Asset pricing Euler equation tests using GMM

We study the implications of investor optimization for asset returns and use the resulting moment conditions to estimate parameters of investors' preferences.

Theory

Euler equations

We use data on asset returns and consumption growth to estimate preference parameters of representative investor using Euler equations of the form

$$E_t\left[\frac{S_{t+1}}{S_t}R^n_{t+1}\right] = 1, \qquad n = 1, \ldots, N$$

The stochastic discount factor equals the marginal rate of substitution of an investor with CRRA preferences

$$\frac{S_{t+1}}{S_t} = \beta\left(\frac{C_{t+1}}{C_t}\right)^{-\gamma}.$$

The representative agent assumption implies that C_t is the aggregate consumption process. Preferences parameters β and γ are to be estimated.

Some of the Euler equations may come in the form of excess returns. Specifically, take two Euler equations for returns R^n_{t+1} and R^p_{t+1} and subtract them to obtain

$$E_t\left[\frac{S_{t+1}}{S_t}\left(R^n_{t+1} - R^p_{t+1}\right)\right] = 0$$

The quantinty $R^n_{t+1} - R^p_{t+1}$ is an excess return, typically taken to be the difference between a risky and risk-free return.

Generalized method of moments

We can use instruments z^k_t to multiply the Euler equations and form unconditional moments

$$E\left[z_t^k E_t\left[\beta\left(\frac{C_{t+1}}{C_t}\right)^{-\gamma} R_{t+1}^n\right]\right] = E\left[z_t^k \beta\left(\frac{C_{t+1}}{C_t}\right)^{-\gamma} R_{t+1}^n\right] = 1.$$

Non-instrumented moments are a special case with $z_t^k = 1$. We have in total M such equations, including those for excess returns.

Denote $\theta = (\beta, \gamma)$ the vector of estimated parameters, with a unique value θ_0 for which the set of Euler equations holds exactly (identification assumption), and x_{t+1} the collection of data used in the Euler equation:

$$x_{t+1} = \left(C_{t+1}/C_t, R_{t+1}^1, \ldots, R_{t+1}^N, z_t^1, \ldots, z_t^K\right)$$

Further denote

$$f_m\left(x_{t+1}; \theta\right) = z_t^k \beta\left(\frac{C_{t+1}}{C_t}\right)^{-\gamma} R_{t+1}^n - 1.$$

or, for the case of excess returns,

$$f_m\left(x_{t+1}; \theta\right) = z_t^k \beta\left(\frac{C_{t+1}}{C_t}\right)^{-\gamma} \left(R_{t+1}^n - R_{t+1}^p\right).$$

Then the Euler equations can be written as

$$E\left[f_m\left(x_{t+1}; \theta\right)\right] = 0, \qquad m = 1, \ldots, M.$$

We can further create the $M \times 1$ column vector

$$f\left(x_{t+1}; \theta\right) = \left(f_1\left(x_{t+1}; \theta\right), \ldots, f_M\left(x_{t+1}; \theta\right)\right)',$$

and the resulting $M \times 1$ vector moment condition

$$E\left[f\left(x_{t+1}; \theta\right)\right] = 0.$$

Given the identification condition, for any positive definite matrix W, finding the solution to the moment conditions is equivalent to solving

$$\theta_0 = \min_\theta E\left[f\left(x_{t+1}; \theta\right)\right]' W E\left[f\left(x_{t+1}; \theta\right)\right].$$

In a finite data sample, we approximate the moment condition with

$$g_T(\theta) = \frac{1}{T}\sum_{t=0}^{T-1} f\left(x_{t+1}; \theta\right),$$

and hence we are solving

$$\hat{\theta}_T = \min_{\theta} g_T(\theta)' W g_T(\theta).$$

While any positive definite W will lead, under the identification restrictions, to a consistent estimate, efficiency of the estimates will depend on the choice of W. Hansen (1982) shows that when we choose the weighting matrix to be equal to the reciprocal of the long-run covariance matrix

$$V = \sum_{j=-\infty}^{\infty} E\left[f\left(x_{t+1}; \theta_0\right) f\left(x_{t+1+j}; \theta_0\right)'\right]$$

then the estimator is asymptotically efficient, and

$$T g_T\left(\theta_0\right)' V^{-1} g_T\left(\theta_0\right) \to \chi^2(M).$$

Estimation of the covariance matrix V

There is a range of issues involving the estimation of the long-run covariance matrix V.

1. The infinite sum in the expression for V takes into account temporal dependence of the data, see, for example, Newey and West (1987). For iid data, only the term for $j = 0$ is relevant. In our case, we exploit the fact that the moments are generated by conditional moment conditions, and hence

$$\text{for } j \geq 1: E\left[f\left(x_{t+1}; \theta_0\right) f\left(x_{t+1+j}; \theta_0\right)'\right] = E\left[f\left(x_{t+1}; \theta_0\right) E_{t+j}\left[f\left(\underbrace{x_{t+1+j}}_{=0}; \theta_0\right)'\right]\right] = 0$$

so the covariance matrix simplifies to

$$V = E\left[f\left(x_{t+1}; \theta_0\right) f\left(x_{t+1}; \theta_0\right)'\right] \approx \frac{1}{T}\sum_{t=0}^{T-1} f\left(x_{t+1}; \theta_0\right) f\left(x_{t+1}; \theta_0\right)'$$

2. The theoretical V is a function of the true parameter θ_0, which is a priori unknown. Hansen, Heaton, and Yaron (1996) propose a continuously updated

estimator

$$\hat{\theta}_T = \min_{\theta} g_T(\theta)' V_T(\theta)^{-1} g_T(\theta)$$

where

$$V_T(\theta) = \frac{1}{T} \sum_{t=0}^{T-1} f\left(x_{t+1}; \theta\right) f\left(x_{t+1}; \theta\right)'$$

3. A two-step procedure is asymptotically valid as well because V can be replaced by its consistent estimator. For example:

- compute $\hat{\theta}_T$ by minimizing the quadratic form

for some positive definite W

- compute $V_T\left(\hat{\theta}_T\right)$ using the empirical formula, properly accounting for the serial dependence if needed
- evaluate the left-hand-side of the $\chi^2(M)$ distributed test statistics using $\theta_0 = \hat{\theta}_T$

In principle, we could also use the estimated $V\left(\hat{\theta}_T\right)$ as a new W in the minimization of the quadratic form and obtain a new estimate $\hat{\theta}_T$.

4. In theory, using $V = V\left(\theta_0\right)$ as the weighting

matrix is asymptotically efficient. But there are obstacles:

- θ_0 must be estimated using a finite sample of data, so we

can at best use $V_T\left(\hat{\theta}_T\right)$ instead

- in practical applications, $V_T\left(\hat{\theta}_T\right)$ can be

hard to estimate, which may lead to fragility that puts excessive emphasis on a small number of moments

- this issue is further magnified when misspecifications are present
- asset pricing applications are prone to such fragility
- various aproaches exist how to modify the weighting matrix to deal

with these issues

For more discussion on GMM in the time-series context see, for example, Hansen (2001), Hansen (2008).

Data processing

We will use data on asset returns and consumption growth. Several considerations are important.

- We need to agree on the length of the time period t. Typical period length for this type of problem would be a month, a quarter, or a year. Below, we consider quarterly frequency. This means that all quantities need to be downloaded or converted to quarterly frequency.
- Because the theory applies to real quantities and data on asset returns typically come in nominal form, we will need to deflate them by an appropriate inflation index.
- Since there are alternative time series for consumption growth, inflation, and many different types of available returns, appropriate choice of these quantities often involves considerable judgement.

You can download all data below manually into a single spreadsheet which you use for pre-processing, and then load the prepared spreadsheet directly for the GMM estimation. Below, I conduct all the steps in Python.

```
In [ ]: # import some useful predefined functions from the course package
        from course import *
```

```
Done everything.
```

Consumption and inflation data from FRED

The Federal Reserve Bank of St. Louis maintains a large database of economic time series data (FRED - Federal Reserve Economic Data).

https://fred.stlouisfed.org/

Every time series is identified by a unique series identifier. Python provides API functionality in the **`pandas_datareader`** package to directly download the time series if you know the identifiers, or you can browse the database to find suitable data.

The theoretical moment conditions apply to real consumption. We could download nominal consumption expenditures and deflate them ourselves, or use one of the available real series. We do the latter. Here are some available series:

- **`PCEC`**: Personal consumption expenditures in billions of USD, seasonally adjusted (https://fred.stlouisfed.org/series/PCEC). This series is in nominal dollars but the theoretical moments are expressed in real consumption units, so we would need to deflate it by the proper price index.

- **`PCECC96`**: Personal consumption expenditures in billions of **chained 2012 USD**, seasonally adjusted (https://fred.stlouisfed.org/series/PCECC96). This series is already deflated to real terms. However, it expresses total consumption expenditures of the whole population, while the theory applies to an individual (even though representative) investor. We should therefore perhaps adjust for population growth, even though this will have negligible quantitative implications in our specific application.

- **`A794RX0Q048SBEA`**: Personal consumption expenditures **per capita** in **chained 2012 USD**, seasonally adjusted (https://fred.stlouisfed.org/series/A794RX0Q048SBEA).

- **`PCECTPI`**: Chain-type price index for personal consumption expenditures (https://fred.stlouisfed.org/series/PCECTPI). While we download consumption data already in real terms, we will need to deflate returns as well. Again, there are multiple price indices, the two most prominent ones being the Consumer Price Index (CPI), and the Personal Consumption Expenditures (PCE) Price Index, which are collected in alternative ways. We will use PCE to deflate returns, since the above consumption expenditures series are also deflated by PCE.

As you can see, there is a range of choices to be made when engaging in empirical work, some of which requiring judgement calls. For example, the differences between aggregate consumption and consumption per capita, or between using the CPI or PCE price indices will not have dramatic impacts for our conclusions but may be substantial in other applications.

In [ ]:
```
# personal consumption PCEC, real personal consumption PCECC96,
# real personal consumption per capita A794RX0Q048SBEA, PCE price index PCEC

# myLoadDataFRED uses the pandas_reader.data.Reader function to download tim
data_C = myLoadDataFRED(series=['PCEC','PCECC96','A794RX0Q048SBEA','PCECTPI'
```

In [ ]:
```
# create a YYYYQ (year, quarter) index for merging data with returns later
data_C = data_C['orig'].reset_index()
data_C['YYYYQ'] = data_C['DATE'].dt.year*10 + (data_C['DATE'].dt.month+2)//3
data_C = data_C.drop(columns=['DATE'])
data_C = data_C.set_index('YYYYQ')

# construct inflation time series ((P_{t}/P_{t-1}-1))
data_C['infl'] = data_C['PCECTPI'] + float("nan")
data_C['infl'][1:] =(data_C['PCECTPI'][1:].to_numpy() / data_C['PCECTPI'][0:

# construct real consumption growth per capita ((C_t/C_{t-1}-1))
```

```
data_C['gC'] = data_C['A794RX0Q048SBEA'] + float("nan")
data_C['gC'][1:] =(data_C['A794RX0Q048SBEA'][1:].to_numpy() / data_C['A794RX
data_C = data_C.drop(data_C.index[0])
#data_C = data_C.drop(data_C.index[data_C.index > 20200])
```

Returns data from Kenneth French's website

Kenneth French maintains a large database of asset returns widely used as standard test returns in asset pricing applications. These emerged from his early work from Eugene Fama on factor based asset pricing.

https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html

We will be using the most elementary dataset, containing the excess returns on three "Fama/French" factors:

https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/ftp/F-F_Research_Data_Factors_CSV.zip

These returns were originally used in Fama and French (1992), for a very brief introduction see the Wikipedia page on the Fama-French three-factor model.

The data set contains the following monthly returns, and their annual accumulated versions:

- **RF**: the nominal "risk-free" rate, equivalent to the yield on the U.S. one-month Treasury security
- **Mkt-RF**: the excess return on a broadly constructed U.S. stock portfolio, net of the risk-free rate
- **SMB**: the excess return on a portfolio of small U.S. firms, net of the return on a portfolio of large U.S. firms
- **HML**: the excess return on a portfolio of U.S. firms with high ratio of book valuation to market valuation, net of the return on a portfolio of U.S. firms with low ratio of book valuation to market valuation

We will be using the risk-free rate and the return on the aggregate stock market, the latter constructed from **Mkt-RF** by adding back the risk-free rate.

In our theory, the returns are real gross returns. The data are nominal monthly net returns, expressed in percent. Denote t the quarter and $\tilde{r}_{t+1,j}, j = 1, 2, 3$ the monthly returns in the given quarter $t + 1$, collected in the Kenneth French database.

We build the gross nominal quarterly return as

$$R^N_{t+1} = \prod_{j=1}^{3}\left(1 + \frac{\tilde{r}_{t+1,j}}{100}\right)$$

and then deflate by

$$R_{t+1} = \frac{R^N_{t+1}}{1 + \pi_{t+1}}.$$

In [ ]:
```
# download the data in ZIP format and extract the CSV file
param = {}
param['filename'] = 'https://mba.tuck.dartmouth.edu/pages/faculty/ken.french
param['returns'] = ['Mkt-RF','SMB','HML','RF']

result = requests.get(param['filename'])
data_FF = pd.read_csv(BytesIO(result.content),compression='zip',header=2)
```

The file contains both monthly and annual data, so we first need to extract the right rows, corresponding to monthly data (this may be easier to do manually).

In [ ]:
```
# process data
# rename date column
data_FF = data_FF.rename(columns={'Unnamed: 0':'date'})
# drop non-numerical rows
data_FF = data_FF[pd.to_numeric(data_FF['date'], errors='coerce').notnull()]
# only select data corresponding to months, and create a year-month index YY
data_FF['YYYYMM'] = pd.to_numeric(data_FF['date'])
data_FF = data_FF[data_FF['YYYYMM'] > 192606]
# convert returns columns into numerical values
for f in param['returns']:
    data_FF[f] = pd.to_numeric(data_FF[f])
# create years and months identifiers
data_FF['YYYY'] = data_FF['YYYYMM'] // 100
data_FF['MM'] = data_FF['YYYYMM'] % 100
# create the market return
data_FF['Mkt'] = data_FF['Mkt-RF'].to_numpy() + data_FF['RF']
```

The monthly return data must be converted to quarterly data by multiplicative accumulation of monthly returns in corresponding quarters.

In [ ]:
```
# identify odd initial and terminal months not corresponding to whole quarte
init_year_months = sum(data_FF['YYYY'] == min(data_FF['YYYY']))
end_year_months = sum(data_FF['YYYY'] == max(data_FF['YYYY']))
# isolate months corresponding to whole quarters
if end_year_months % 3 > 0:
    d = data_FF.iloc[init_year_months % 3:-(end_year_months % 3),:].copy()
else:
    d = data_FF.iloc[init_year_months % 3:,:].copy()
# create a quarter index for each month
d['YYYYQ'] = d['YYYY']*10 + (d['MM']-1) // 3 + 1
# aggregate monthly nominal returns into quarterly nominal returns, using th
# returns are stored as net returns
d = d[['YYYYQ','RF','Mkt']].groupby('YYYYQ').apply(lambda grp: ((1+grp/100).
d = d.drop(columns=['YYYYQ'])
```

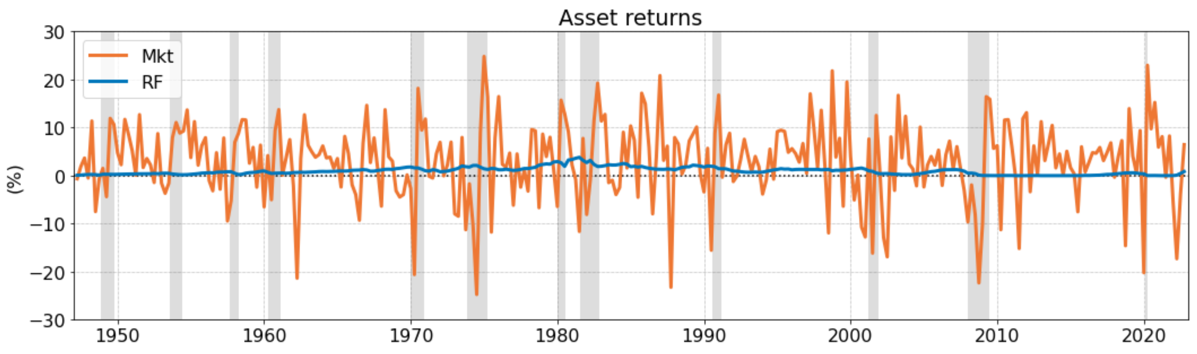
Finally, we can merge both datasets.

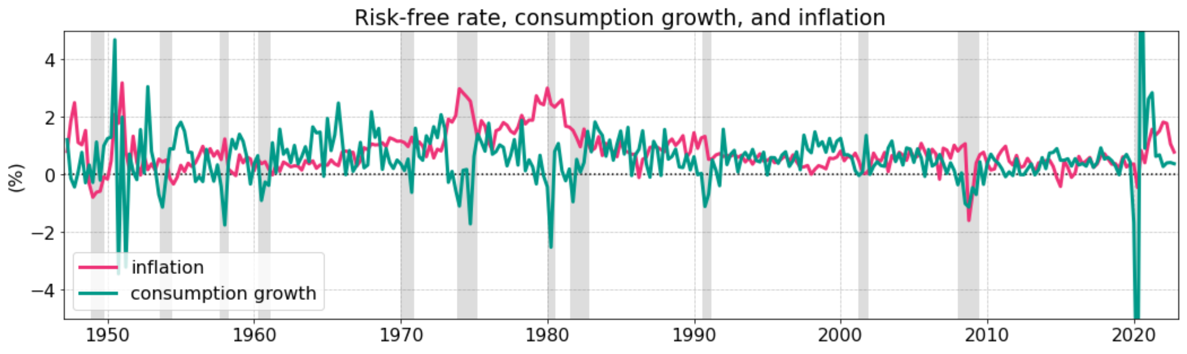
In [ ]:
```
# merge the dataset from FRED with the returns data, only using overlapping
data = pd.concat([d,data_C],axis=1,join="inner")
data['date'] = data.index//10 + ((data.index % 10)-1)/4

# create real returns
data['Mkt_real'] = (1+data['Mkt'])/(1+data['infl'])-1
data['RF_real'] = (1+data['RF'])/(1+data['infl'])-1
data['Mkt-RF_real'] = data['Mkt_real'] - data['RF_real']
```

Processed data

It is instructive to plot all the constructed quantities.

In [ ]:
```
fig_param = {'figsize' : [15,9], 'fontsize': 16, 'subplots': [2,1],
         'title': 'Asset returns',#'Central Bank Policy Rates',
         'xlim': [1947,2023], 'ylim': [-30,30],
         'xlabel': '', 'ylabel': '(%)',
         'ylogscale': False,
         'showgrid': True, 'highlightzero': True,
         'showNBERrecessions' : True, 'showNBERrecessions_y': [-30,30]}

fig,ax = myGenerateTSPlot(fig_param)

ax[0,0].plot(data['date'],data['Mkt'].to_numpy()*100,
             linewidth=3,color=myColor['tolVibrantOrange'],label='Mkt')
ax[0,0].plot(data['date'],(data['RF'].to_numpy())*100,
             linewidth=3,color=myColor['tolVibrantBlue'],label='RF')
#ax[0,0].plot(data['date'],data['SMB'].to_numpy()*100,
#             linewidth=3,color=myColor['tolVibrantGrey'],label='SMB')
#ax[0,0].plot(data['date'],data['HML'].to_numpy()*100,
#             linewidth=3,color=myColor['tolMutedRose'],label='HML')
x = ax[0,0].legend(loc="upper left")

ax[1,0].plot(data['date'],data['infl'].to_numpy()*100,
             linewidth=3,color=myColor['tolVibrantMagenta'],label='inflation'
ax[1,0].plot(data['date'],data['gC'].to_numpy()*100,
             linewidth=3,color=myColor['tolVibrantTeal'],label='consumption g
ax[1,0].set_ylim([-5,5])
ax[1,0].set_title('Risk-free rate, consumption growth, and inflation')

x = ax[1,0].legend(loc="lower left")
fig.tight_layout()
```





All quantities in the above graph are quarterly, not annualized. The first thing you should notice are the dramatically different volatilities of asset returns relative to macroeconomic quantities and the risk-free rate. This is known as the **excess volatility** puzzle (Shiller (1981)) - the stock market seems to be fluctuating way too much, both at high frequencies and over the business cycle, relative to the magnitude of the changes in macroeconomic conditions.

Generalized method of moments

We now construct the GMM estimate of the parameters β and γ.

We will use the fmin function from the optimize package. The next function computes the GMM objective function.

In [ ]:
```
# par = [bet, gam] is the vector of parameter values which are being estimat
def GMM_objective(par):
    bet = par[0]
    gam = par[1]
    sdf = bet*(1+data['gC'])**(-gam)
    # moment conditions
    f1 = sdf * (data['Mkt-RF_real'])
    f2 = sdf * (1 + data['RF_real']) - 1
    T = len(f1)
    # weighting matrix
    W = np.identity(2)
    # f is the T*M matrix of data for the moment conditions
    f = np.array([f1,f2]).transpose()
    RP = np.cov(np.array([sdf,data['Mkt-RF_real']]))
    g = sum(f).transpose()/T
    return g.transpose()@W@g
```

Choose an initial guess of parameters and then run the minimization function.

In [ ]:
```
# initial parameter guess for [bet,gam]
par = [0.96, 100]

# restrict data for estimation if desired
# data_backup = data.copy()
# data = data.drop(data.index[data.index > 20200])

xopt = optimize.fmin(GMM_objective,par,xtol=1e-12)
```

```
Optimization terminated successfully.
         Current function value: 0.000408
         Iterations: 112
         Function evaluations: 242
```

Evaluate output. In this example, we are estimating two parameters using two moment conditions, so distribution theory for the estimate is degenerate but we can still check the point estimate and fit of the model at this point estimate.

In [ ]:
```
# collect estimated parameters
bet = xopt[0]
gam = xopt[1]

sdf = bet*(1+data['gC'])**(-gam)
f1 = sdf * (data['Mkt-RF_real'])
f2 = sdf * (1 + data['RF_real']) - 1
T = len(f1)
f = np.array([f1,f2]).transpose()
g = sum(f)/T

W = np.identity(2)
# create column array with moment condition data
RP = np.cov(np.array([sdf,data['Mkt-RF_real']]))[0,1] * (1+data['RF'].mean()
Mkt_RF_av = data['Mkt-RF_real'].mean()

print(f'Using data for period {data.index[0]}-{data.index[-1]}.')
print(f'Estimated parameters: gamma = {gam:.3f}, beta = {bet:.3f}')
print(f'GMM objective value from first stage: {GMM_objective(xopt):.6f}')
print(f'Average moment condition errors (g, in %): {(g[0]*100):.5f}, {(g[1]*
print(f'Risk premium computed from covariance with SDF at the estimated valu
print(f'Average market excess return in the data (%, quarterly frequency): {
```

```
Using data for period 19472-20224.
Estimated parameters: gamma = 3.837, beta = 1.017
GMM objective value from first stage: 0.000408
Average moment condition errors (g, in %): 2.01888, -0.04078
Risk premium computed from covariance with SDF at the estimated values (%):
0.00894
Average market excess return in the data (%, quarterly frequency): 2.03227
```

In principle, the GMM objective value should be equal to zero, if it is not, it means that either there is not a combination of parameter values for which the moment conditions are satisfied simultaneously, or the optimizer has not found it.

The average moment condition errors indicate by how much are the moment conditions missed at the preferred parameter values - a high (absolute) value at the equity premium moment condition indicates that the model struggles to explain the equity premium.

This is verified by comparing the average market excess return in the data with the risk premium implied by the moment restriction evaluated at the estimated parameter value. The theory implies that this risk premium must be equal to the negative of the covariance between the SDF and the realized excess return (slightly adjusted by the risk-free rate). A large difference between the empirical and theoretically predicted value again reflects model misspecification.

Extensions

Two different estimation methods:

- fix beta, estimate gamma, with two moments, evaluate the J test
- add an instrument

In [ ]:
```
# code for the evaluation of the J-test
M = len(g)   # number of moment conditions
k = len(par) # number of estimated parameters

# covariance matrix - computed in two different ways, the latter under the a
V = np.cov(f.transpose())
Valt = 1/T * f.transpose()@f
GMM_obj = T * g@np.linalg.inv(V)@g.transpose()
p_val = 1 - stats.chi2.cdf(GMM_obj,M-k)

print(f'{M} moment conditions, {k} estimated parameters, {M-k} degrees of fr
print(f'GMM test statistic: {GMM_obj:.3f}, p-value: {p_val:.5f}')
```

```
2 moment conditions, 2 estimated parameters, 0 degrees of freedom.
GMM test statistic: 17.858, p-value: nan
```