tutorial_2

October 4, 2019

1 Tutorial 2 - Relaxing OLS assumptions

2 Homeskadasticity

2.1 Simulate data

We have a sample $(x_t, y_t)_{t=1}^T$ generated from the following model

$$y_t = \alpha + \beta x_t + e_t$$

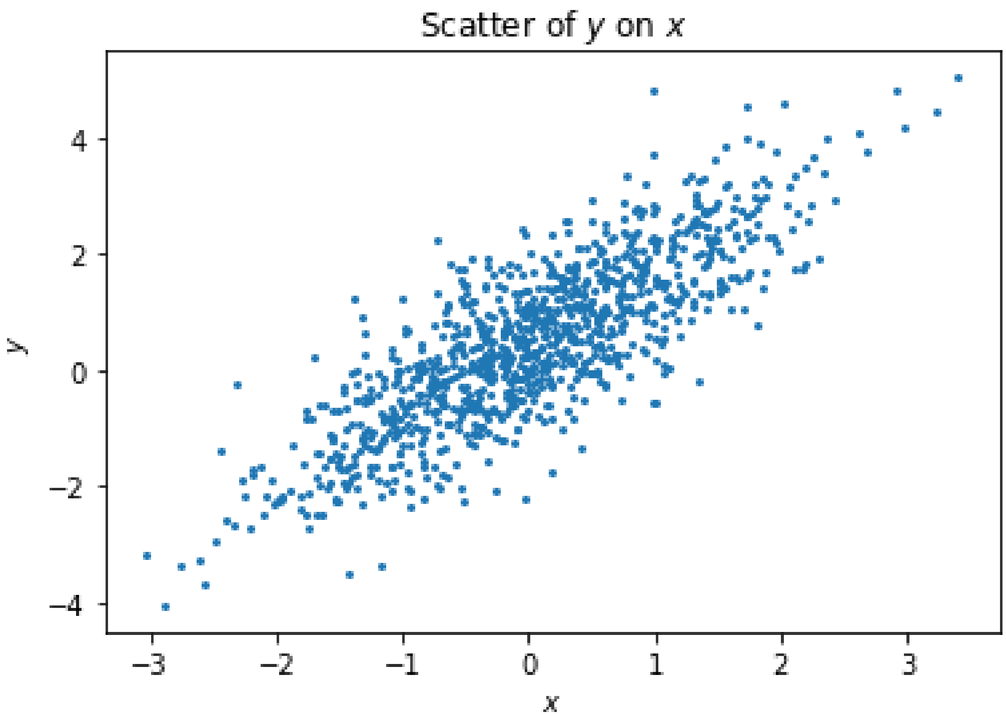
where $x \perp e$ and x, e follow, respectively, a $N(0,1)$ and $N(0,\sigma^2)$ distribution.

[1]:
```
import numpy as np                        # linear algebra
import statsmodels.formula.api as smf    # linear regression
import matplotlib.pyplot as plt          # plotting
import pandas as pd                      # dataframes
from scipy import stats                  # N(0, 1) pdf

# simulation parameters
npop  = 1000   # must be even
alpha = .5
beta  = 1.2
sigma = .8

# generate random data
# np.random.seed(1)   # important! Seed for reproducibility
x1 = np.random.normal(0, 1, npop)
e1 = np.random.normal(0, sigma, npop)
y1 = alpha + beta*x1 + e1

# create dataframe with population data
df1 = pd.DataFrame()
df1['y'] = y1
df1['x'] = x1
```

[2]:
```
# plot scatter
fig, ax = plt.subplots()
ax.set_title("Scatter of $y$ on $x$")
ax.scatter(df1['x'], df1['y'], s=4)
```

```
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

[2]: Text(0, 0.5, 'y')



2.1.1 True vs empirical distribution OLS estimator

Here is the experiment that we are going to run in a somewhat heuristic way:

1. we fit a linear regression `y ~ x` on the *whole* sample and save the estimated $\hat{\beta}$ and its SE
2. we sample a random subset of our data $(x_t^{(i)}, y_t^{(i)})_t$
3. we fit the same linear regression `y ~ x` on this sub-sample and save the estimated $\hat{\beta}^{(i)}$
4. we repeat steps 2-3) for $i = 1, 2, \ldots, Nsim$ and obtain an empirical distribution of all the $(\hat{\beta}^{(i)})_{i=1}^{N}$

Then, if we correctly specified our model, we should expect that the empirical distribution of the $\hat{\beta}^{(i)}$ approaches the “true” asymptotic distribution of $\hat{\beta}$ which - under homoskedasticity - we know is

$$\sqrt{T}\begin{pmatrix}\hat{\alpha}-\alpha\\ \hat{\beta}-\beta\end{pmatrix} \underset{T\to+\infty}{\to} \mathcal{N}\left(\begin{pmatrix}0\\0\end{pmatrix}, \sigma^2 plim\left(\frac{X'X}{T}\right)^{-1}\right)$$

[3]:
```python
# 1) fit beta hat from full dataset
reg_mod1 = smf.ols("y ~ x", df1)
fit_hom1 = reg_mod1.fit()  # homosk errors
betahat_se_hom1 = fit_hom1.bse[1]
fit_hom1.summary()
```

[3]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.679
Model:                            OLS   Adj. R-squared:                  0.678
Method:                 Least Squares   F-statistic:                     2109.
Date:                Fri, 04 Oct 2019   Prob (F-statistic):          2.45e-248
Time:                        10:00:41   Log-Likelihood:                -1211.3
No. Observations:                1000   AIC:                             2427.
Df Residuals:                     998   BIC:                             2436.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.4482      0.026     17.413      0.000       0.398       0.499
x              1.1730      0.026     45.922      0.000       1.123       1.223
==============================================================================
Omnibus:                        3.957   Durbin-Watson:                   2.028
Prob(Omnibus):                  0.138   Jarque-Bera (JB):                4.180
Skew:                           0.083   Prob(JB):                        0.124
Kurtosis:                       3.270   Cond. No.                         1.04
==============================================================================

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

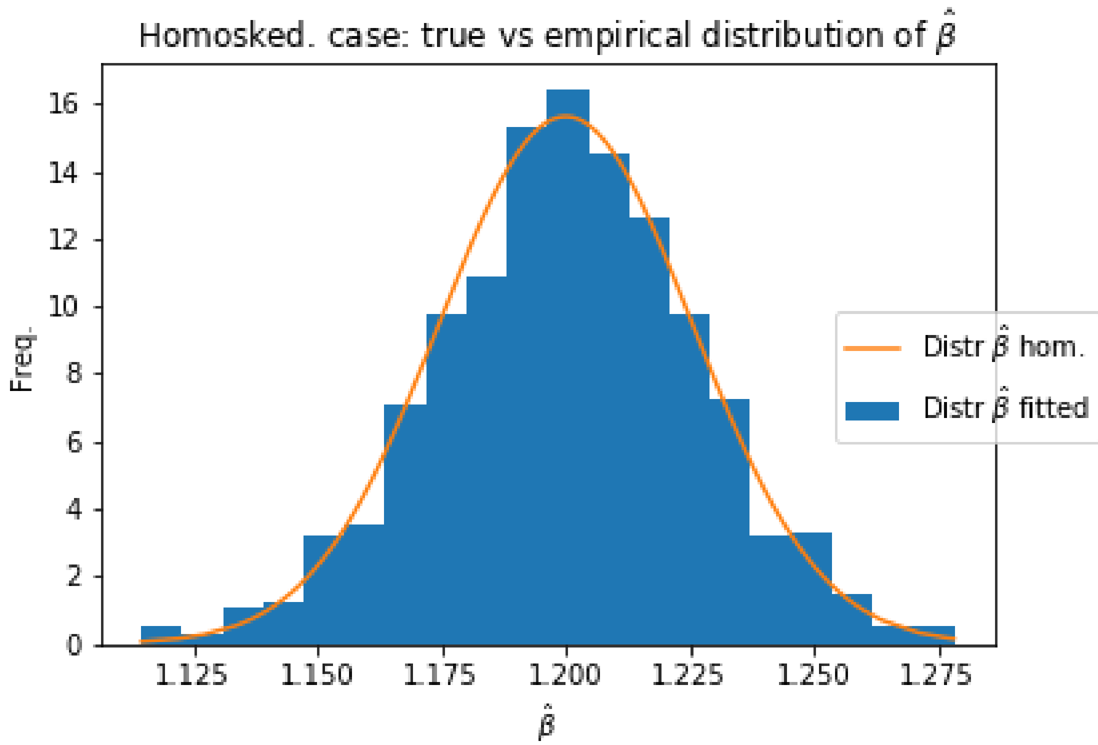
[4]:
```python
# 2-3) fit model on random sub-sample
def bootstrap_lm_hom(x, params):
    """ Fit univariate lin reg to random sample

        The function assumes we know the true DGP

        Arguments
        ---------
        x : np.array, univariate explanatory variable
        params : 1D np.array, alpha, beta and sigma

        Returns
```

```
        -------
        beta_boot : estimated beta coeff on bootstrap sample
    """
    # generate bootstrap sample
    alpha, beta, sigma = params
    n = len(x)
    indices = np.random.choice(np.arange(n), n)

    # generate residuals using right DGP
    x_boot = x[indices]
    e_boot = np.random.normal(0, sigma, n)
    y_boot = alpha + beta*x_boot + e_boot

    # create dataframe with boostrapped data
    df = pd.DataFrame()
    df['y'] = y_boot
    df['x'] = x_boot

    # fit linear regression
    fit = smf.ols("y ~ x", df).fit()
    beta_boot = fit.params[1]

    return beta_boot
```

[5]:
```
# fit on random sample
nsim = 1000
f1 = lambda i: bootstrap_lm_hom(df1['x'], [alpha, beta, sigma])
betas1 = list(map(f1, range(nsim)))
```

[6]:
```
# 4) plot empirical vs true density of betahat
fig, ax = plt.subplots()
ax.hist(betas1, bins=20, density=True, label="Distr $\hat{\\beta}$ fitted" )
ax.plot(sorted(betas1), stats.norm.pdf(sorted(betas1), beta, betahat_se_hom1),
        label="Distr $\hat{\\beta}$ hom.")
ax.set_xlabel("$\hat{\\beta}$")
ax.set_ylabel("Freq.")
ax.set_title("Homosked. case: true vs empirical distribution of $\hat{\\beta}$")
fig.legend(loc="center right")
```

[6]:
```
<matplotlib.legend.Legend at 0x2f3632e4f28>
```



2.2 Heteroskedasticity

2.2.1 Simulate data

We have a sample $(x_t, y_t)_{t=1}^{T}$ generated from the following model

$$y_t = \alpha + \beta x_t + e_t$$

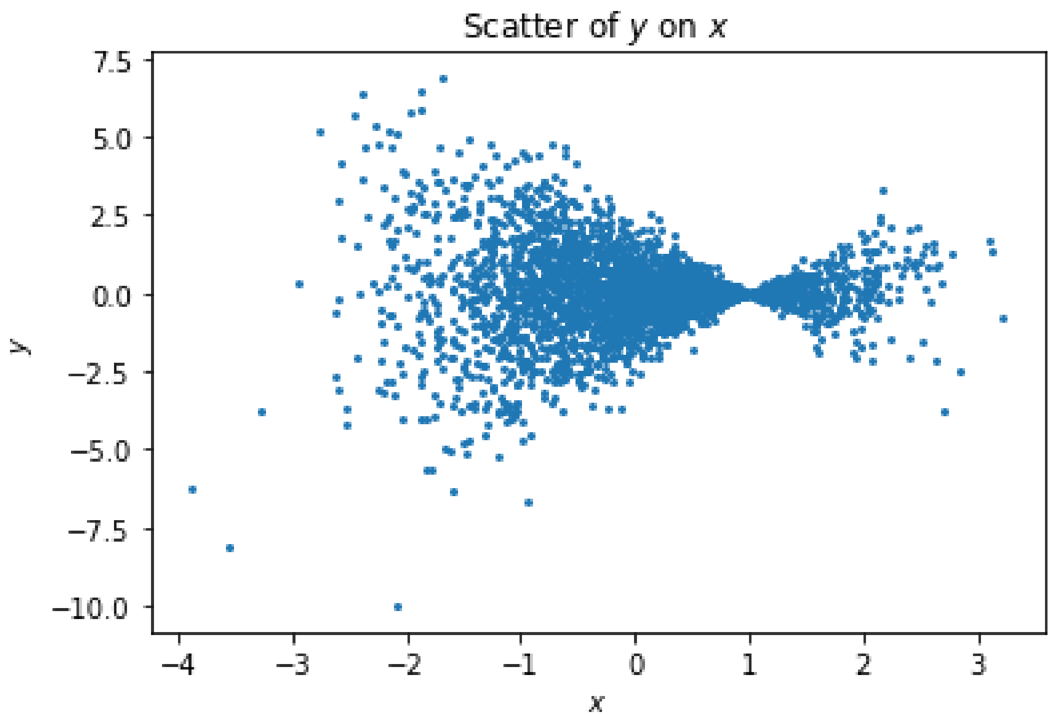
where $x \perp e$, x follows a $N(0,1)$ and $e_t \sim N(0, |1 - x_t|)$ for $t = 1, 2, \ldots, T$.

[7]:
```
# simulation parameters
npop = 3000
alpha = .5
beta = 1.2

# generate random data
np.random.seed(1234)   # important! Seed for reproducibility
x2 = np.random.normal(0, 1, npop)
e2 = np.random.normal(0, np.abs(1 - x2), npop)
y2 = alpha + beta*x2 + e2

# create dataframe with population data
df2 = pd.DataFrame()
df2['y'] = y2
```

```
df2['x'] = x2
```

[8]:
```
fig, ax = plt.subplots()
ax.set_title("Scatter of $y$ on $x$")
ax.scatter(df2['x'], e2, s=4)
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

[8]:
```
Text(0, 0.5, '$y$')
```



2.2.2 True vs empirical distribution OLS estimator

Here is the experiment that we are going to run in a somewhat heuristic way:

1. we fit a linear regression `y ~ x` on the whole sample and save the estimated $\hat{\beta}$ and its SE. We also (wrongly) assume *homoskedadastic* errors.
2. we fit a linear regression `y ~ x` on the whole sample and save the estimated $\hat{\beta}$ and its SE. We also (correctly) assume *heteroskedastic* errors.
3. we sample a random subset of our data $(x_t^{(i)}, y_t^{(i)})_t$
4. we fit the same linear regression `y ~ x` on this sub-sample and save the estimated $\hat{\beta}^{(i)}$
5. we repeat steps 3-4) for $i = 1, 2, \ldots, Nsim$ and obtain an empirical distribution of all the $(\hat{\beta}^{(i)})_{i=1}^{N}$

Then, if we correctly specified our model, we should expect that the empirical distribution of the $\hat{\beta}^{(i)}$ approaches the "true" asymptotic distribution of $\hat{\beta}$ which - under heteroskedasticy - we know is

$$\sqrt{T}\begin{pmatrix}\hat{\alpha}-\alpha\\ \hat{\beta}-\beta\end{pmatrix} \to \mathcal{N}\left(\begin{pmatrix}0\\0\end{pmatrix}, (X'X/T)^{-1}\frac{X'\Omega X}{T}(X'X/T)^{-1}\right)$$

with

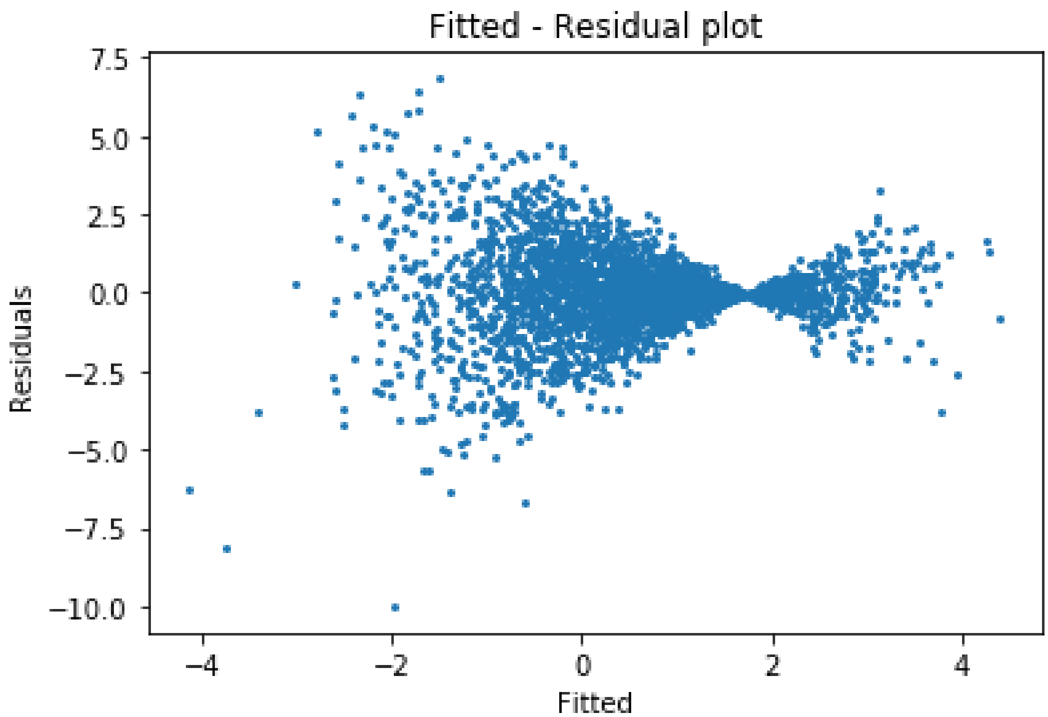
$$\Omega = \begin{pmatrix}\sigma_1^2 & 0 & \dots & 0\\ 0 & \sigma_2^2 & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & \sigma_T^2\end{pmatrix}.$$

Note: we have suppressed the plim operator for the sake of simplicity.

[9]:
```python
# 1) fit beta hat from full dataset under homeskd
reg_mod2 = smf.ols("y ~ x", df2)
fit_hom2 = reg_mod2.fit()                    # homosk errors
betahat_se_hom2 = fit_hom2.bse[1]



# plot fit vs residuals
fig, ax = plt.subplots()
ax.set_title("Fitted - Residual plot")
ax.scatter(fit_hom2.fittedvalues, fit_hom2.resid, s=4)
ax.set_xlabel('Fitted')
ax.set_ylabel('Residuals')
# Show summary of the regression results
fit_hom2.summary()
```

[9]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.425
Model:                            OLS   Adj. R-squared:                  0.424
Method:                 Least Squares   F-statistic:                     2212.
Date:                Fri, 04 Oct 2019   Prob (F-statistic):               0.00
Time:                        10:00:56   Log-Likelihood:                -5218.4
No. Observations:                3000   AIC:                         1.044e+04
Df Residuals:                    2998   BIC:                         1.045e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.5279      0.025     20.964      0.000       0.478       0.577
x              1.2012      0.026     47.035      0.000       1.151       1.251
==============================================================================
```

```
Omnibus:                      283.977   Durbin-Watson:                   1.971
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1933.630
Skew:                          -0.126   Prob(JB):                         0.00
Kurtosis:                       6.925   Cond. No.                         1.04
==============================================================================

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



[10]:
```python
# 2-3) fit model on random sub-sample
def bootstrap_lm_het(x, params):
    """ Fit univariate lin reg to random sample

        The function assumes we know the true DGP

        Arguments
        ---------
        x : np.array, univariate explanatory variable
        params : 1D np.array, alpha, beta and sigma

        Returns
        -------
```

```
        beta_boot : estimated beta coeff on bootstrap sample
    """
    # generate bootstrap sample
    alpha, beta = params
    n = len(x)
    indices = np.random.choice(np.arange(n), n)

    # generate residuals using right DGP
    x_boot = x[indices]
    e_boot = np.random.normal(0, np.abs(1 - x_boot), n)
    y_boot = alpha + beta*x_boot + e_boot

    # create dataframe with boostrapped data
    df = pd.DataFrame()
    df['y'] = y_boot
    df['x'] = x_boot

    # fit linear regression
    fit = smf.ols("y ~ x", df).fit()
    beta_boot = fit.params[1]

    return beta_boot
```

[11]:
```
# fit on random sample
nsim = 1000
np.random.seed(987)
f2 = lambda i: bootstrap_lm_het(df2['x'], [alpha, beta])
betas2 = list(map(f2, range(nsim)))
```

[12]:
```
# 2) fit beta hat from full dataset under heterosk
fit_het2 = reg_mod2.fit(cov_type = 'HC0')  # White heter errors
betahat_se_het2 = fit_het2.bse[1]
print(f"beta SE hom {betahat_se_hom2}")
print(f"beta SE het {betahat_se_het2}")
fit_het2.summary()
```

```
beta SE hom 0.025538200266201246
beta SE het 0.036871988227998316
```

[12]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.425
Model:                            OLS   Adj. R-squared:                  0.424
Method:                 Least Squares   F-statistic:                     1061.
Date:                Fri, 04 Oct 2019   Prob (F-statistic):          1.44e-199
Time:                        10:01:09   Log-Likelihood:                -5218.4
```

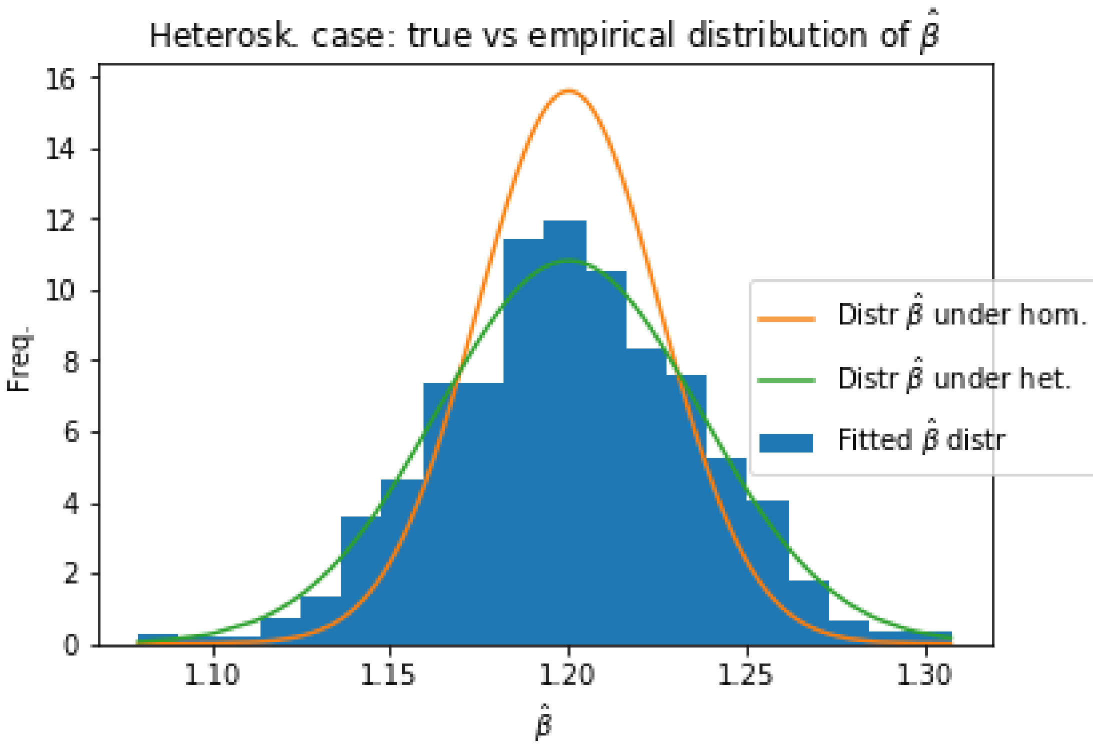
```
No. Observations:                3000   AIC:                         1.044e+04
Df Residuals:                    2998   BIC:                         1.045e+04
Df Model:                           1
Covariance Type:                  HC0
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.5279      0.026     20.260      0.000       0.477       0.579
x              1.2012      0.037     32.577      0.000       1.129       1.273
==============================================================================
Omnibus:                      283.977   Durbin-Watson:                   1.971
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1933.630
Skew:                          -0.126   Prob(JB):                         0.00
Kurtosis:                       6.925   Cond. No.                         1.04
==============================================================================

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
"""
```

[13]:

```
# 5) plot empirical vs true density of betahat
fig, ax = plt.subplots()
ax.hist(betas2, bins=20, density=True, label="Fitted $\hat{\\beta}$ distr" )
ax.plot(sorted(betas2), stats.norm.pdf(sorted(betas2), beta, betahat_se_hom2),
        label="Distr $\hat{\\beta}$ under hom.")
ax.plot(sorted(betas2), stats.norm.pdf(sorted(betas2), beta, betahat_se_het2),
        label="Distr $\hat{\\beta}$ under het.")
ax.set_xlabel("$\hat{\\beta}$")
ax.set_ylabel("Freq.")
ax.set_title("Heterosk. case: true vs empirical distribution of $\hat{\\beta}$")
fig.legend(loc="center right")
```

[13]:

```
<matplotlib.legend.Legend at 0x2f3634c8668>
```



2.3 Serial correlation

2.3.1 Simulate data

We have a sample $(x_t, y_t)_{t=1}^T$ generated from the following model

$$y_t = \alpha + \beta x_t + e_t$$

where $x \perp e$, x follows a $U(0,1)$ and $e_t \sim MA(1)$ process. A MA(1) process is simple a time series process of the form

$$e_t = u_t + \theta u_{t-1}$$

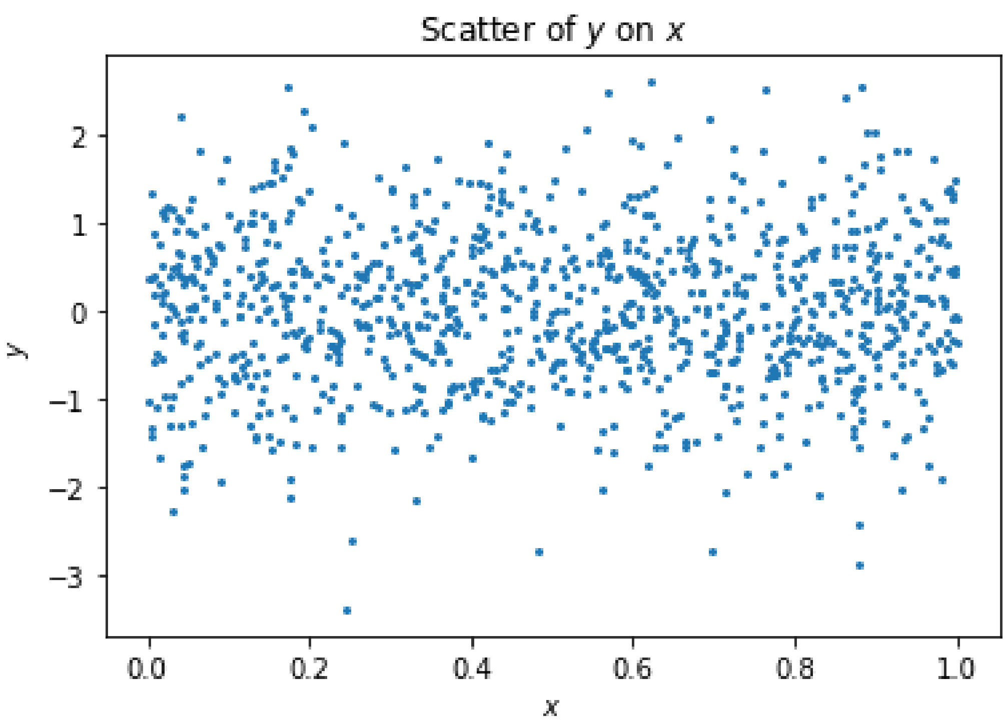
where u_t, u_{t-1} are white noise (i.e. orthogonal, zero-mean random variables).

[14]:
```
from statsmodels.tsa.arima_process import arma_generate_sample  # generate MA

# simulation parameters
npop = 1000
alpha = .5
beta = 1.2
sigma = .7  # MA(1) error sd
theta = .8  # MA(1) param
```

```
# generate random data
np.random.seed(1234)  # important! Seed for reproducibility
x3 = np.random.uniform(0, 1, npop)
e3 = arma_generate_sample([1], [1, theta], npop, sigma)
y3 = alpha + beta*x3 + e3

# create dataframe with population data
df3 = pd.DataFrame()
df3['y'] = y3
df3['x'] = x3
```

[15]:
```
fig, ax = plt.subplots()
ax.set_title("Scatter of $y$ on $x$")
ax.scatter(df3['x'], e3, s=4)
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
```

[15]: Text(0, 0.5, 'y')



[16]:
```
# 1) fit beta hat from full dataset under homeskd
reg_mod3 = smf.ols("y ~ x", df3)
fit_hom3 = reg_mod3.fit()                   # homosk errors
betahat_se_hom3 = fit_hom3.bse[1]
```

2.3.2 Check for serial correlation

[17]:
```
from scipy.stats import norm  # get quantile funct for N(mu, sigma)

# get resid from heterosk model
ehat = fit_hom3.resid  # it does not matter if the original fit was assuming␣
 ↪hom

# create ehat_t*ehat_{t-j} for j=0, ..., 4
def cross_prod_lag(x, k):
    x_std = (x - np.mean(x))/np.std(x)
    return x_std*x_std.shift(-k)
e2_lag0 = cross_prod_lag(ehat, 0)
e2_lag1 = cross_prod_lag(ehat, 1)
e2_lag2 = cross_prod_lag(ehat, 2)
e2_lag3 = cross_prod_lag(ehat, 3)
e2_lag4 = cross_prod_lag(ehat, 4)
e2_lag5 = cross_prod_lag(ehat, 5)


# put cross-products in dataset
df_e2_lags = pd.DataFrame()
df_e2_lags['e2_lag1'] = e2_lag1
df_e2_lags['e2_lag2'] = e2_lag2
df_e2_lags['e2_lag3'] = e2_lag3
df_e2_lags['e2_lag4'] = e2_lag4
df_e2_lags['e2_lag4'] = e2_lag4
df_e2_lags['e2_lag5'] = e2_lag5

# run regressions on a constant
eac_reg1 = smf.ols('e2_lag1 ~ 1', df_e2_lags).fit()
eac_reg2 = smf.ols('e2_lag2 ~ 1', df_e2_lags).fit()
eac_reg3 = smf.ols('e2_lag3 ~ 1', df_e2_lags).fit()
eac_reg4 = smf.ols('e2_lag4 ~ 1', df_e2_lags).fit()
eac_reg5 = smf.ols('e2_lag5 ~ 1', df_e2_lags).fit()

# get point est
corr_lag1 = eac_reg1.params[0]
corr_lag2 = eac_reg2.params[0]
corr_lag3 = eac_reg3.params[0]
corr_lag4 = eac_reg4.params[0]
corr_lag5 = eac_reg5.params[0]
corr_lags = (corr_lag1, corr_lag2,
             corr_lag3, corr_lag4, corr_lag5)

# get SE
se_corr_lag1 = eac_reg1.bse[0]
```

```python
se_corr_lag2 = eac_reg2.bse[0]
se_corr_lag3 = eac_reg3.bse[0]
se_corr_lag4 = eac_reg4.bse[0]
se_corr_lag5 = eac_reg5.bse[0]
se_corr_lags = (se_corr_lag1, se_corr_lag2,
                se_corr_lag3, se_corr_lag4, se_corr_lag5)

# put corr, SE and lags in dataframe
df_acf_res = pd.DataFrame()
df_acf_res['lag'] = np.arange(6)[1:]
df_acf_res['corr'] = corr_lags
df_acf_res['se'] = se_corr_lags
df_acf_res['ci'] = df_acf_res['se']*norm.ppf(.995)
df_acf_res
```

[17]:

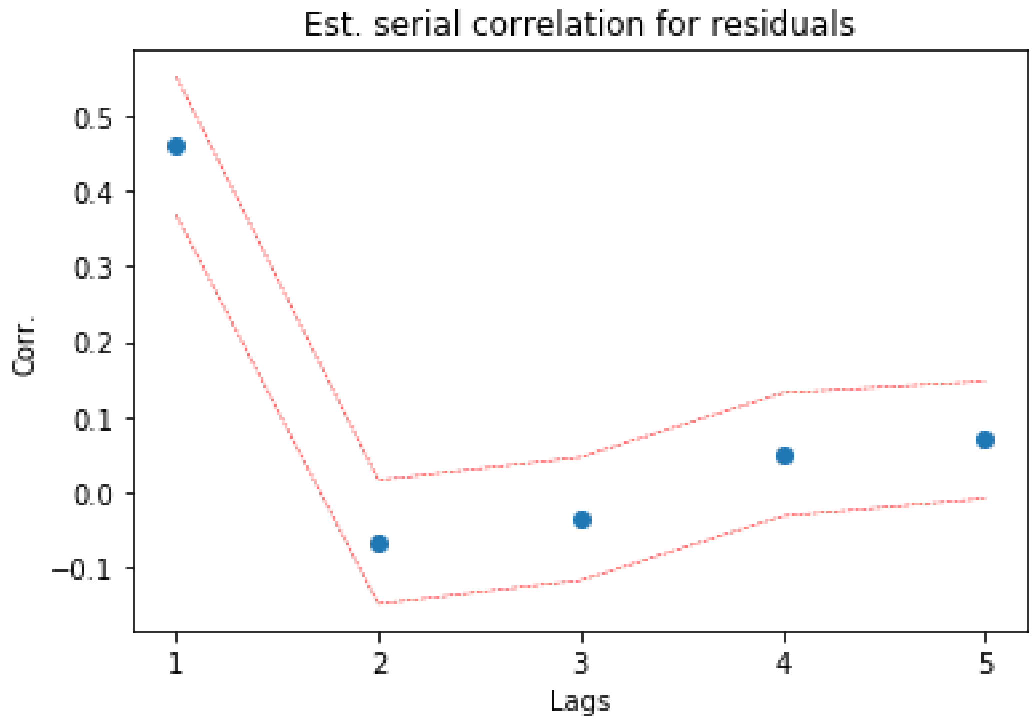
	lag	corr	se	ci
0	1	0.460620	0.035812	0.092246
1	2	-0.066885	0.031966	0.082338
2	3	-0.035929	0.031871	0.082093
3	4	0.049611	0.031822	0.081968
4	5	0.068970	0.030310	0.078074

[18]:

```python
fig, ax = plt.subplots()
ax.scatter(df_acf_res['lag'], df_acf_res['corr'])
ax.plot(df_acf_res['lag'], df_acf_res['corr'] + df_acf_res['ci'],
        linestyle = '--', color = 'red', linewidth = .5)
ax.plot(df_acf_res['lag'], df_acf_res['corr'] - df_acf_res['ci'],
        linestyle = '--', color = 'red', linewidth = .5)
ax.set_xticks(df_acf_res['lag'])
ax.set_xlabel('Lags')
ax.set_ylabel('Corr.')
ax.set_title('Est. serial correlation for residuals')
```

[18]:

```
Text(0.5, 1.0, 'Est. serial correlation for residuals')
```



2.3.3 True vs empirical distribution OLS estimator

Here is the experiment that we are going to run in a somewhat heuristic way:

1. we fit a linear regression `y ~ x` on the whole sample and save the estimated $\hat{\beta}$ and its SE. We also (wrongly) assume *homoskedadastic* errors.
2. we fit a linear regression `y ~ x` on the whole sample and save the estimated $\hat{\beta}$ and its SE. We also (correctly) assume *serially correlated* errors.
3. we sample a random subset of our data $(x_t^{(i)}, y_t^{(i)})_t$
4. we fit the same linear regression `y ~ x` on this sub-sample and save the estimated $\hat{\beta}^{(i)}$
5. we repeat steps 3-4) for $i = 1, 2, \ldots, Nsim$ and obtain an empirical distribution of all the $(\hat{\beta}^{(i)})_{i=1}^{N}$

Then, if we correctly specified our model, we should expect that the empirical distribution of the $\hat{\beta}^{(i)}$ approaches the "true" asymptotic distribution of $\hat{\beta}$ which - under $MA(1)$ errors - we know is

$$\sqrt{n}\begin{pmatrix}\hat{\alpha}-\alpha\\ \hat{\beta}-\beta\end{pmatrix} \to \mathcal{N}\left(\begin{pmatrix}0\\0\end{pmatrix}, (X'X)^{-1}X'\Omega X(X'X)^{-1}\right)$$

with

$$\Omega = \begin{pmatrix} (1+\theta^2)\sigma^2 & \theta\sigma^2 & 0 & \dots & 0 & 0 \\ \theta\sigma^2 & (1+\theta^2)\sigma^2 & \theta\sigma^2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \theta\sigma^2 & (1+\theta^2)\sigma^2 \end{pmatrix}$$

[19]:
```python
# 2) fit beta hat from full dataset under serial correlation
nw_cov_opt = {'maxlags':1, 'use_correction':True}
fit_nw3 = reg_mod3.fit(cov_type = 'HAC', cov_kwds=nw_cov_opt) # NW SE
betahat_se_nw3 = fit_nw3.bse[1]
print(f"beta SE hom {betahat_se_hom3}")
print(f"beta SE NW {betahat_se_nw3}")
fit_nw3.summary()
```

```
beta SE hom 0.09470910054205718
beta SE NW 0.09912844755947796
```

[19]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.143
Model:                            OLS   Adj. R-squared:                  0.142
Method:                 Least Squares   F-statistic:                     151.6
Date:                Fri, 04 Oct 2019   Prob (F-statistic):           1.53e-32
Time:                        10:01:10   Log-Likelihood:                -1293.0
No. Observations:                1000   AIC:                             2590.
Df Residuals:                     998   BIC:                             2600.
Df Model:                           1
Covariance Type:                  HAC
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.5077      0.061      8.301      0.000       0.388       0.628
x              1.2207      0.099     12.315      0.000       1.026       1.415
==============================================================================
Omnibus:                        2.827   Durbin-Watson:                   1.079
Prob(Omnibus):                  0.243   Jarque-Bera (JB):                2.948
Skew:                          -0.049   Prob(JB):                        0.229
Kurtosis:                       3.247   Cond. No.                         4.34
==============================================================================

Warnings:
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC)
using 1 lags and with small sample correction
"""
```

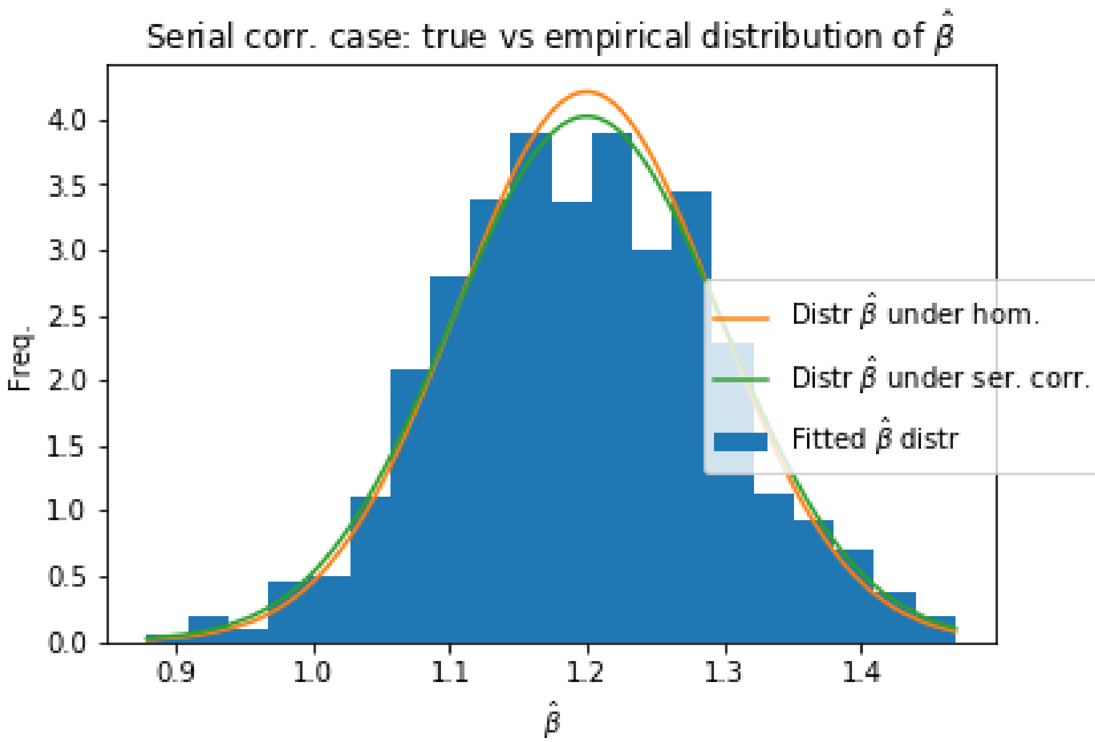
[20]:
```python
# 2-3) fit model on random sub-sample
def bootstrap_lm_nw(x, params):
    """ Fit univariate lin reg to random sample

        The function assumes we know the true DGP

        Arguments
        ---------
        x : np.array, univariate explanatory variable
        params : 1D np.array, alpha, beta, theta and sigma

        Returns
        -------
        beta_boot : estimated beta coeff on bootstrap sample
    """
    # generate bootstrap sample
    alpha, beta, theta, sigma = params
    n = len(x)
    indices = np.random.choice(np.arange(n), n)

    # generate residuals using right DGP
    x_boot = x[indices]
    e_boot = arma_generate_sample([1], [1, theta], n, sigma)
    y_boot = alpha + beta*x_boot + e_boot

    # create dataframe with boostrapped data
    df = pd.DataFrame()
    df['y'] = y_boot
    df['x'] = x_boot

    # fit linear regression
    fit = smf.ols("y ~ x", df).fit()
    beta_boot = fit.params[1]

    return beta_boot
```

[21]:
```python
# fit on random sample
nsim = 1200
np.random.seed(3)
f3 = lambda i: bootstrap_lm_nw(df3['x'], [alpha, beta, theta, sigma])
betas3 = list(map(f3, range(nsim)))
```

[22]:
```python
# 5) plot empirical vs true density of betahat
fig, ax = plt.subplots()
ax.hist(betas3, bins=20, density=True, label="Fitted $\hat{\\beta}$ distr" )
ax.plot(sorted(betas3), stats.norm.pdf(sorted(betas3), beta, betahat_se_hom3),
        label="Distr $\hat{\\beta}$ under hom.")
ax.plot(sorted(betas3), stats.norm.pdf(sorted(betas3), beta, betahat_se_nw3),
```

```
        label="Distr $\hat{\\beta}$ under ser. corr.")
ax.set_xlabel("$\hat{\\beta}$")
ax.set_ylabel("Freq.")
ax.set_title("Serial corr. case: true vs empirical distribution of␣
↪$\hat{\\beta}$")
fig.legend(loc="center right")
```

[22]: `<matplotlib.legend.Legend at 0x2f363887668>`



[ ]: