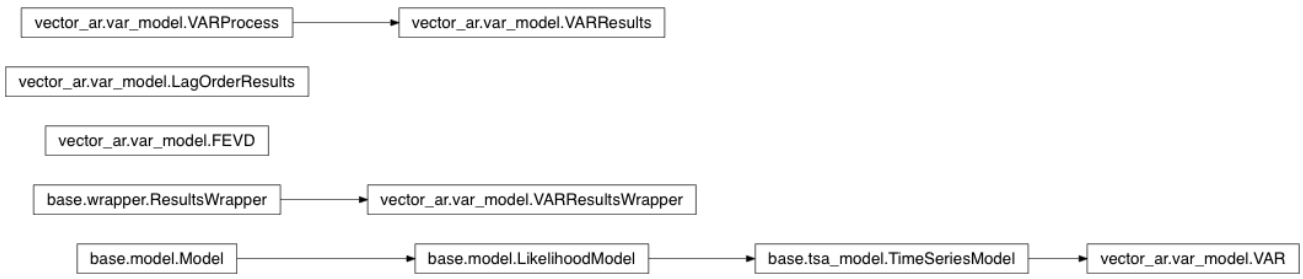

ResultsWrapper
RegressionResultsWrapper
GLMResultsWrapper
Results
LikelihoodModelResults
GLMResults
Model
LikelihoodModel
GLM
Family
Binomial
Gamma
Gaussian
InverseGaussian
Poisson
Tweedie
NegativeBinomial
Link
Logit
Log
Power
nbinom
CDFLink
CLogLog
logit
log
identity
inverse_power
inverse_squared
sqrt
cauchy
probit
cloglog

离散模型

稳健模型



向量自回归模型



目录

线性模型具有独立同分布误差，且误差可能具有异方差和自相关。该模块允许通过普通最小二乘法（OLS），加权最小二乘法（WLS），广义最小二乘法（GLS）以及具有p阶自相关AR(p)误差的可行广义最小二乘法进行估计。

有关命令和参数，请参阅模块参考。

示例

```
# Load modules and data
In [1]: import numpy as np
In [2]: import statsmodels.api as sm

In [3]: spector_data = sm.datasets.spector.load()
In [4]: spector_data.exog = sm.add_constant(spector_data.exog, prepend=False)

# Fit and summarize OLS model
In [5]: mod = sm.OLS(spector_data.endog, spector_data.exog)
In [6]: res = mod.fit()
In [7]: print(res.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.416
Model:                            OLS   Adj. R-squared:                  0.353
Method:                 Least Squares   F-statistic:                     6.646
Date:                Mon, 14 May 2018   Prob (F-statistic):            0.00157
Time:                        21:48:12   Log-Likelihood:                -12.978
No. Observations:                  32   AIC:                             33.96
Df Residuals:                      28   BIC:                             39.82
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.4639      0.162      2.864      0.008       0.132       0.796
x2             0.0105      0.019      0.539      0.594      -0.029       0.050
x3             0.3786      0.139      2.720      0.011       0.093       0.664
const         -1.4980      0.524     -2.859      0.008      -2.571      -0.425
==============================================================================
Omnibus:                        0.176   Durbin-Watson:                   2.346
Prob(Omnibus):                  0.916   Jarque-Bera (JB):                0.167
Skew:                           0.141   Prob(JB):                        0.920
Kurtosis:                       2.786   Cond. No.                         176.
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

详细示例

OLS

Ordinary Least Squares

```
%matplotlib inline
```

```
from __future__ import print_function
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.sandbox.regression.predstd import wls_prediction_std

np.random.seed(9876789)

#Artificial data:
nsample = 100
x = np.linspace(0, 10, 100)
X = np.column_stack((x, x**2))
beta = np.array([1, 0.1, 10])
e = np.random.normal(size=nsample)

# Our model needs an intercept so we add a column of 1s:
X = sm.add_constant(X)
y = np.dot(X, beta) + e

#Fit and summary:
model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 4.020e+06
Date:                Mon, 14 May 2018   Prob (F-statistic):          2.83e-239
Time:                        21:45:09   Log-Likelihood:                -146.51
No. Observations:                 100   AIC:                             299.0
Df Residuals:                      97   BIC:                             306.8
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.3423      0.313      4.292      0.000       0.722       1.963
x1            -0.0402      0.145     -0.278      0.781      -0.327       0.247
x2            10.0103      0.014    715.745      0.000       9.982      10.038
==============================================================================
Omnibus:                        2.042   Durbin-Watson:                   2.274
Prob(Omnibus):                  0.360   Jarque-Bera (JB):                1.875
Skew:                           0.234   Prob(JB):                        0.392
Kurtosis:                       2.519   Cond. No.                         144.
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
# Quantities of interest can be extracted directly from the fitted model. Type dir(results) for
a full list. Here are some examples:

print('Parameters: ', results.params)
print('R2: ', results.rsquared)
Parameters:  [ 1.34233516 -0.04024948 10.01025357]
R2:  0.9999879365025871
```

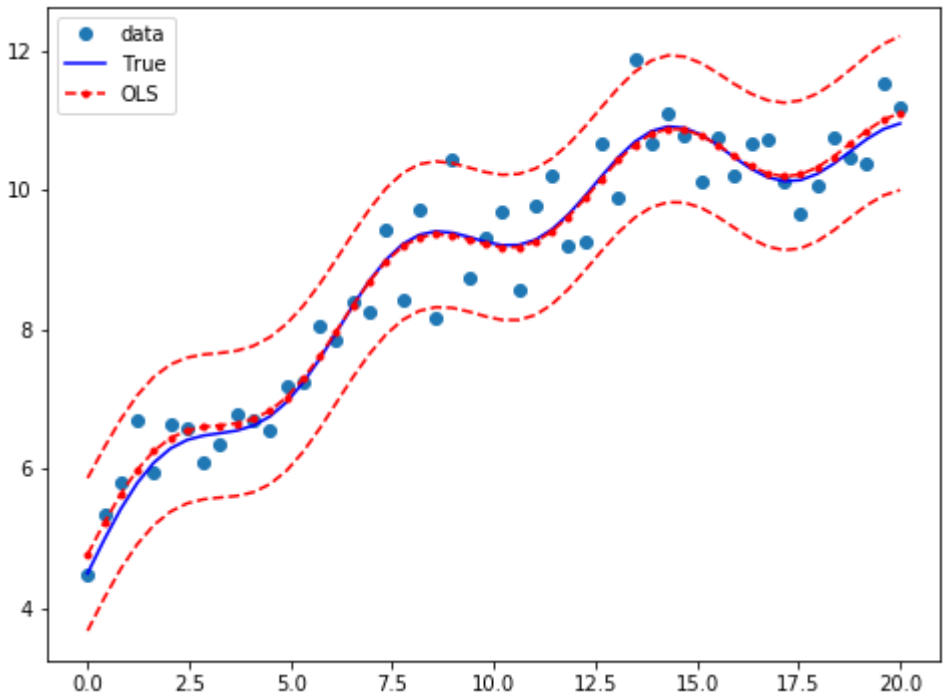
OLS non-linear curve but linear in parameters

```
# We simulate artificial data with a non-linear relationship between x and y:

nsample = 50
sig = 0.5
x = np.linspace(0, 20, nsample)
X = np.column_stack((x, np.sin(x), (x-5)**2, np.ones(nsample)))
beta = [0.5, 0.5, -0.02, 5.]

y_true = np.dot(X, beta)
y = y_true + sig * np.random.normal(size=nsample)

# Fit and summary:
res = sm.OLS(y, X).fit()
print(res.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.933
Model:                            OLS   Adj. R-squared:                  0.928
Method:                 Least Squares   F-statistic:                     211.8
Date:                Mon, 14 May 2018   Prob (F-statistic):           6.30e-27
Time:                        21:45:09   Log-Likelihood:                -34.438
No. Observations:                  50   AIC:                             76.88
Df Residuals:                      46   BIC:                             84.52
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.4687      0.026     17.751      0.000       0.416       0.522
x2             0.4836      0.104      4.659      0.000       0.275       0.693
x3            -0.0174      0.002     -7.507      0.000      -0.022      -0.013
const          5.2058      0.171     30.405      0.000       4.861       5.550
==============================================================================
Omnibus:                        0.655   Durbin-Watson:                   2.896
Prob(Omnibus):                  0.721   Jarque-Bera (JB):                0.360
Skew:                           0.207   Prob(JB):                        0.835
Kurtosis:                       3.026   Cond. No.                         221.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
# Extract other quantities of interest:
print('Parameters: ', res.params)
print('Standard errors: ', res.bse)
print('Predicted values: ', res.predict())
```

```
Parameters:  [ 0.46872448  0.48360119 -0.01740479  5.20584496]
Standard errors:  [0.02640602 0.10380518 0.00231847 0.17121765]
Predicted values:  [ 4.77072516  5.22213464  5.63620761  5.98658823  6.25643234  6.44117491
  6.54928009  6.60085051  6.62432454  6.6518039   6.71377946  6.83412169
  7.02615877  7.29048685  7.61487206  7.97626054  8.34456611  8.68761335
  8.97642389  9.18997755  9.31866582  9.36587056  9.34740836  9.28893189
  9.22171529  9.17751587  9.1833565   9.25708583  9.40444579  9.61812821
  9.87897556 10.15912843 10.42660281 10.65054491 10.8063004  10.87946503
 10.86825119 10.78378163 10.64826203 10.49133265 10.34519853 10.23933827
 10.19566084 10.22490593 10.32487947 10.48081414 10.66779556 10.85485568
 11.01006072 11.10575781]
```

```
# Draw a plot to compare the true relationship to OLS predictions. Confidence intervals around
the predictions are built using the wls_prediction_std command.

prstd, iv_l, iv_u = wls_prediction_std(res)

fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label="data")
ax.plot(x, y_true, 'b-', label="True")
ax.plot(x, res.fittedvalues, 'r--.', label="OLS")
ax.plot(x, iv_u, 'r--')
ax.plot(x, iv_l, 'r--')
ax.legend(loc='best');
```

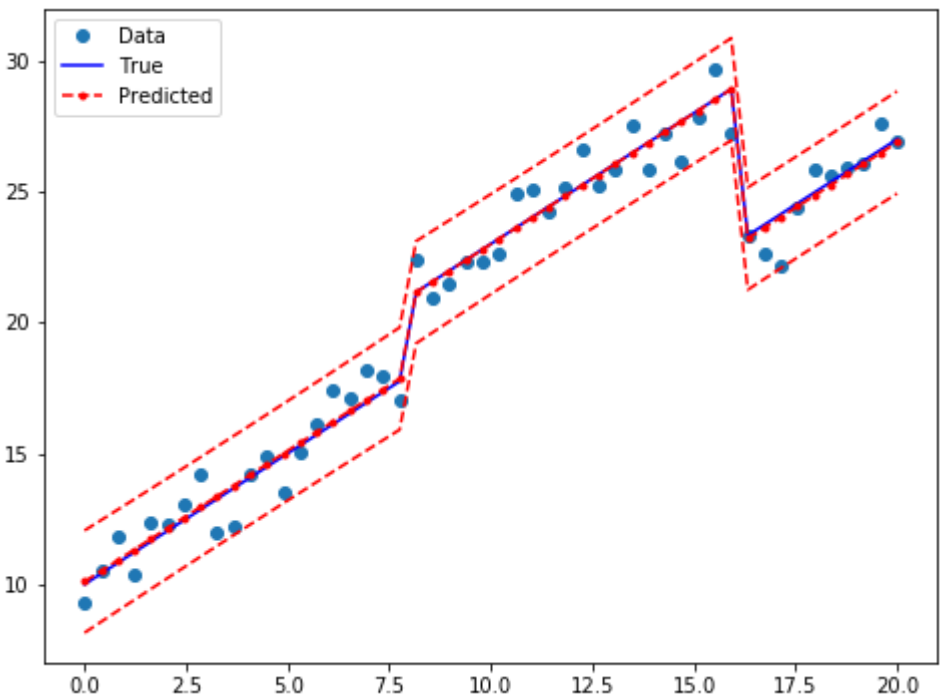


OLS with dummy variables

```
# We generate some artificial data. There are 3 groups which will be modelled using dummy
variables. Group 0 is the omitted/benchmark category.
nsample = 50
groups = np.zeros(nsample, int)
groups[20:40] = 1
groups[40:] = 2

#dummy = (groups[:,None] == np.unique(groups)).astype(float)
dummy = sm.categorical(groups, drop=True)

x = np.linspace(0, 20, nsample)

# drop reference category
X = np.column_stack((x, dummy[:,1:]))
X = sm.add_constant(X, prepend=False)

beta = [1., 3, -3, 10]
y_true = np.dot(X, beta)
e = np.random.normal(size=nsample)
y = y_true + e

# Inspect the data:
print(X[:5,:])
print(y[:5])
print(groups)
print(dummy[:5,:])
[[0.         0.         0.         1.        ]
 [0.40816327 0.         0.         1.        ]
 [0.81632653 0.         0.         1.        ]
 [1.2244898  0.         0.         1.        ]
 [1.63265306 0.         0.         1.        ]]
[ 9.28223335 10.50481865 11.84389206 10.38508408 12.37941998]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 2 2 2 2 2 2 2 2 2 2]
[[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]

# Fit and summary:
res2 = sm.OLS(y, X).fit()
print(res2.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.978
Model:                            OLS   Adj. R-squared:                  0.976
Method:                 Least Squares   F-statistic:                     671.7
Date:                Mon, 14 May 2018   Prob (F-statistic):           5.69e-38
Time:                        21:45:09   Log-Likelihood:                -64.643
No. Observations:                  50   AIC:                             137.3
Df Residuals:                      46   BIC:                             144.9
Df Model:                           3
```

```
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.9999      0.060     16.689      0.000       0.879       1.121
x2             2.8909      0.569      5.081      0.000       1.746       4.036
x3            -3.2232      0.927     -3.477      0.001      -5.089      -1.357
const         10.1031      0.310     32.573      0.000       9.479      10.727
==============================================================================
Omnibus:                        2.831   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.243   Jarque-Bera (JB):                1.927
Skew:                          -0.279   Prob(JB):                        0.382
Kurtosis:                       2.217   Cond. No.                         96.3
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
# Draw a plot to compare the true relationship to OLS predictions:
prstd, iv_l, iv_u = wls_prediction_std(res2)

fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label="Data")
ax.plot(x, y_true, 'b-', label="True")
ax.plot(x, res2.fittedvalues, 'r--.', label="Predicted")
ax.plot(x, iv_u, 'r--')
ax.plot(x, iv_l, 'r--')
legend = ax.legend(loc="best")
```



Joint hypothesis test

F检验

我们想要检验虚拟变量的系数均等于0这一假设，也就是$R \times \beta = 0$。F检验会使我们强烈拒绝3个分组具有同一常数项这一原假设。

```
R = [[0, 1, 0, 0], [0, 0, 1, 0]]
print(np.array(R))
print(res2.f_test(R))
[[0 1 0 0]
 [0 0 1 0]]
<F test: F=array([[145.49268198]]), p=1.2834419617287236e-20, df_denom=46, df_num=2>

# You can also use formula-like syntax to test hypotheses
print(res2.f_test("x2 = x3 = 0"))
<F test: F=array([[145.49268198]]), p=1.2834419617287776e-20, df_denom=46, df_num=2>
```

Small group effects

如果我们人工生成的数据具有小组效应，T检验可能不再拒绝原假设：

```
beta = [1., 0.3, -0.0, 10]
y_true = np.dot(X, beta)
y = y_true + np.random.normal(size=nsample)
res3 = sm.OLS(y, X).fit()

print(res3.f_test(R))
<F test: F=array([[1.22491119]]), p=0.3031864410631761, df_denom=46, df_num=2>

print(res3.f_test("x2 = x3 = 0"))
<F test: F=array([[1.22491119]]), p=0.3031864410631761, df_denom=46, df_num=2>
```

Multicollinearity

众所周知，Longley数据集具有高度多重共线性。也就是说，外生预测因子是高度相关的。这会成为一个问题，因为当我们对模型设定进行微小的改变，它就会影响我们对系数估计的稳定性。

```
from statsmodels.datasets.longley import load_pandas
y = load_pandas().endog
X = load_pandas().exog
X = sm.add_constant(X)

#Fit and summary:
ols_model = sm.OLS(y, X)
ols_results = ols_model.fit()
print(ols_results.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                 TOTEMP   R-squared:                       0.995
Model:                            OLS   Adj. R-squared:                  0.992
Method:                 Least Squares   F-statistic:                     330.3
Date:                Mon, 14 May 2018   Prob (F-statistic):           4.98e-10
Time:                        21:45:10   Log-Likelihood:                -109.62
No. Observations:                  16   AIC:                             233.2
```

```
Df Residuals:                     9   BIC:                             238.6
Df Model:                         6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -3.482e+06    8.9e+05     -3.911      0.004    -5.5e+06   -1.47e+06
GNPDEFL       15.0619     84.915      0.177      0.863    -177.029     207.153
GNP           -0.0358      0.033     -1.070      0.313      -0.112       0.040
UNEMP         -2.0202      0.488     -4.136      0.003      -3.125      -0.915
ARMED         -1.0332      0.214     -4.822      0.001      -1.518      -0.549
POP           -0.0511      0.226     -0.226      0.826      -0.563       0.460
YEAR        1829.1515    455.478      4.016      0.003     798.788    2859.515
==============================================================================
Omnibus:                        0.749   Durbin-Watson:                   2.559
Prob(Omnibus):                  0.688   Jarque-Bera (JB):                0.684
Skew:                           0.420   Prob(JB):                        0.710
Kurtosis:                       2.434   Cond. No.                     4.86e+09
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.86e+09. This might indicate that there are
strong multicollinearity or other numerical problems.
```

condition number

评估多重共线性的一种方法是计算条件数量。超过20的值是令人担忧的（参见Greene 4.9）。第一步是将自变量标准化为单位长度：

```
norm_x = X.values
for i, name in enumerate(X):
    if name == "const":
        continue
    norm_x[:,i] = X[name]/np.linalg.norm(X[name])
norm_xtx = np.dot(norm_x.T,norm_x) # 标准化X的协方差矩阵
```

然后，我们取最大与最小特征值之比的平方根：

```
eigs = np.linalg.eigvals(norm_xtx) # 求特征值
condition_number = np.sqrt(eigs.max() / eigs.min())
print(condition_number)
56240.8714071224
```

Dropping an observation

Greene还指出，删除一个观察值会对系数估计产生显着影响：

```
ols_results2 = sm.OLS(y.iloc[:14], X.iloc[:14]).fit()
print("Percentage change %4.2f%%\n"*7 % tuple([i for i in (ols_results2.params -
ols_results.params)/ols_results.params*100]))
```

```
Percentage change 4.55%
Percentage change -2228.01%
Percentage change 154304695.31%
Percentage change 1366329.02%
Percentage change 1112549.36%
Percentage change 92708715.91%
Percentage change 817944.26%

# We can also look at formal statistics for this such as the DFBETAS -- a standardized measure
of how much each coefficient changes when that observation is left out.
infl = ols_results.get_influence()

# In general we may consider DBETAS in absolute value greater than 2/√N to be influential
observations
2./len(X)**.5
0.5

print(infl.summary_frame().filter(regex="dfb"))
    dfb_const  dfb_GNPDEFL       dfb_GNP     dfb_UNEMP     dfb_ARMED  \
0   -0.016406  -169.822675  1.673981e+06  54490.318088  51447.824036
1   -0.020608  -187.251727  1.829990e+06  54495.312977  52659.808664
2   -0.008382   -65.417834  1.587601e+06  52002.330476  49078.352379
3    0.018093   288.503914  1.155359e+06  56211.331922  60350.723082
4    1.871260  -171.109595  4.498197e+06  82532.785818  71034.429294
5   -0.321373  -104.123822  1.398891e+06  52559.760056  47486.527649
6    0.315945  -169.413317  2.364827e+06  59754.651394  50371.817827
7    0.015816   -69.343793  1.641243e+06  51849.056936  48628.749338
8   -0.004019   -86.903523  1.649443e+06  52023.265116  49114.178265
9   -1.018242  -201.315802  1.371257e+06  56432.027292  53997.742487
10   0.030947   -78.359439  1.658753e+06  52254.848135  49341.055289
11   0.005987  -100.926843  1.662425e+06  51744.606934  48968.560299
12  -0.135883   -32.093127  1.245487e+06  50203.467593  51148.376274
13   0.032736   -78.513866  1.648417e+06  52509.194459  50212.844641
14   0.305868   -16.833121  1.829996e+06  60975.868083  58263.878679
15  -0.538323   102.027105  1.344844e+06  54721.897640  49660.474568

          dfb_POP      dfb_YEAR
0   207954.113588 -31969.158503
1    25343.938289 -29760.155888
2   107465.770565 -29593.195253
3   456190.215133 -36213.129569
4  -389122.401699 -49905.782854
5   144354.586054 -28985.057609
6  -107413.074918 -32984.462465
7    92843.959345 -29724.975873
8    83931.635336 -29563.619222
9    18392.575057 -29203.217108
10   93617.648517 -29846.022426
11   95414.217290 -29690.904188
12  258559.048569 -29296.334617
13  104434.061226 -30025.564763
14  275103.677859 -36060.612522
15 -110176.960671 -28053.834556
```

WLS

Weighted Least Squares

```
%matplotlib inline

from __future__ import print_function
import numpy as np
from scipy import stats
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.sandbox.regression.predstd import wls_prediction_std
from statsmodels.iolib.table import (SimpleTable, default_txt_fmt)
np.random.seed(1024)
```

人工数据：异方差2组

模型假定：

- 误设：真实模型是二次的，估计只是线性的
- 独立的噪声/误差项
- 两组误差方差，低和高方差组

```
nsample = 50
x = np.linspace(0, 20, nsample)
X = np.column_stack((x, (x - 5)**2))
X = sm.add_constant(X)
beta = [5., 0.5, -0.01]
sig = 0.5
w = np.ones(nsample)
w[nsample * 6//10:] = 3
y_true = np.dot(X, beta)
e = np.random.normal(size=nsample)
y = y_true + sig * w * e
X = X[:,[0,1]]
```

WLS知道异方差的真实方差比

在此示例中，`w`是误差项的标准差。 `WLS`要求权重与误差方差的倒数成比例。

```
mod_wls = sm.WLS(y, X, weights=1./(w ** 2))
res_wls = mod_wls.fit()
print(res_wls.summary())
```

```
                            WLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.927
Model:                            WLS   Adj. R-squared:                  0.926
Method:                 Least Squares   F-statistic:                     613.2
Date:                Mon, 14 May 2018   Prob (F-statistic):           5.44e-29
Time:                        21:44:50   Log-Likelihood:                -51.136
No. Observations:                  50   AIC:                             106.3
Df Residuals:                      48   BIC:                             110.1
```

```
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.2469      0.143     36.790      0.000       4.960       5.534
x1             0.4466      0.018     24.764      0.000       0.410       0.483
==============================================================================
Omnibus:                        0.407   Durbin-Watson:                   2.317
Prob(Omnibus):                  0.816   Jarque-Bera (JB):                0.103
Skew:                          -0.104   Prob(JB):                        0.950
Kurtosis:                       3.075   Cond. No.                         14.6
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

OLS vs. WLS

估计一个OLS模型用于比较：

```
res_ols = sm.OLS(y, X).fit()
print(res_ols.params)
print(res_wls.params)
[5.24256099 0.43486879]
[5.24685499 0.44658241]
```

将WLS标准误差与异方差校正的OLS标准误差进行比较：

```
se = np.vstack([[res_wls.bse], [res_ols.bse], [res_ols.HC0_se],
                [res_ols.HC1_se], [res_ols.HC2_se], [res_ols.HC3_se]])
se = np.round(se,4)
colnames = ['x1', 'const']
rownames = ['WLS', 'OLS', 'OLS_HC0', 'OLS_HC1', 'OLS_HC3', 'OLS_HC3']
tabl = SimpleTable(se, colnames, rownames, txt_fmt=default_txt_fmt)
print(tabl)
=====================
          x1   const
---------------------
WLS     0.1426 0.018
OLS     0.2707 0.0233
OLS_HC0 0.194  0.0281
OLS_HC1 0.198  0.0287
OLS_HC3 0.2003 0.029
OLS_HC3 0.207   0.03
---------------------
```

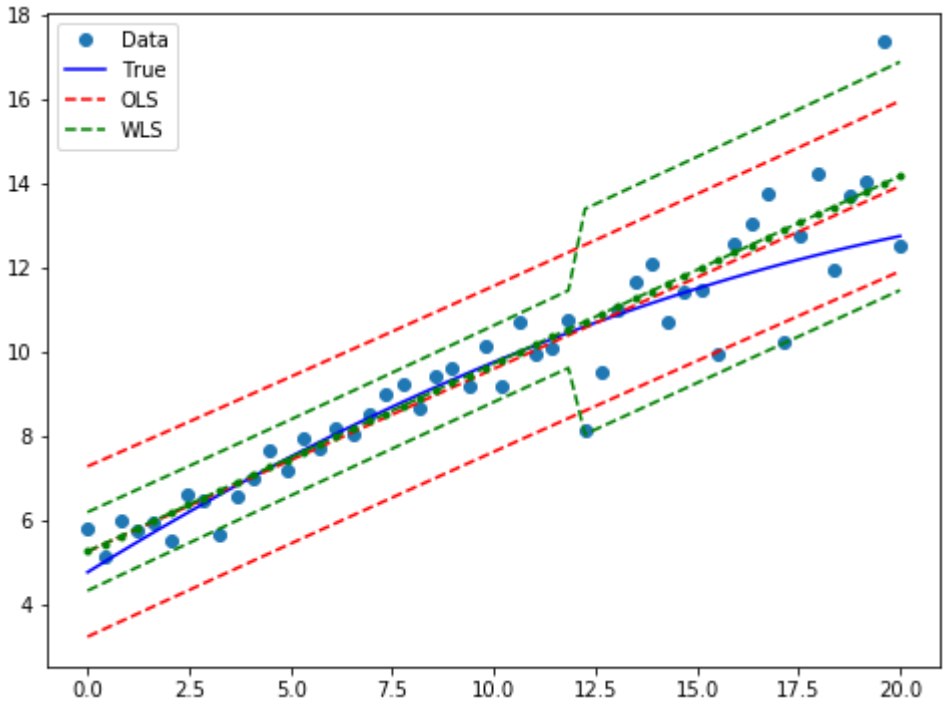
计算OLS预测 区间：

```
covb = res_ols.cov_params()
prediction_var = res_ols.mse_resid + (X * np.dot(covb,X.T).T).sum(1)
prediction_std = np.sqrt(prediction_var)
tppf = stats.t.ppf(0.975, res_ols.df_resid)
```

```
prstd_ols, iv_l_ols, iv_u_ols = wls_prediction_std(res_ols)
```

作图比较WLS 和OLS预测值：

```
prstd, iv_l, iv_u = wls_prediction_std(res_wls)

fig, ax = plt.subplots(figsize=(8,6))
ax.plot(x, y, 'o', label="Data")
ax.plot(x, y_true, 'b-', label="True")
# OLS
ax.plot(x, res_ols.fittedvalues, 'r--')
ax.plot(x, iv_u_ols, 'r--', label="OLS")
ax.plot(x, iv_l_ols, 'r--')
# WLS
ax.plot(x, res_wls.fittedvalues, 'g--.')
ax.plot(x, iv_u, 'g--', label="WLS")
ax.plot(x, iv_l, 'g--')
ax.legend(loc="best");
```



Feasible Weighted Least Squares (2-stage FWLS)

就像， `w` ， `w_est` 与标准差成比例，因此必须进行平方。

```
resid1 = res_ols.resid[w==1.]
```

```
var1 = resid1.var(ddof=int(res_ols.df_model)+1)
resid2 = res_ols.resid[w!=1.]
var2 = resid2.var(ddof=int(res_ols.df_model)+1)
w_est = w.copy()
w_est[w!=1.] = np.sqrt(var2) / np.sqrt(var1)
res_fwls = sm.WLS(y, X, 1./((w_est ** 2))).fit()
print(res_fwls.summary())
```

```
                            WLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.931
Model:                            WLS   Adj. R-squared:                  0.929
Method:                 Least Squares   F-statistic:                     646.7
Date:                Mon, 14 May 2018   Prob (F-statistic):           1.66e-29
Time:                        21:44:51   Log-Likelihood:                -50.716
No. Observations:                  50   AIC:                             105.4
Df Residuals:                      48   BIC:                             109.3
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.2363      0.135     38.720      0.000       4.964       5.508
x1             0.4492      0.018     25.431      0.000       0.414       0.485
==============================================================================
Omnibus:                        0.247   Durbin-Watson:                   2.343
Prob(Omnibus):                  0.884   Jarque-Bera (JB):                0.179
Skew:                          -0.136   Prob(JB):                        0.915
Kurtosis:                       2.893   Cond. No.                         14.3
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

GLS

Generalized Least Squares

```
from __future__ import print_function
import statsmodels.api as sm
import numpy as np
from statsmodels.iolib.table import (SimpleTable, default_txt_fmt)
```

Longley数据集是一个时间序列数据集：

```
data = sm.datasets.longley.load()
data.exog = sm.add_constant(data.exog)
print(data.exog[:5])
```

让我们假设数据具有异方差，并且我们知道异方差的本质。然后我们可以定义 `sigma` 并使用它来为我们提供GLS模型。首先，我们将从OLS拟合中获得残差：

```
ols_resid = sm.OLS(data.endog, data.exog).fit().resid
```

假定误差项遵循带有趋势的AR（1）过程：

$\epsilon_i = \beta_0 + \rho\epsilon_{i-1} + \eta_i$

其中，$\eta \sim N(0, \Sigma^2)$

并且ρ

```
resid_fit = sm.OLS(ols_resid[1:], sm.add_constant(ols_resid[:-1])).fit()
print(resid_fit.tvalues[1])
print(resid_fit.pvalues[1])

-1.4390229839744777
0.17378444788789812
```

虽然我们没有强有力的证据证明误差项遵循AR（1）过程，但我们仍继续

```
rho = resid_fit.params[1]
```

我们知道，AR（1）过程意味着近邻具有更强的关系，因此我们可以通过使用toeplitz矩阵来给出这种结构。

```
from scipy.linalg import toeplitz

toeplitz(range(5))
array([[0, 1, 2, 3, 4],
       [1, 0, 1, 2, 3],
       [2, 1, 0, 1, 2],
       [3, 2, 1, 0, 1],
       [4, 3, 2, 1, 0]])
order = toeplitz(range(len(ols_resid)))
```

所以我们的误差协方差结构实际上是rho **order，它 定义了自相关结构

```
sigma = rho**order
gls_model = sm.GLS(data.endog, data.exog, sigma=sigma)
gls_results = gls_model.fit()
```

当然，在这种情况下确切的rho是未知的，所以使用可行的gls可能更有意义，目前只是为了实验说明。

我们可以使用具有一阶滞后的GLSAR模型来获得类似的结果：

```
glsar_model = sm.GLSAR(data.endog, data.exog, 1)
glsar_results = glsar_model.iterative_fit(1)
print(glsar_results.summary())
                        GLSAR Regression Results
==============================================================================
```

```
Dep. Variable:                      y   R-squared:                       0.996
Model:                           GLSAR   Adj. R-squared:                  0.992
Method:                 Least Squares   F-statistic:                     295.2
Date:                Mon, 14 May 2018   Prob (F-statistic):           6.09e-09
Time:                        21:45:02   Log-Likelihood:                -102.04
No. Observations:                  15   AIC:                             218.1
Df Residuals:                       8   BIC:                             223.0
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -3.468e+06   8.72e+05     -3.979      0.004   -5.48e+06   -1.46e+06
x1            34.5568     84.734      0.408      0.694    -160.840     229.953
x2            -0.0343      0.033     -1.047      0.326      -0.110       0.041
x3            -1.9621      0.481     -4.083      0.004      -3.070      -0.854
x4            -1.0020      0.211     -4.740      0.001      -1.489      -0.515
x5            -0.0978      0.225     -0.435      0.675      -0.616       0.421
x6          1823.1829    445.829      4.089      0.003     795.100    2851.266
==============================================================================
Omnibus:                        1.960   Durbin-Watson:                   2.554
Prob(Omnibus):                  0.375   Jarque-Bera (JB):                1.423
Skew:                           0.713   Prob(JB):                        0.491
Kurtosis:                       2.508   Cond. No.                     4.80e+09
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.8e+09. This might indicate that there are
strong multicollinearity or other numerical problems.
```

比较gls和glsar结果，我们发现参数估计值和参数估计的结果标准误存在一些小的差异。这可能与算法中的数值差异有关，例如由于longley数据集中的观测值数量较少时初始条件的处理差异。

```
print(gls_results.params)
print(glsar_results.params)
print(gls_results.bse)
print(glsar_results.bse)

[-3.79785490e+06 -1.27656454e+01 -3.80013250e-02 -2.18694871e+00
 -1.15177649e+00 -6.80535580e-02  1.99395293e+03]
[-3.46796063e+06  3.45567846e+01 -3.43410090e-02 -1.96214395e+00
 -1.00197296e+00 -9.78045986e-02  1.82318289e+03]
[6.70688699e+05 6.94308073e+01 2.62476822e-02 3.82393151e-01
 1.65252692e-01 1.76428334e-01 3.42634628e+02]
[8.71584052e+05 8.47337145e+01 3.28032450e-02 4.80544865e-01
 2.11383871e-01 2.24774369e-01 4.45828748e+02]
```

Recursive LS

递归最小二乘

递归最小二乘是普通最小二乘的扩展窗口版本(expanding window version)。除了递归计算回归系数的可用性之外，还递归计算残差构造统计数据以研究参数不稳定性。

`RLS`类允许递归残差的计算，并计算CUSUM和平方统计量的CUSUM。绘制这些带参考线的统计量可以表示出稳定参数的原假设在统计上的显著差异，这很容易直观地指示参数稳定性。

```
%matplotlib inline
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from pandas_datareader.data import DataReader

np.set_printoptions(suppress = True)
```

例1：铜

我们首先考虑铜数据集中的参数稳定性（描述如下）

```
print(sm.datasets.copper.DESCRLONG)

dta = sm.datasets.copper.load_pandas().data
dta.index = pd.date_range('1951-01-01', '1975-01-01', freq='AS')
endog = dta['WORLDCONSUMPTION']

# To the regressors in the dataset, we add a column of ones for an intercept
exog = sm.add_constant(dta[['COPPERPRICE', 'INCOMEINDEX', 'ALUMPRICE', 'INVENTORYINDEX']])
```

首先，构建并完善模型，打印出摘要。`RLS`模型递归地计算回归参数，因此存在与数据点一样多的估计值，摘要表仅显示在整个样本上估计的回归参数; 除了初始化递归的小影响外，这些估计等同于OLS估计。

```
mod = sm.RecursiveLS(endog, exog)
res = mod.fit()
print(res.summary())
```

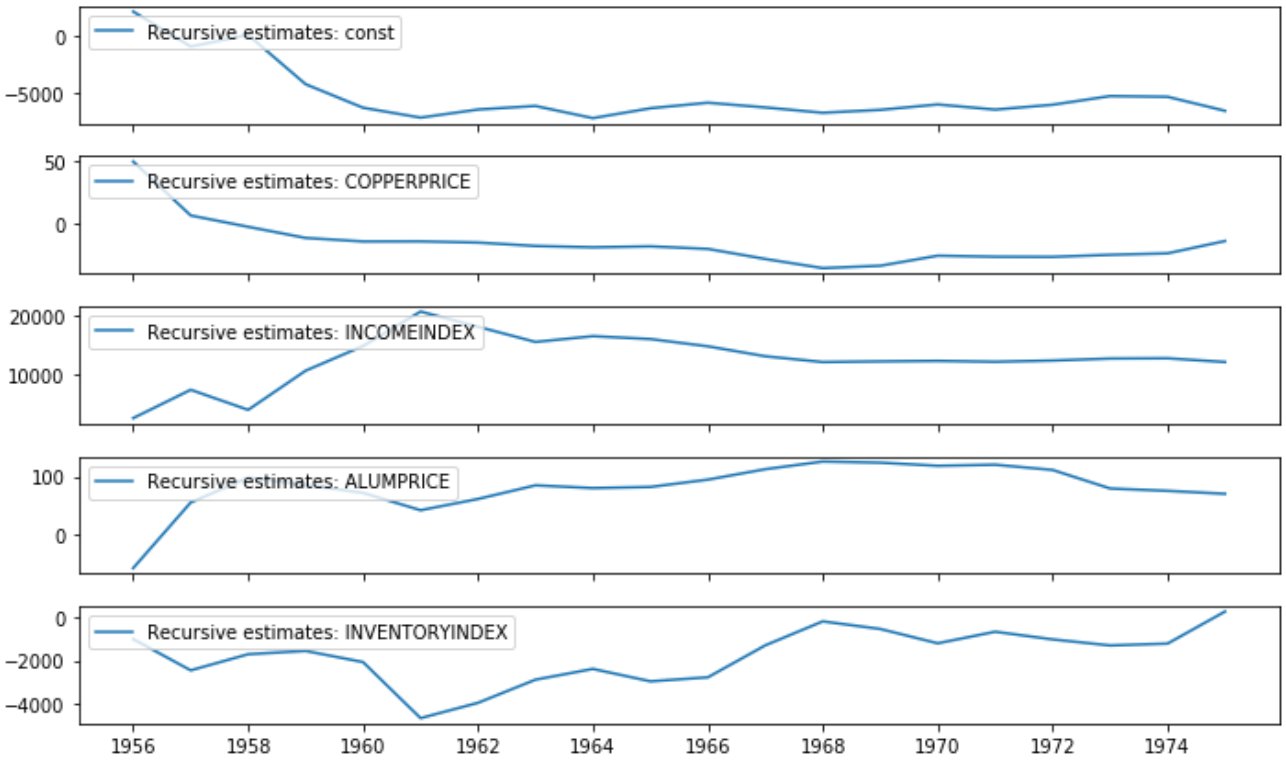
```
                         Statespace Model Results
==============================================================================
Dep. Variable:       WORLDCONSUMPTION   No. Observations:                   25
Model:                   RecursiveLS   Log Likelihood                -153.737
Date:                Mon, 14 May 2018   AIC                            317.474
Time:                        21:45:18   BIC                            322.453
Sample:                    01-01-1951   HQIC                           318.446
                         - 01-01-1975
Covariance Type:             nonrobust
==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          -6513.9922   2367.666     -2.751      0.006   -1.12e+04   -1873.452
COPPERPRICE      -13.6553     15.035     -0.908      0.364     -43.123      15.812
INCOMEINDEX     1.209e+04    762.595     15.853      0.000    1.06e+04    1.36e+04
ALUMPRICE         70.1441     32.667      2.147      0.032       6.117     134.171
INVENTORYINDEX   275.2805   2120.286      0.130      0.897   -3880.403    4430.964
```

```
================================================================================
Ljung-Box (Q):                       14.53   Jarque-Bera (JB):                 1.91
Prob(Q):                              0.75   Prob(JB):                         0.39
Heteroskedasticity (H):               3.48   Skew:                            -0.74
Prob(H) (two-sided):                  0.12   Kurtosis:                         2.65
================================================================================
Warnings:
[1] Parameters and covariance matrix estimates are RLS estimates conditional on the entire
sample.
```

递归系数通过`recursive_coefficients`属性可以获得。此外，可使用`plot_recursive_coefficient`方法作图。

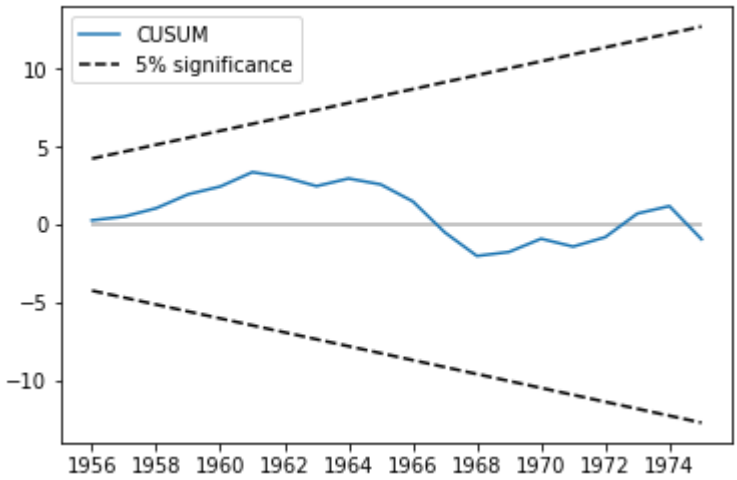
```
print(res.recursive_coefficients.filtered[0])
res.plot_recursive_coefficient(range(mod.k_exog), alpha=None, figsize=(10,6));
```

```
[2.88890056 4.94425552 1505.27013958 1856.55145415
  1597.98710024 2171.9827817 -889.38268842 122.17365406
 -4184.26788419 -6242.7260132 -7111.4525403 -6400.38238234
 -6090.45417093 -7154.9661487 -6290.92444686 -5805.25744629
 -6219.31774195 -6684.49621649 -6430.13809589 -5957.57738675
 -6407.05966649 -5983.49282153 -5224.71693423 -5286.62151094
 -6513.99218247]
```



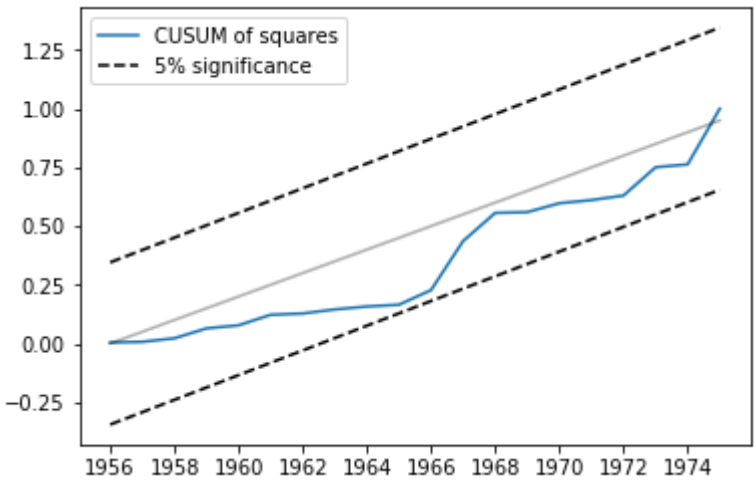
CUSUM统计量在`cusum`属性中可获得，但通常使用`plot_cusum`方法观察参数稳定性更方便。在下图中，CUSUM统计量不会超出5%显着性带（significance bands），因此我们未能拒绝参数在5%水平稳定的零假设。

```
print(res.cusum)
fig = res.plot_cusum()

[0.27819036 0.51898777 1.05399613 1.94931229 2.44814579 3.37264501
  3.04780124 2.47333616 2.96189072 2.58229248 1.49434496 -0.50007844
 -2.01111974 -1.75502595 -0.90603272 -1.41035323 -0.80382506 0.71254655
  1.19152452 -0.93369337]
```



另一个相关的统计量是平方CUSUM。通过`cusum_squares`属性中可获得，但使用`plot_cusum_squares`方法直接观察同样更方便。在下图中，平方统计量的CUSUM不会移动到5%显着性带之外，因此我们无法在5%水平拒绝参数稳定的零假设。

```
res.plot_cusum_squares()
```



货币数量理论

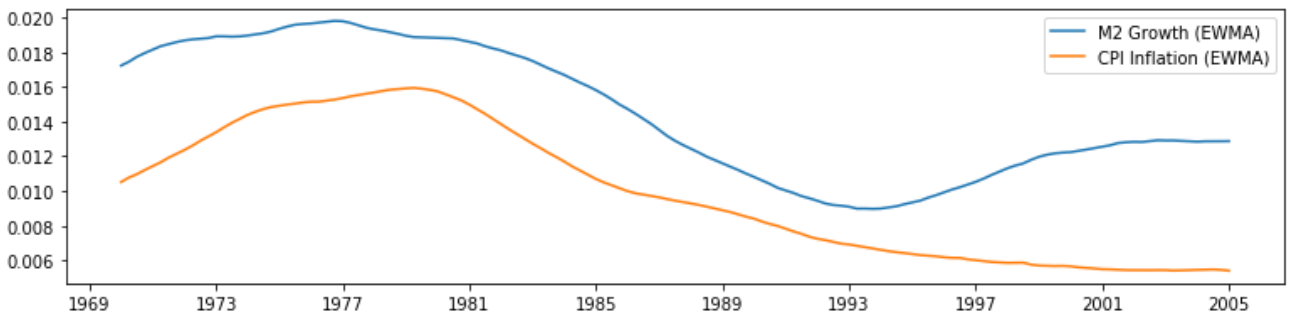
货币数量理论表明，“货币数量变化率的给定变化会导致价格通胀率的同等变化”（Lucas，1980）。在卢卡斯之后，我们研究了货币增长的双边指数加权移动平均线与CPI通胀之间的关系。虽然卢卡斯发现这些变量之间的关系是稳定的，但最近似乎这种关系是不稳定的; 参见Sargent和Surico(2010)。

```
start = '1959-12-01'
end = '2015-01-01'
m2 = DataReader('M2SL', 'fred', start=start, end=end)
cpi = DataReader('CPIAUCSL', 'fred', start=start, end=end)
```

```
def ewma(series, beta, n_window):
    nobs = len(series)
    scalar = (1 - beta) / (1 + beta)
    ma = []
    k = np.arange(n_window, 0, -1)
    weights = np.r_[beta**k, 1, beta**k[::-1]]
    for t in range(n_window, nobs - n_window):
        window = series.iloc[t - n_window:t + n_window+1].values
        ma.append(scalar * np.sum(weights * window))
    return pd.Series(ma, name=series.name, index=series.iloc[n_window:-n_window].index)

m2_ewma = ewma(np.log(m2['M2SL'].resample('QS').mean()).diff().iloc[1:], 0.95, 10*4)
cpi_ewma = ewma(np.log(cpi['CPIAUCSL'].resample('QS').mean()).diff().iloc[1:], 0.95, 10*4)
```

在使用$\beta = 0.95$ 构建移动平均线之后

```
fig, ax = plt.subplots(figsize=(13,3))
ax.plot(m2_ewma, label='M2 Growth (EWMA)')
ax.plot(cpi_ewma, label='CPI Inflation (EWMA)')
ax.legend();
```

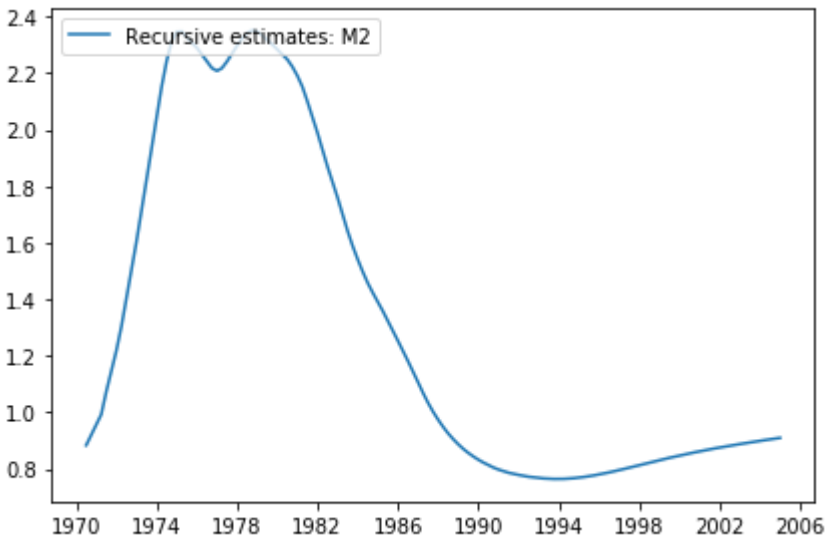



```
endog = cpi_ewma
exog = sm.add_constant(m2_ewma)
exog.columns = ['const', 'M2']

mod = sm.RecursiveLS(endog, exog)
res = mod.fit()
print(res.summary())
```

```
                         Statespace Model Results
==============================================================================
Dep. Variable:               CPIAUCSL   No. Observations:                  141
Model:                   RecursiveLS   Log Likelihood                 686.267
Date:                Mon, 14 May 2018   AIC                          -1368.535
Time:                        21:45:23   BIC                          -1362.666
```

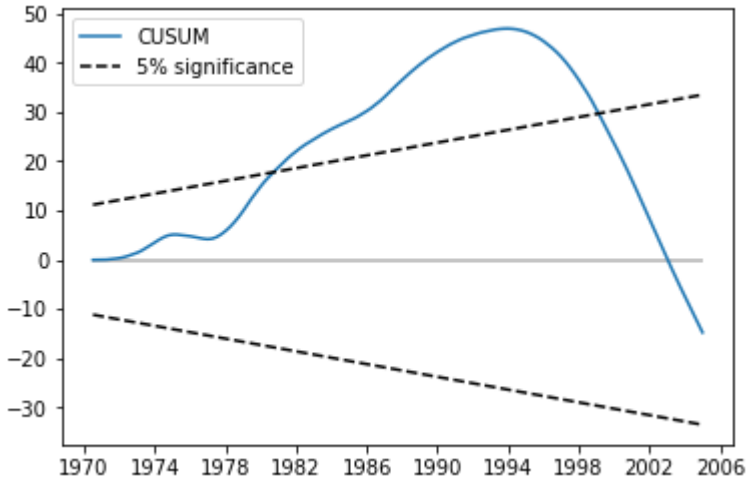
```
Sample:                    01-01-1970   HQIC                          -1366.150
                         - 01-01-2005
Covariance Type:             nonrobust
================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
const         -0.0033      0.001     -5.935      0.000      -0.004      -0.002
M2             0.9098      0.037     24.597      0.000       0.837       0.982
===================================================================================
Ljung-Box (Q):                     1857.42   Jarque-Bera (JB):                18.30
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               5.30   Skew:                            -0.81
Prob(H) (two-sided):                  0.00   Kurtosis:                         2.29
===================================================================================
Warnings:
[1] Parameters and covariance matrix estimates are RLS estimates conditional on the entire
sample.
```

```
res.plot_recursive_coefficient(1, alpha=None)
```



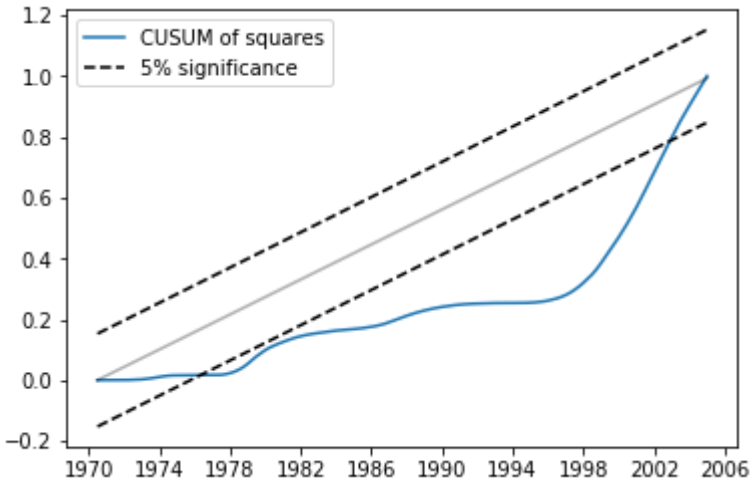
CUSUM图现在显示在5%水平的实质偏差，表明拒绝参数稳定的零假设。

```
res.plot_cusum()
```



类似地，平方CUSUM显示出在5%水平的实质偏差，也表明拒绝参数稳定的零假设。

```
res.plot_cusum_squares()
```



技术文档

统计模型被假定为

> $Y = X\beta + \mu, \text{其中} \mu \sim N(0, \Sigma)$

取决于属性 Σ(即方差-协方差矩阵），我们目前有四个类：

- GLS：适用任意协方差的广义最小二乘法
- OLS：误差项服从独立同分布的普通最小二乘法 ($\Sigma = I$)
- WLS：误差项具有异方差的加权最小二乘法 ($diag(\Sigma)$)
- GLSAR：具有p阶自相关AR(p)误差的可行广义最小二乘法 ($\Sigma = \Sigma(\rho)$)

所有回归模型都定义了相同的方法并遵循相同的结构，并且可以以类似的方式使用。差别在于其中一些模型包含特定的方法和属性。

除RecursiveLS之外，GLS是其他回归类的超类。

参考文献

回归模型的一般参考：

```
* D.C. Montgomery and E.A. Peck. “Introduction to Linear Regression Analysis.” 2nd. Ed., Wiley, 1992.
```

计量经济学回归模型参考：

- R.Davidson and J.G. MacKinnon. “Econometric Theory and Methods,” Oxford, 2004.
- W.Green. “Econometric Analysis,” 5th ed., Pearson, 2003.

属性

以下更详细地描述了所有回归类最常见的属性

pinv_wexog：*array*

p行n列(pxn)白化设计矩阵的摩尔-彭罗斯伪逆(Moore-Penrose pseudoinverse)，它近似等于$(X^T\Sigma^{-1}X)^{-1}X^T\Psi$，其中$\Psi$定义为$\Psi\Psi^T = \Sigma^{-1}$。

cholsimgainv: *array*

n阶上三角矩阵Ψ^T，满足$\Psi\Psi^T = \Sigma^{-1}$。

df_model： *float*

模型的自由度。等于 *p - 1*，其中*p*是回归元的数量。请注意，这里截距项被计算为不使用自由度。

df_resid： *float*

残差自由度。等于 *n - p*，其中 *n*是观测数，*p*是参数数。请注意，这里截距被计算为使用了一个自由度。

llf： *float*

拟合模型的似然函数值。

nobs: *float*

观测值的个数n。

normalized_cov_params： *array*

一个 p x p 数组，等于$(X^T\Sigma^{-1}X)^{-1}$。

sigma: *array*

误差项的n阶协方差矩阵：$\mu \sim N(0, \Sigma)$。

wexog: *array*

白化设计矩阵$\Psi^T X$。

wendog: *array*

白化响应变量$\Psi^T Y$。

模块参考

模型类

模型类	描述
OLS(endog[, exog, missing, hasconst])	一个简单最小二乘模型
GLS(endog,exog[,sigma,missing,hasconst])	具有一般协方差结构的广义最小二乘模型
WLS(endog,exog[,weights,missing,hasconst])	对角元素非一致协方差结构的回归模型
GLSAR(endog[, exog, rho, missing])	具有AR(p)协方差结构的回归模型
yule_walker(X[,order,method,df,inv,demean])	使用Yule-Walker方程估计序列X中的AR(p)参数
QuantReg(endog, exog, **kwargs)	分位数回归
RecursiveLS(endog, exog, **kwargs)	递归最小二乘

结果类

结果类	**拟合一个线性回归模型返回的结果类。与其他线性模型的结果类相比，OLS具有一些特定的结果类和方法**
RegressionResults(model,params[，...])	该类总结了线性回归模型的拟合
OLSResults(model,params[, ...])	OLS模型的结果类
PredictionResults(predicted_mean，...[, df, ...])	预测结果类
QuantRegResults(model, params[, ...])	分位数回归模型的结果实例
RecursiveLSResults(model,params,filter_results)	递归最小二乘模型的结果类

广义线性模型

广义线性模型目前支持使用单参数指数族(one-parameter exponential families)进行估计。

有关命令和参数，请参阅模块参考。

示例

```
# Load modules and data
In [1]: import statsmodels.api as sm

In [2]: data = sm.datasets.scotland.load()
In [3]: data.exog = sm.add_constant(data.exog)

# Instantiate a gamma family model with the default link function.
In [4]: gamma_model = sm.GLM(data.endog, data.exog, family=sm.families.Gamma())
In [5]: gamma_results = gamma_model.fit()
In [6]: print(gamma_results.summary())
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   32
Model:                            GLM   Df Residuals:                       24
```

```
Model Family:                  Gamma   Df Model:                            7
Link Function:         inverse_power   Scale:                       0.0035843
Method:                         IRLS   Log-Likelihood:                -83.017
Date:               Mon, 14 May 2018   Deviance:                      0.087389
Time:                       21:48:07   Pearson chi2:                    0.0860
No. Iterations:                    6   Covariance Type:             nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0178      0.011     -1.548      0.122      -0.040       0.005
x1          4.962e-05   1.62e-05      3.060      0.002    1.78e-05    8.14e-05
x2             0.0020      0.001      3.824      0.000       0.001       0.003
x3         -7.181e-05   2.71e-05     -2.648      0.008      -0.000   -1.87e-05
x4             0.0001   4.06e-05      2.757      0.006    3.23e-05       0.000
x5         -1.468e-07   1.24e-07     -1.187      0.235   -3.89e-07    9.56e-08
x6             -0.0005      0.000     -2.159      0.031      -0.001   -4.78e-05
x7         -2.427e-06   7.46e-07     -3.253      0.001   -3.89e-06   -9.65e-07
==============================================================================
```

详细示例

- GLM
- Formula

技术文档

对每个观测 i 的统计模型 被假定为

> $Y_i \sim F_{EDM}(\cdot|\theta,\phi,w_i)$和 $\mu_i = E[Y_i|x_i] = g^{-1}(x'_i\beta)$

其中g 是联系函数，和$F_{EDM}(\cdot|\theta,\phi,w)$ 是一个具有自然参数θ，尺度参数(scale parameter)φ和权重w的指数扩散模型族（EDM）的分布。密度函数为：

> $f_{EDM}(y|\theta,\phi,w) = c(y,\phi,w)exp(\frac{y\theta-b(\theta)}{\phi}w)$

该密度遵循$\mu = b'(\theta)$和 $Var[Y|x] = \frac{\phi}{w}b''(\theta)$。第一个等式的逆给出了自然参数是期望值的函数$\theta(\mu)$ 。因此

> $Var[Y_i|x_i] = \frac{\phi}{w_i}v(\mu_i)$

由于$v(\mu) = b''(\theta(\mu))$，因此GLM由联系函数$g$和方差函数 $v(\mu)$单独决定（当然也由x决定）。

请注意，尽管 ϕ 对每个观测y_i都是一样的，不影响对 β的估计，但是权数w_i可能对每个观测y_i都不同，因此β的估计依赖于权数w_i。

Distribution	Domain	$\mu = E[Y\|x]$	$v(\mu)$	$\theta(\mu)$	$b(\theta)$	ϕ
Binomial $B(n,p)$	$0,1,\ldots,n$	np	$\mu - \frac{\mu^2}{n}$	$log\frac{p}{1-p}$	$nlog(1+e^{\theta})$	1
Poisson $P(\mu)$	$0,1,\ldots,\infty$	μ	μ	$log(\mu)$	e^θ	1
Neg. Binom. $NB(\mu,\alpha)$	$0,1,\ldots,\infty$	μ	$\mu+\alpha\mu^2$	$log(\frac{\alpha\mu}{1+\alpha\mu})$	$-\frac{1}{\alpha}log(1-\alpha e^{\theta})$	1
Gaussian/Normal $N(\mu,\sigma 2)$	$(-\infty,\infty)$	μ	1	μ	$\frac{1}{2}\theta^2$	σ^2
Gamma $N(\mu,\nu)$	$(0,\infty)$	μ	μ^2	$-\frac{1}{\mu}$	$-log(-\theta)$	$\frac{1}{\nu}$
Inv. Gauss. $IG(\mu,\sigma^2)$	$(0,\infty)$	μ	μ^3	$-\frac{1}{2\mu^2}$	$-\sqrt{-2\theta}$	σ^2
Tweedie $p \geq 1$	depends on p	μ	μ^p	$\frac{\mu^(1-p)}{1-p}$	$\frac{\alpha-1}{\alpha}(\frac{\theta}{\alpha-1})^{\alpha}$	ϕ

Tweedie分布具有特殊情形$p=0,1,2$，并没有在表中列出，并且使用 $\alpha=\frac{p-2}{p-1}$。

数学变量与代码的对应关系：

- Y 和y对应代码 `endog`，想要模型化的变量
- x 对应代码 `exog`，协变量也叫解释变量
- β对应代码 `params`，要估计的参数
- μ对应代码 `mu`，Y的期望值（以x为条件）
- g对应代码为 `class Family` 的 `link` 参数
- ϕ对应代码为 `scale`，EDM的分散参数(the dispersion parameter)
- w目前不支持（即 $w=1$），将来可能变为 `var_weights`
- p对应代码为 `var_power`，Tweedie分布方差函数 $v(\mu)$的幂，请参见表格
- α可以是
- 负二项(Negative Binomial)：辅助参数 `alpha`，见表
- Tweedie：$\frac{p-2}{p-1}$的缩写，是方差函数的幂 p，见表

参考文献

- Gill, Jeff. 2000. Generalized Linear Models: A Unified Approach. SAGE QASS Series.
- Green, PJ. 1984. “Iteratively reweighted least squares for maximum likelihood estimation, and some robust and resistant alternatives.” Journal of the Royal Statistical Society, Series B, 46, 149-192.
- Hardin, J.W. and Hilbe, J.M. 2007. “Generalized Linear Models and Extensions.” 2nd ed. Stata Press, College Station, TX.
- McCullagh, P. and Nelder, J.A. 1989. “Generalized Linear Models.” 2nd ed. Chapman & Hall, Boca Rotan.

模块参考

模型类

类	描述
`GLM`(endog, exog[, family, offset, exposure, ...])	广义线性模型类

结果类

GLMResults (model, params, ...[, cov_type, ...])	GLM结果类
PredictionResults (predicted_mean, var_pred_mean)	预测结果类

分布族(Families)

目前实行的分布族有

Family (link, variance)	单参数指数族的父类
Binomial ([link])	二项式指数族分布
Gamma ([link])	Gamma指数族分布
Gaussian ([link])	高斯指数族分布
InverseGaussian ([link])	逆高斯指数族
NegativeBinomial ([link, alpha])	负二项指数族
Poisson ([link])	泊松指数族
Tweedie ([link, var_power])	特威迪分布族(Tweedie family)

联系函数(Link Functions)

目前应用的联系函数如下。并非所有联系函数都适用于每个分布族。可使用的联系函数列表可通过如下方式获得

```
>>> sm.families.family.<familyname>.links
```

Link	单参数指数族的通用联系函数
CDFLink([dbn])	使用 scipy.stats分布的CDF
CLogLog	对数-对数转换
Log	对数转换
Logit	逻辑(logit)转换
NegativeBinomial([alpha])	负二项联系函数
Power([power])	指数（幂）转换
cauchy()	柯西（标准柯西CDF）转换
cloglog	CLogLog转换联系函数
identity()	特性（identity ）转换
inverse_power()	逆变换
inverse_squared()	逆平方变换
log	对数转换
logit	
nbinom([alpha])	负二项联系函数
probit([dbn])	probit（标准正态CDF）变换

广义估计方程

当观测值可能聚类相关但聚类间不相关时，可以使用广义估计方程估计广义面板线性模型、聚类或重复测量的数据。它也支持与GLM相同的单参数指数族的估计。

有关命令和参数，请参阅模块参考。

示例

下面使用“癫痫发作”的数据，说明具有交互相关的组内聚类(exchangeable correlation within clusters)泊松回归

```
In [1]: import statsmodels.api as sm
In [2]: import statsmodels.formula.api as smf

In [3]: data = sm.datasets.get_rdataset('epil', package='MASS').data
In [4]: fam = sm.families.Poisson()
In [5]: ind = sm.cov_struct.Exchangeable()
In [6]: mod = smf.gee("y ~ age + trt + base", "subject", data,
   ...:               cov_struct=ind, family=fam)
   ...:
In [7]: res = mod.fit()
In [8]: print(res.summary())
```

```
                               GEE Regression Results
==============================================================================
Dep. Variable:                          y   No. Observations:                  236
Model:                                GEE   No. clusters:                       59
Method:                   Generalized      Min. cluster size:                   4
                     Estimating Equations   Max. cluster size:                   4
Family:                          Poisson   Mean cluster size:                 4.0
Dependence structure:        Exchangeable   Num. iterations:                    51
Date:                    Mon, 14 May 2018   Scale:                          1.000
Covariance type:                  robust   Time:                         21:46:28
==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept            0.5730      0.361      1.589      0.112      -0.134       1.280
trt[T.progabide]    -0.1519      0.171     -0.888      0.375      -0.487       0.183
age                  0.0223      0.011      1.960      0.050    2.11e-06       0.045
base                 0.0226      0.001     18.451      0.000       0.020       0.025
==============================================================================
Skew:                          3.7823   Kurtosis:                      28.6672
Centered skew:                 2.7597   Centered kurtosis:             21.9865
==============================================================================
```

有几个使用notebook的GEE 例子参见维基百科: Wiki notebooks for GEE 。

参考文献

- KY Liang and S Zeger. “Longitudinal data analysis using generalized linear models”. Biometrika (1986) 73 (1): 13-22.
- S Zeger and KY Liang. “Longitudinal Data Analysis for Discrete and Continuous Outcomes”. Biometrics Vol. 42, No. 1 (Mar., 1986), pp. 121-130
- A Rotnitzky and NP Jewell (1990). “Hypothesis testing of regression parameters in semiparametric generalized linear models for cluster correlated data”, Biometrika, 77, 485-497.
- Xu Guo and Wei Pan (2002). “Small sample performance of the score test in GEE”.http://www.sph.umn.edu/faculty1/wp-content/uploads/2012/11/rr2002-013.pdf
- LA Mancl LA, TA DeRouen (2001). A covariance estimator for GEE with improved small-sample properties. Biometrics. 2001 Mar;57(1):126-34.

模块参考

模型类

`GEE`(endog，exog，groups[，time，family，...])	使用广义估计方程（GEE）的边际回归模型估计

结果类

`GEEResults`**(model, params, cov_params, scale)**	**该类总结了使用GEE的边际回归模型的拟合**
`GEEMargins`(results, args[, kwargs])	用于通过GEE进行回归模型拟合时估计边际效应。

依赖结构

目前实行的依赖结构是

`CovStruct` **([cov_nearest_method])**	**分组数据相关性和协方差结构的基类。**
`Autoregressive` ([dist_func])	一阶自回归工作依赖结构。
`Exchangeable` ()	一个可交换的工作依赖结构。
`GlobalOddsRatio` (endog_type)	估计具有序数或名义数据的GEE的全局几率比
`Independence` ([cov_nearest_method])	一个独立工作依赖结构
`Nested` ([cov_nearest_method])	一个嵌套的工作依赖

分布族（Families）

同GLM

联系函数（Link Function）

同GLM

稳健线性模型

稳健线性模型支持Norms下所列出的M估计量(M-estimators)。

有关命令和参数，请参阅模块参考。

示例

```
# Load modules and data
In [1]: import statsmodels.api as sm

In [2]: data = sm.datasets.stackloss.load()
In [3]: data.exog = sm.add_constant(data.exog)

# Fit model and print summary
In [4]: rlm_model = sm.RLM(data.endog, data.exog, M=sm.robust.norms.HuberT())
In [5]: rlm_results = rlm_model.fit()
In [6]: print(rlm_results.params)
[-41.0265   0.8294   0.9261  -0.1278]
```

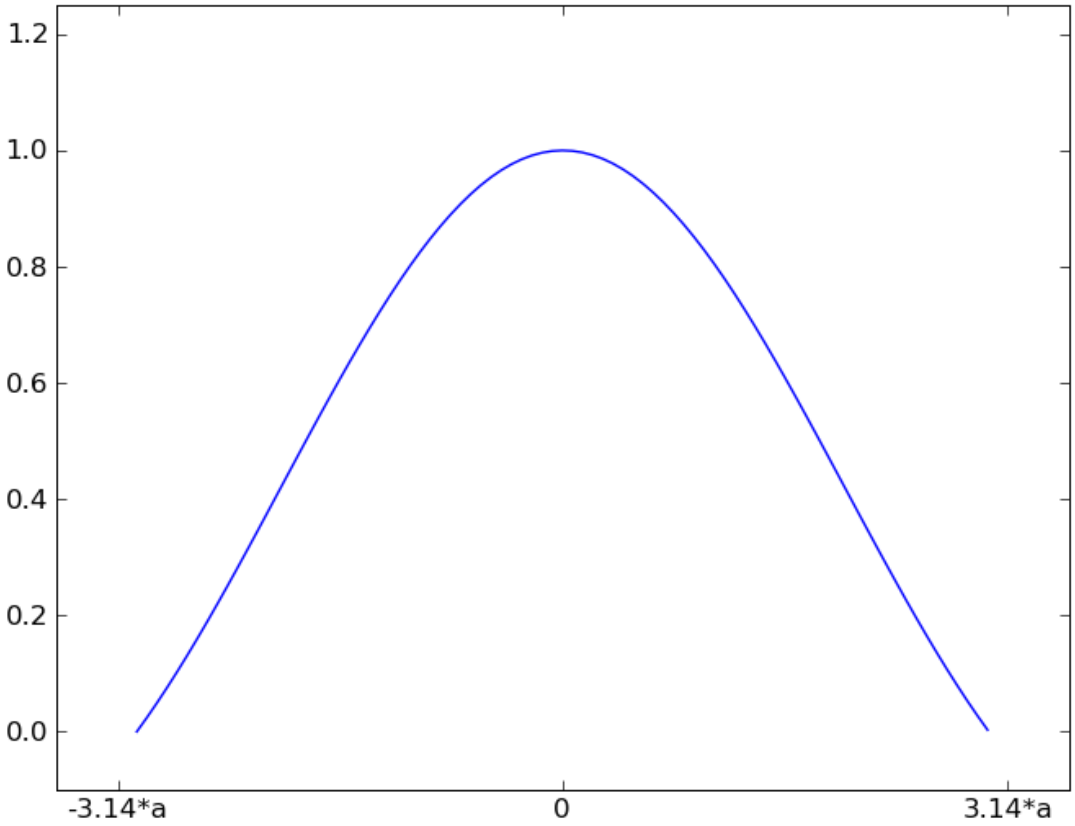
详细示例

- 稳健模型1
- 稳健模型2

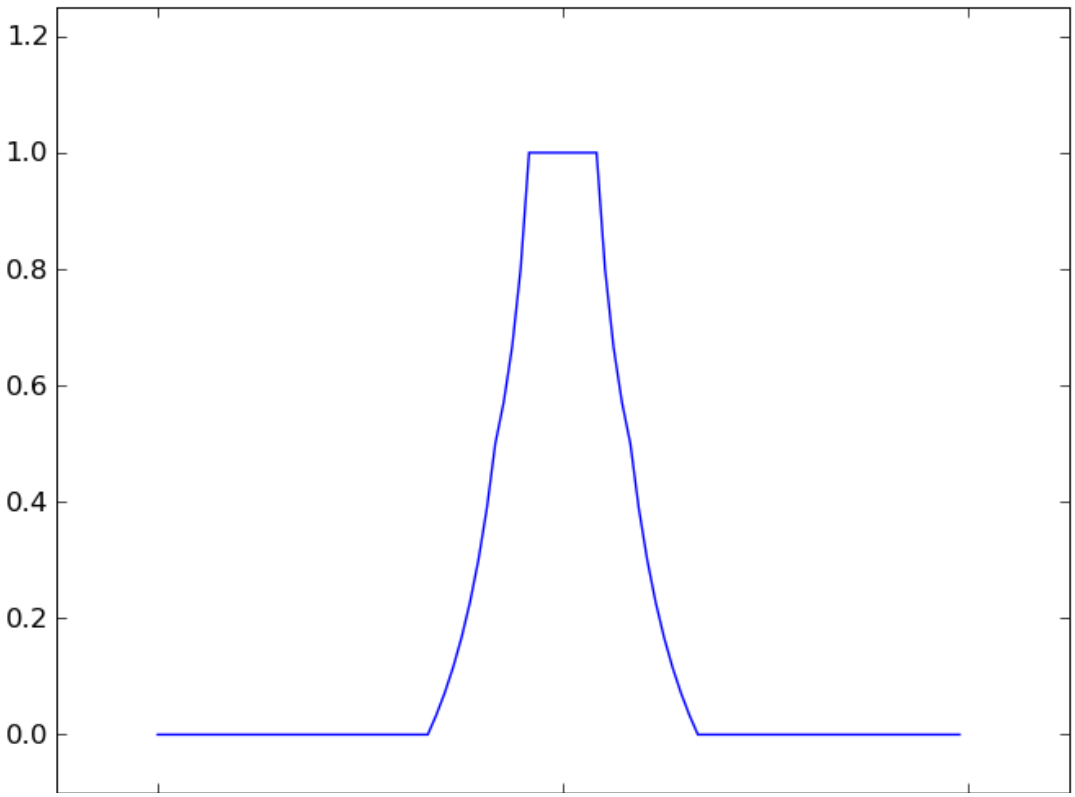
技术文档

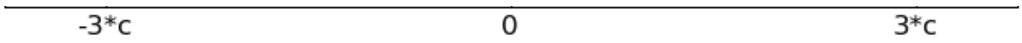
权重函数

Andrew's Wave

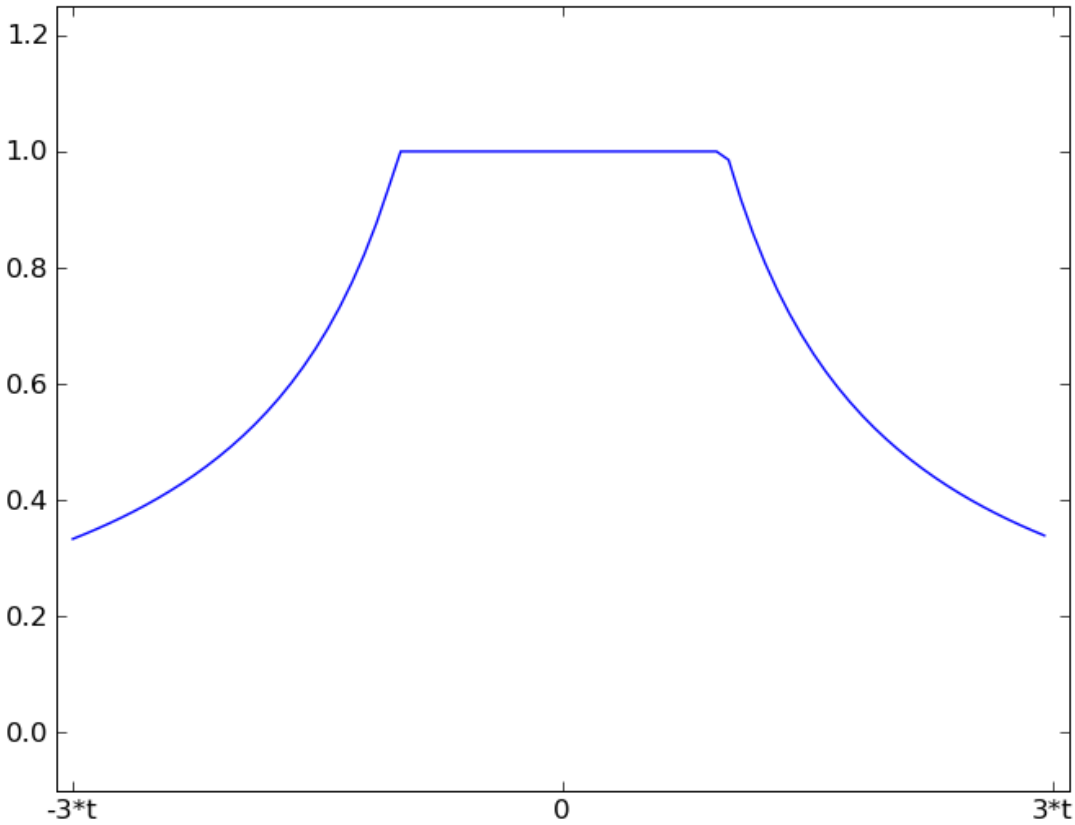


Hampel 17A

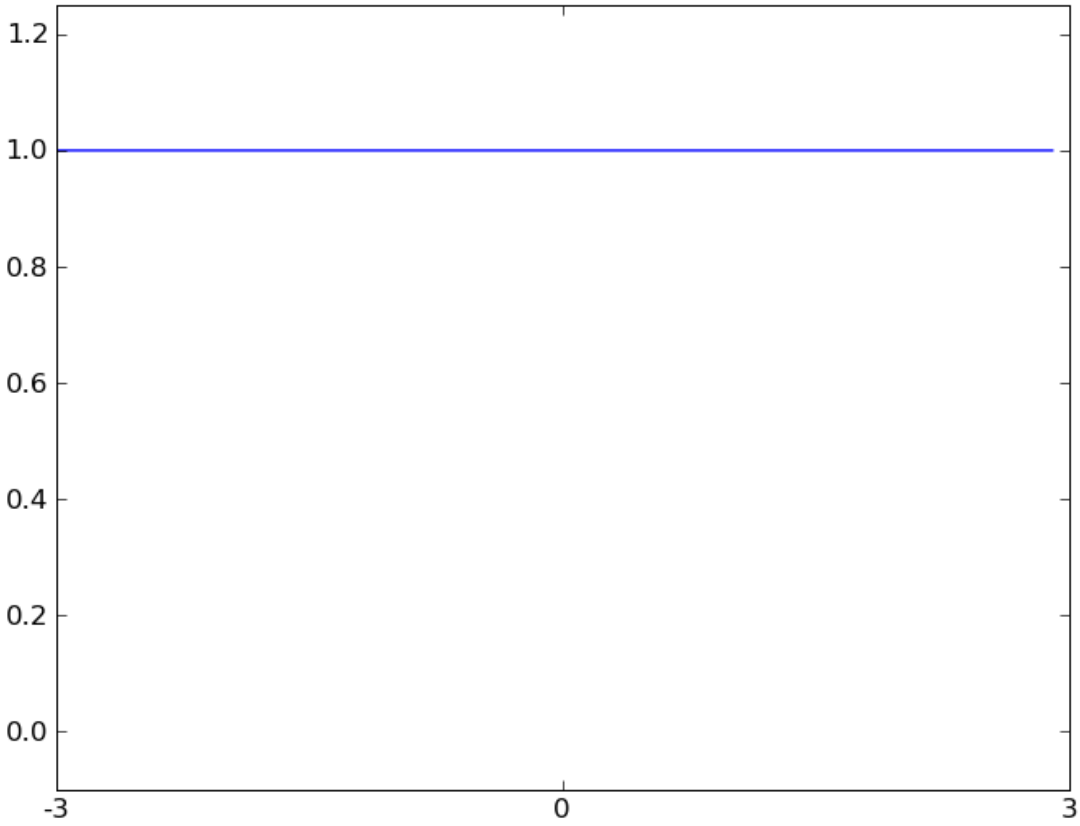




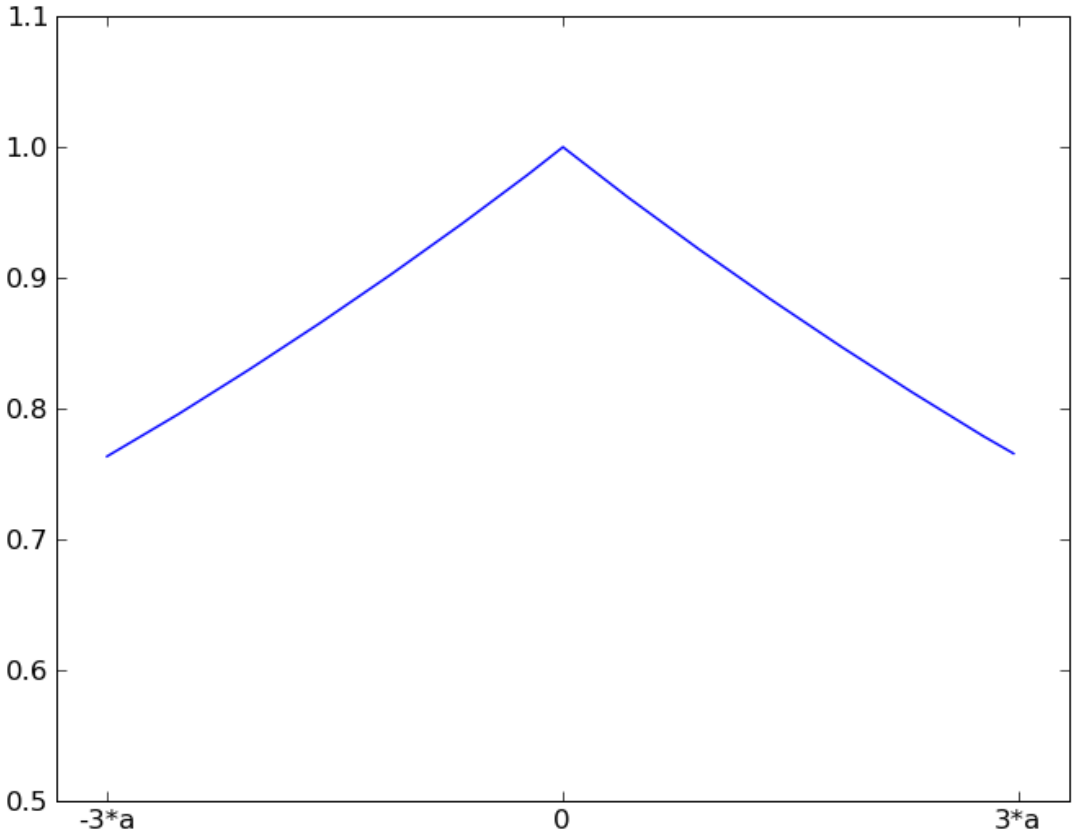
Huber’s t



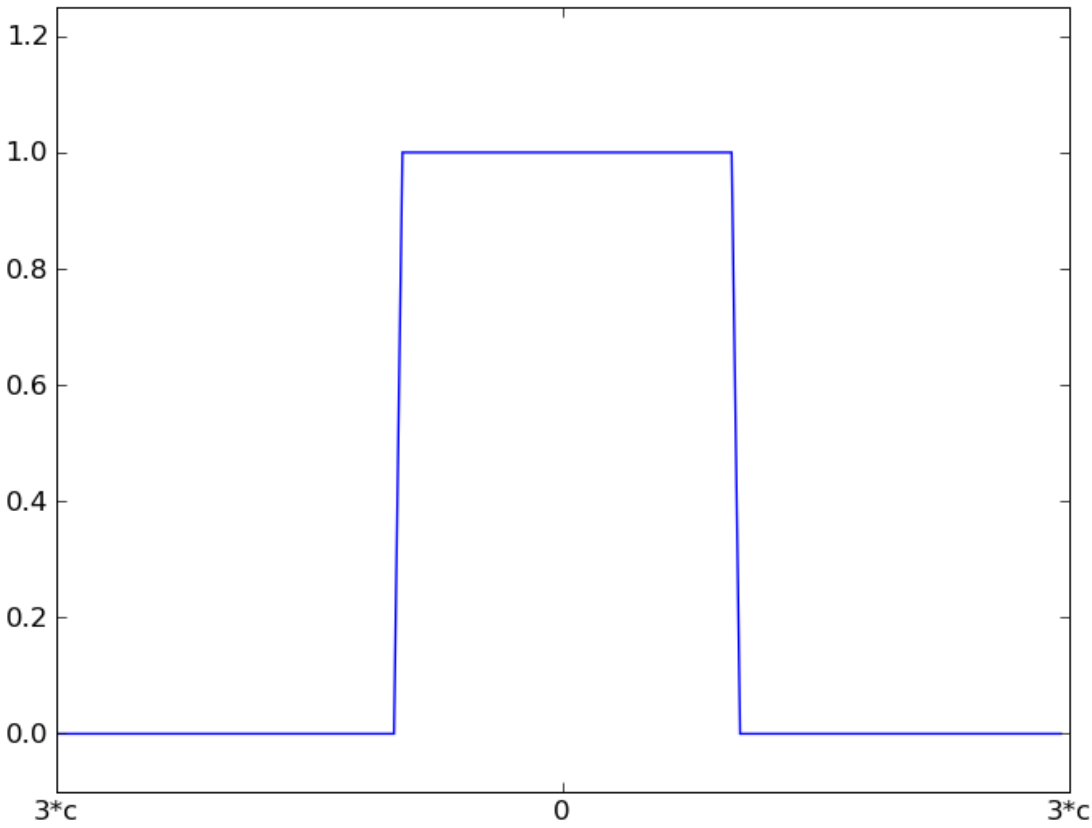
Least Squares



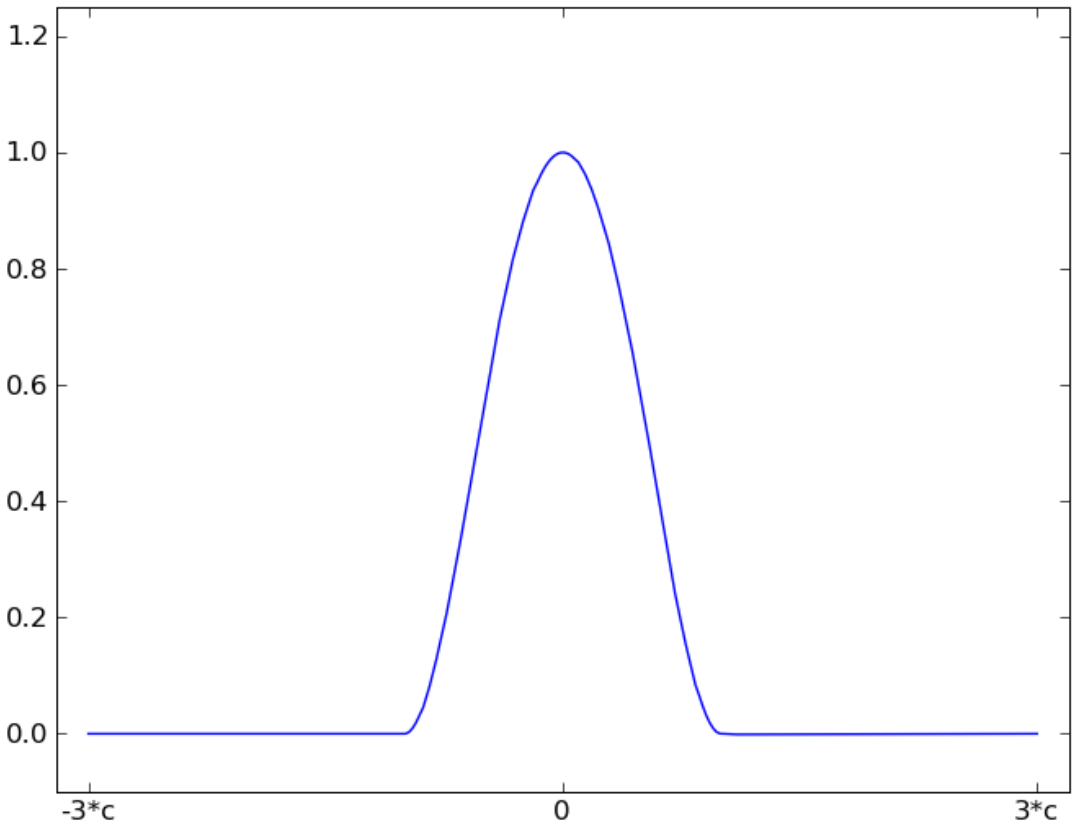
Ramsay’s Ea



Trimmed Mean



Tukey’s Biweight



参考文献

- PJ Huber. 'Robust Statistics' John Wiley and Sons, Inc., New York. 1981.
- PJ Huber. 1973, 'The 1972 Wald Memorial Lectures: Robust Regression: Asymptotics, Conjectures, and Monte Carlo.' The Annals of Statistics, 1.5, 799-821.
- R Venables, B Ripley. 'Modern Applied Statistics in S' Springer, New York

模块参考

模型类

`RLM` (endog, exog[, M, missing])	稳健的线性模型

模型结果类

`RLMResults` (model, params, ...)	RLM结果类

范数（Norms)

`AndrewWave` **([a])**	**Andrew's wave for M estimation.**
`Hampel` ([a, b, c])	Hampel function for M-estimation.
`HuberT` ([t])	Huber's T for M estimation.
`LeastSquares`	Least squares rho for M-estimation and its derived functions.
`RamsayE` ([a])	Ramsay's Ea for M estimation.
`RobustNorm`	The parent class for the norms used for robust regression.
`TrimmedMean` ([c])	Trimmed mean function for M-estimation.
`TukeyBiweight` ([c])	Tukey's biweight function for M-estimation.
`estimate_location` (a, scale[, norm, axis, ...])	M-estimator of location using self.norm and a current estimator of scale.

尺度（Scale）

`Huber` **([c, tol, maxiter, norm])**	**Huber's proposal 2 for estimating location and scale jointly.**
`HuberScale` ([d, tol, maxiter])	Huber's scaling for fitting robust linear models.
`mad` (a[, c, axis, center])	The Median Absolute Deviation along given axis of an array
`hubers_scale`	Huber's scaling for fitting robust linear models.

线性混合效应模型

线性混合效应模型用于涉及依赖数据(dependent data)的回归分析。这些数据产生于使用纵列和其他对每一个对象进行多次观测的研究设计。一些特定的线性混合效应模型是

- *随机截距模型*，组中的所有响应都会加上一个特定于该组的特定值。
- *随机斜率模型*，组中的响应遵循（条件）平均轨迹，该轨迹在观察到的协变量中是线性的，其中斜率（也可能有截距）随组别变化。
- *方差分量模型(Variance components models)*，其中一个或多个分类协变量的水平与分布提取(draws from distributions)相关联。这些随机项基于其协变量值累加地确定每个观测的条件均值。

Statsmodels的LME的实现主要是基于组的，意味着不同组中的响应，随机效应是独立实现的。在我们实现的混合模型中有两种类型的随机效应：（i）具有未知协方差矩阵的随机系数（可能是向量），以及（ii）独立于从常见单变量分布中提取的随机系数。对于（i）和（ii），随机效应通过一个组特有的设计矩阵的矩阵/向量乘积来影响组的条件均值。

如上面（i），随机系数的一个简单例子是：

$$Y_{ij} = \beta_0 + \beta_1 X_{ij} + \gamma_{0i} + \gamma_{1i} X_{ij} + \epsilon_{ij}$$

其中，Y_{ij}是个第i个对象的第j个测量响应，X_{ij} 是该响应的协变量。"固定效应参数"β_0和β_1是所有对象共有的，并且误差项ϵ_{ij}独立同分布（均值为零）。"随机效应参数"γ_{0i}和 γ_{1i}遵循均值为零的二元分布，由三个参数描述：$var(\gamma_{0i})$, $var(\gamma_{1i})$和 $cov(\gamma_{0i}, \gamma_{1i})$。$var(\epsilon_{ij})$也有一个参数。

如上面（ii），方差分量的简单示例是：

$$Y_{ijk} = \beta_0 + \eta_{1i} + \eta_{2j} + \epsilon_{ijk}$$

其中，Y_{ijk} 是在条件': $math: i, j$ '下的第k个测量响应 $k`thmeasuredresponseunderconditions: math: `i, j$ 。唯一的"平均结构参数"是β_0。η_{1i}是独立同分布的，具有零均值和方差 τ_1^2，且 η_{2j} 是独立同分布的，具有零均值和方差 τ_2^2。

Statsmodels MixedLM可以处理大多数非截面随机效应模型和一些截面模型。要在模型中包含截面随机效应，有必要将整个数据集视为单一组。然后可以使用模型的方差分量参数来定义具有各种截面和非截面随机效应组合的模型。

Statsmodels LME框架目前支持通过Wald检验和系数置信区间、概况(profile)似然分析、似然比检验和AIC进行估计后推断(post-estimation inference)。

示例

```
In [1]: import statsmodels.api as sm
In [2]: import statsmodels.formula.api as smf

In [3]: data = sm.datasets.get_rdataset("dietox", "geepack").data
In [4]: md = smf.mixedlm("Weight ~ Time", data, groups=data["Pig"])
In [5]: mdf = md.fit()
In [6]: print(mdf.summary())
         Mixed Linear Model Regression Results
=========================================================
Model:            MixedLM Dependent Variable: Weight
No. Observations: 861     Method:             REML
No. Groups:       72      Scale:              11.3669
Min. group size:  11      Likelihood:         -2404.7753
Max. group size:  12      Converged:          Yes
Mean group size:  12.0
---------------------------------------------------------
             Coef.  Std.Err.    z    P>|z| [0.025 0.975]
---------------------------------------------------------
Intercept    15.724    0.788  19.952 0.000 14.179 17.268
Time          6.943    0.033 207.939 0.000  6.877  7.008
Group Var    40.394    2.149
=========================================================
```

详细示例

- Mixed LM

一些notebook示例在维基百科上: Wiki notebooks for MixedLM

技术文档

数据被划分为不相交的组。组i的概率模型是:

$$Y = X\beta + Z_\gamma + Q_{1\eta_1} + \cdots + Q_{k\eta_k} + \epsilon$$

其中

- n_i是组 i 中观测值的数量

- Y 是一个 n_i 维响应向量
- X 是一个$n_i * k_{fe}$维固定效应系数矩阵
- β 是一个 k_fe 维固定效应斜率向量
- Z 是一个$n_i * k_{fe}$维随机效应系数矩阵
- γ是一个具有0均值和协方差矩阵Ψ的k_re维随机向量；注意每个组都有自己独立的γ实现。
- Q_j是一个 $n_i \times q_j$ 维的第j个方差分量的设计矩阵 。
- η_j是一个包含独立同分布值方差为τ_j^2的q_j维度随机向量。
- ϵ 是一个服从$N(0, \sigma^2)$的n_i维向量；ϵ值在组内和组间都是独立的。

Y, X, Q_j和 Z必须都能被观测。$\beta, \Psi,$和 σ^2使用ML或REML进行估计，且γ, η_j和 ϵ是随机的，以此定义概率模型。

边际平均结构(marginal mean structure)是 $E[Y|X,Z] = X * \beta E[Y|X,Z] = X * \beta$。如果只考虑边际平均结构，GEE是混合模型的一个很好的选择。

注释：

- cov_re是随机效应协方差矩阵（指上述 Ψ）和 $scale$是（标量）误差方差。对每个方差分量还有一个估计的方差参数τ_j^2。对单个组，给定exog，endog的边际协方差矩阵为$scale * I + Z * cov_re * Z$，其中$Z$ 是一组中随机效应的设计矩阵。

参考文献

实现细节的主要参考是：

- MJ Lindstrom, DM Bates (1988). *Newton Raphson and EM algorithms for linear mixed effects models for repeated measures data*. Journal of the American Statistical Association. Volume 83, Issue 404, pages 1014-1022.

另请参阅这个更新的文档：

- http://econ.ucsb.edu/~doug/245a/Papers/Mixed%20Effects%20Implement.pdf

所有似然值，梯度和海塞矩阵的计算都紧密遵循Lindstrom和Bates。

以下两个文档更多是从用户的角度撰写的：

- http://lme4.r-forge.r-project.org/lMMwR/lrgprt.pdf
- http://lme4.r-forge.r-project.org/slides/2009-07-07-Rennes/3Longitudinal-4.pdf

模块参考

模型类

`MixedLM`**(endog, exog, groups[, exog_re, ...])**	**指定线性混合效果模型的对象**

结果类

`MixedLMResults`**(model, params, cov_params)**	**拟合线性混合效应模型的结果类**

离散因变量回归

广义线性混合效应模型

方差分析

时间序列分析 `tsa`

状态空间方法的时间序列分析 `statespace`

向量自回归 `tsa.vector_ar`

生存和久期分析方法

统计 `stats`

非参数方法 `nonparametric`

广义的矩估计 `gmm`

列联表

链式方程多重插补

多元统计 `multivariate`

经验似然 `emplike`

其他模型 `miscmodels`

分布

图形

输入输出 `iolib`

`statsmodels` 提供了一些输入和输出的函数。包含STATA文件的读取，以多种格式生成并打印表格的类和两个序列化(for pickling)辅助函数。【***译注：*** 序列化过程即将文本信息转变为二进制数据流】

用户还可以利用pandas.io所提供的强大输入/输出功能。此外，`pandas`（`statsmodels`依赖项）允许读取和写入Excel，CSV和HDF5（PyTables）。

示例

SimpleTable: Basic example

同"线性回归"的详细示例-WLS

模块参考

statsmodels.iolib.foreign.StataReader

statsmodels.iolib.foreign.StataWriter

statsmodels.iolib.foreign.genfromdta

statsmodels.iolib.foreign.savetxt

statsmodels.iolib.table.SimpleTable

statsmodels.iolib.table.csv2st

statsmodels.iolib.smpickle.save_pickle

statsmodels.iolib.smpickle.load_pickle

statsmodels.iolib.summary.Summary

statsmodels.iolib.summary2.Summary

`foreign.StataReader` **(fname[, missing_values, ...])**	**Stata.dta文件阅读器**
`foreign.StataWriter` (fname, data[, ...])	一个用于从类似数组的对象中写入Stata二进制dta文件的类
`foreign.genfromdta` (fname [, missing_flt, ...])	从一个Stata .dta文件返回一个ndarray或DataFrame
`foreign.savetxt` (fname, X [, names, fmt, ...])	将数组保存为文本文件
`table.SimpleTable` (data [, headers, stubs, ...])	从*多维(rectangular)*（2d！）数据数组生成简单的ASCII，CSV，HTML或LaTeX表，不一定是数字
`table.csv2st` (csvfile[, headers, stubs, title])	返回SimpleTable实例，由csv文件中的数据创建，该数据为逗号分隔值格式
`smpickle.save_pickle` (obj, fname)	通过序列化将对象保存到文件
`smpickle.load_pickle` (FNAME)	从文件加载以前保存的对象

以下是用于返回估计结果摘要的类和函数，主要于内部使用。目前有两个版本用于创建摘要。

`summary.Summary` **()**	**该类结果摘要的呈现支持表格**
`summary2.Summary` ()	

指数和表