Financial Econometrics II : HW2*

Maxime Borel

February 11, 2023

Q1 : Euler equations a la Tauchen (1986)

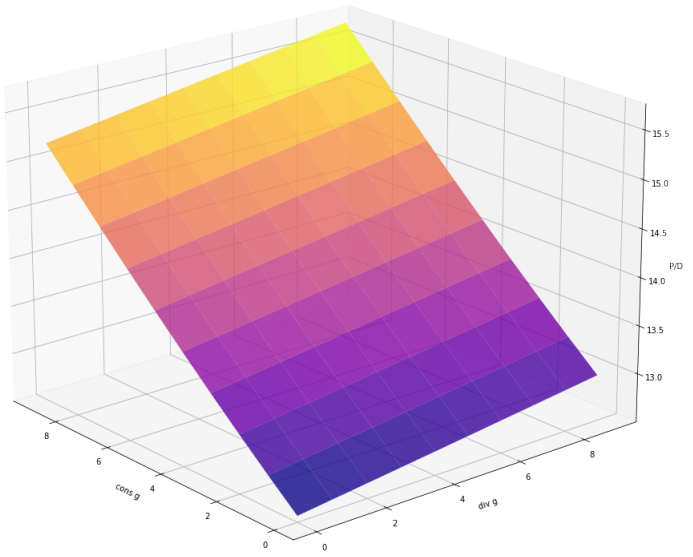

Figure 1: Price/dividend ratio surface

In this exercise, we are solving the Stochastic Growth Model by Tauchen (1990). We have the usual Euler equation of the price/dividend ratio which depends on the consumption growth and the dividend growth processes. They are modelled using a Vector Auto Regressive process (VAR) with parameter b for the intercept and A for past observation. The model description are as in the Lecture note 1 at page 10 and following.

In the code, we start by setting the parameters of each process, namely b, A and Ω, β, γ. We also set parameters to solve the Bellman equation such as the maximum number

*The code is available on github here

of iterations at 1000 and the tolerance at 0.0001. We transform the VAR process such that the errors are uncorrelated which allows simulating the resulting decorrelated errors. Then, we discretized the two components of the VAR model in 10 different states, each yielding 100 possible combinations. To this aim, one computes 10 middle points and the corresponding boundaries. Once, we get the simulation, we retrieve the dividend growth and consumption growth processes using $y_t = C^{-1}z_t + (I - A)^{-1}b$ with C being the lower triangular matrix of the cholesky decomposition of Ω. Furthermore, one computes the transition probability matrix Π. Eventually, one obtained the fixed point using the value function iteration methods. This gives the price to dividend ratio in every state.

Figure 1 shows the solution to the bellman equation. It represents the optimal price to dividend ratio in every possible state of the world which depends on the realisation of the dividend growth and the consumption growth. It is an increasing function in both consumption and dividend growth. It is increasing at a higher rate with consumption due to the risk aversion.

We can then simulate the model and plot the dividend growth, the consumption growth, the rate of return of the risky asset and the price to dividend ratio. This is shown in Figure 2. Based on these simulations, one can check if the processes behave as expected by running some regressions.

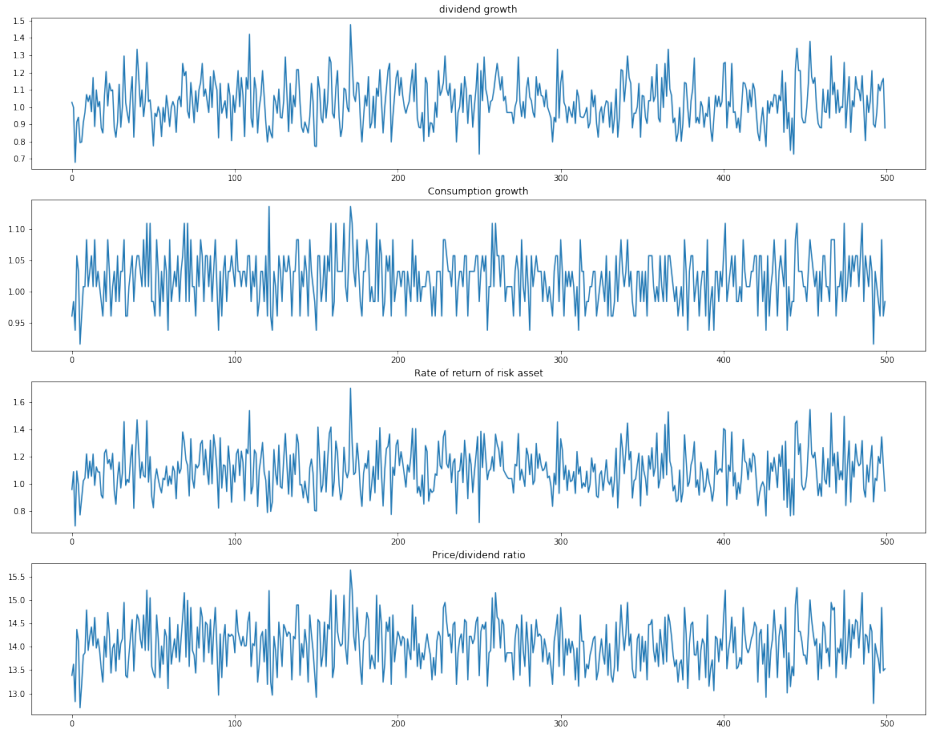

Figure 2: Simulation of the dividend growth, the consumption growth, the rate of return of the risky assets and the price to dividend ratio. The length of the simulation is T=500.

The VAR process was set up using this value

$$A = \begin{pmatrix} 0.073 & 0.620 \\ 0.015 & -0.122 \end{pmatrix}, \quad b = \begin{pmatrix} 0.003 \\ 0.022 \end{pmatrix}$$

Thus, we can run the following regressions for the dividend growth and the consumption process.

$$d_{t+1} = b_1 + a_{1,1} d_t + a_{1,1} c_t + \varepsilon^d_{t+1} \tag{1}$$

$$c_{t+1} = b_2 + a_{2,2} c_t + a_{2,1} d_t + \varepsilon^c_{t+1} \tag{2}$$

Table 1 and Table 2 present the results of equations (1) and (2), respectively. One can see that the point estimates are very close to the one set in the original matrices. All coefficient estimates have the right sign and the true parameter lies in the 5% confidence interval.

Table 1: This table shows the results of the regression of equation (1). Namely, it is the regression relative to the dividend growth process.

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
b_1	0.0047	0.006	0.781	0.435	-0.007	0.017
$a_{1,1}$	0.1103	0.050	2.220	0.027	0.013	0.208
$a_{1,2}$	0.5478	0.154	3.557	0.000	0.245	0.850

Table 2: This table shows the results of the regression of equation (2). Namely, it is the regression relative to the consumption growth process.

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
b_2	0.0223	0.002	11.194	0.000	0.018	0.026
$a_{2,1}$	0.0418	0.016	2.561	0.011	0.010	0.074
$a_{2,2}$	-0.1665	0.051	-3.295	0.001	-0.266	-0.067

Finally, one can regress the risky asset returns on the price-dividend as follow shown in equation (3) to check how the price-dividend affects the returns. The results are shown in Table 3. Overall, it seems that the returns are not related to the price-dividend ratio as the coefficient β is not significant. Moreover, the model generates a skewness of 0.31 and a kurtosis of 3.05 for the risky asset returns. This is far from the reality as risky asset return are negatively skewed and with fat tails. Thus, the model is not capturing the main properties of financial asset returns.

$$R_{t+1} = \alpha + \beta f_t + \varepsilon^R_{t+1} \tag{3}$$

Table 3: This table shows the results of the regression of equation (3). Namely, it is the regression relative to the risky asset returnson price-dividend ratio.

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
α	1.2818	0.198	6.471	0.000	0.893	1.671
β	-0.0128	0.014	-0.909	0.364	-0.040	0.015

Q2 : Gaussian Mixture Model

In the report, I show the most important part of the code. However, the file GaussianMixture.py contains all the additional details. The report only shows a sample of the file.

Q2.1 : vech

```
def vech_loop( A ):
    N = A.shape[ 0 ]
    K = A.shape[ 1 ]
    if N != K:
        raise NameError( "The input matrix needs to be a squared matrix" )
    else :
        to_remove = 0
        to_store = np.array( [] )
        for i in range( K ):
            to_store = np.hstack( [ to_store, A[ to_remove:, i ] ] )
            to_remove += 1
        return to_store.reshape( [ -1, 1 ] )
```

Q2.2 : Multivariate normal density

I implement the special case of the bivariate normal distribution. This allows avoiding loops and thus save time. Obviously, this function only works for bivariate datasets. If one would like to have additional variable to cluster the data, one should generalise the function using the inverse of the matrix as well as the determinant.

```
def mv_normal_density( x, mu, sigma ):
    if mu.shape.__len__() == 1:
        mu = mu.reshape( [ -1, 1 ] )

    y = x[ :, 0 ]
    z = x[ :, 1 ]
    sigmaY = np.sqrt( sigma[ 0, 0 ] )
    sigmaZ = np.sqrt( sigma[ 1, 1 ] )
    rho = sigma[ 1, 0 ] / ( sigmaZ * sigmaY )

    first = ( y - mu[ 0 ] ) / sigmaY
    second = ( z - mu[ 1 ] ) / sigmaZ
    third = np.exp( -( 1/(2*( 1 - rho**2 ) ) ) *( first**2 -
    ↪  2*rho*first*second + second**2 ) )
    cst = 1 / ( 2 * np.pi * sigmaY * sigmaZ * np.sqrt( 1 - rho**2 ))

    density = ( cst * third ).reshape( [ -1, 1 ] )
    return density
```

Q2.3 : EM Algorithm

```
while ( dist > e_tol ) & ( position < max_iter ):
    f1 = mv_normal_density( X, mu1, sigma1 )
    f2 = mv_normal_density( X, mu2, sigma2 )
    density = smallPi1*f1 + smallPi2*f2

    # update proba
    proba_state1 = ( f1*smallPi1 ) / density
    proba_state2 = ( f2*smallPi2 ) / density

    # update mu
    mu1 = ( X*proba_state1 ).sum( 0 )  / proba_state1.sum()
    mu2 = ( X*proba_state2 ).sum( 0 )  / proba_state2.sum()

    # update sigma
    partial1 = ( X - mu1 ) * np.sqrt( proba_state1 )
    partial2 = ( X - mu2 ) * np.sqrt( proba_state2 )

    sigma1 = ( partial1.T @ partial1 ) / proba_state1.sum( 0 )
    sigma2 = ( partial2.T @ partial2 ) / proba_state2.sum( 0 )

    smallPi1 = proba_state1.mean()
    smallPi2 = 1 - smallPi1

    theta_new = np.vstack( (
        mu1.reshape( [ -1, 1 ] ), vech_loop( sigma1 ),
        mu2.reshape( [ -1, 1 ] ), vech_loop( sigma2 ),
        smallPi1
    ))

    dist = ( ( theta_old - theta_new )**2 ).sum()
    theta_old = theta_new
    position += 1

print( position )
```

Parameter Estimates		
π	0.49	
	Gaussian 1	
μ_2	4.99	6.82
Σ_2	0.99	
	0.63	0.73
	Gaussian 2	
μ_1	3.82	5.61
Σ_1	1.76	
	1.89	5.09

Table 4: This table presents the estimates of the parameters that characterized a mixture of bivariate distribution. The parameters are estimated using the EM algorithm.

Q2.4 : Classification

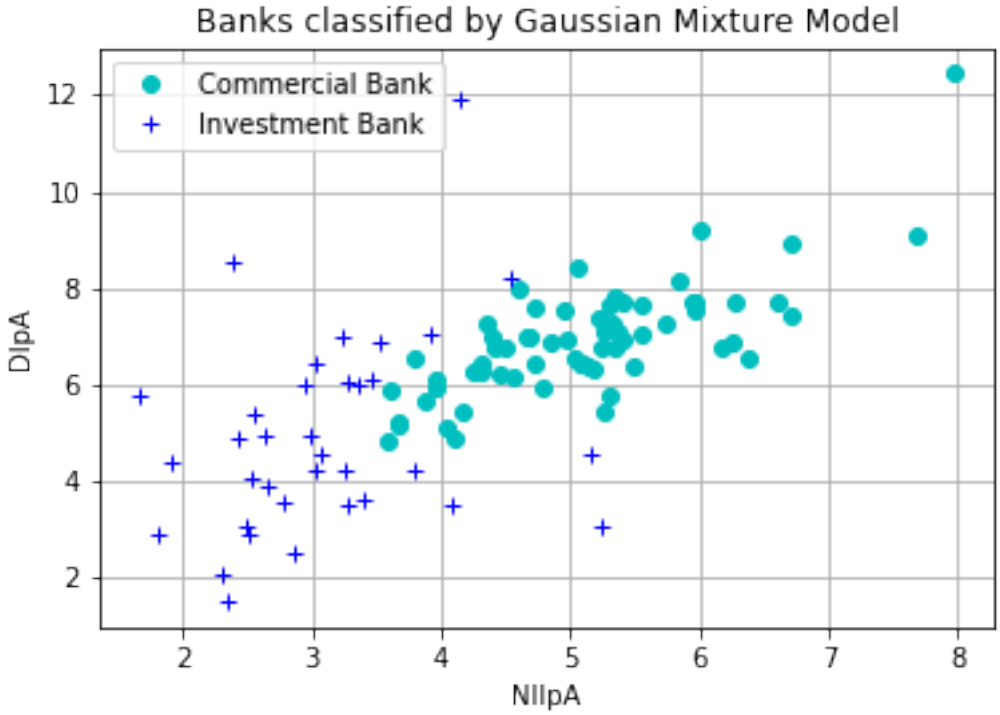

Figure 3: Bank Classification using Gaussian Mixture. The banks are classified in a group based on the value of the density. For an observation, if the density of the first bivariate Gaussian is superior or equal to the density of the second bivariate Gaussian, the bank is classified as Commercial Bank else as Investment Bank.

Q3 : Generalized Method of Moments

In start by computing the total return of the S&P500 index by adding the dividend time-series to the prices. Then I computed the gross rate of return. Regarding the risk-free rate, first, I divided by 100 and add 1. Then, I took the annual gross rate at the power 1/12 to get the gross monthly interest rate. I computed the inflation rate as the monthly percentage change of the CPI. I obtain the real term time-series by dividing all the nominal time-series by the gross monthly inflation rate.

As before, the code shown below is a sample of the file GMM.py that shows the main element of the code. However, there is a lot of additional detail in the file. Table 5 and 6 shows the coefficient for the discount factor and the risk aversion parameter using GMM for nominal and real terms, respectively.

```
zvar = df[ [ 'const', 'ct_lag1', 'r_lag1', 'rf_lag1' ] ]  # instrument
xvar = df[ [ 'ct', 'Rt', 'Rft' ] ] # exog variables

yvar = np.zeros( xvar.shape[ 0 ] ) # endog variable,not used
xvar = np.array( xvar )
zvar = np.array( zvar )

class GMMREM( GMM ):

    def momcond(self, params):
        b0, b1 = params
        x = self.exog
        z = self.instrument

        # moment condition of stock return
        m1 = ( z*( b0*( x[ :, 0 ]**( b1 ) )*x[ :, 1 ] - 1 ).reshape( -1, 1
        ↪  ) )

        # moment condition for risk-free
        m2 = ( z*( b0*( x[ :, 0 ]**( b1 ) )*x[ :, 2 ] - 1 ).reshape( -1, 1
        ↪  ) )
        return np.column_stack(( m1, m2 ))

# 2 Euler functions with 4 instruments in each equation
model1 = GMMREM( yvar, xvar, zvar, k_moms=8, k_params=2 )
b0 = [ 1, -1 ]
res1 = model1.fit( b0, maxiter=100, optim_method='bfgs' )
```

Table 5: This table presents estimates of the discount factor, β, and the risk aversion, γ, using the GMM. The parameters are computed using the nominal value of consumption growth, the return on risky assets, and the risk-free return. The instrumental variables are the one-period lags of each variable, namely consumption growth, return on risky assets and risk-free return.

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
β	1.0035	0.003	400.652	0.000	0.999	1.008
γ	-1.5927	0.440	-3.621	0.000	-2.455	-0.731

Table 6: This table presents estimates of the discount factor, β, and the risk aversion, γ, using the GMM. The parameters are computed using the real value of consumption growth, the return on risky assets, and the risk-free return. The real value are obtained by dividing the gross return by the gross rate of inflation. The instrumental variables are the one-period lags of each variable, namely consumption growth, return on risky assets and risk-free return.

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
β	1.0033	0.001	775.929	0.000	1.001	1.006
γ	-2.2850	0.523	-4.372	0.000	-3.309	-1.261