Empirical Asset Pricing - Problem set 1 - Disem Sula

February 26, 2025

In this problem set we will study the properties of the bias correction presented in Stambaugh (1999) Predictive Regressions, Journal of Financial Economics.

1 Question 1

For this question we create a function (routine) that given a set of parameters, and a sample size T simulates the dynamics of the system:

$$y_{t+1} = \alpha + \beta\, x_t + u_{t+1}$$

$$x_{t+1} = \theta + \rho\, x_t + \nu_{t+1}$$

where u and follow a bivariate normal distribution ($\mathcal{N}(0, \Sigma)$).

We then plot the dynamics of x_t and y_t for T = 100

We first define the function:

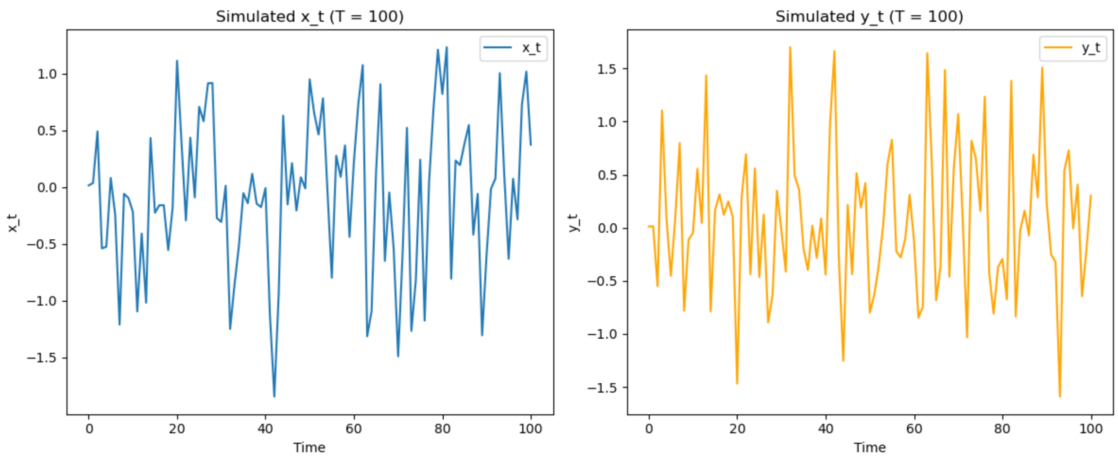
[3]:
```
import numpy as np
import matplotlib.pyplot as plt

def simulate_system(params, T): #a function that simulates the system above, we define the parameters in a dictionary. It returns the simulated time series for the given sample size T

    # we create the parameters
    alpha      = params.get('alpha', 0.01)
    beta       = params.get('beta', 0.05)
    theta      = params.get('theta', 0.01)
    rho        = params.get('rho', 0.3)
    sigma_u2   = params.get('sigma_u2', 0.6)
    sigma_nu2  = params.get('sigma_nu2', 0.5)
    sigma_u_nu = params.get('sigma_u_nu', -0.5)

    # we create the covariance matrix for residuals [u,  ]
    cov = np.array([[sigma_u2, sigma_u_nu],
                    [sigma_u_nu, sigma_nu2]])
```

```
    # we create empty arrays for x and y (we simulate T+1 points to include x0␣
↪and y0)
    x = np.zeros(T+1)
    y = np.zeros(T+1)

    # as requested we assume that initial values are their unconditional means
    x[0] = theta / (1 - rho)
    y[0] = alpha + beta * x[0]

    # we simulta the system with a loop up to the sample size T, where we␣
↪generate a pair u, nu from a bivariate normal distribution (correlation is␣
↪handled automatically)
    for t in range(T):
        # as reminded in the description, residuals are correlated
        u, nu = np.random.multivariate_normal(mean=[0, 0], cov=cov)

        # Update x and y empty arrays previously created according to the model
        x[t+1] = theta + rho * x[t] + nu
        y[t+1] = alpha + beta * x[t] + u

    return x, y
```

We can now simulate the dynamics of the system above with different parameters or sample size

[4]:
```
params = {
    'alpha': 0.01,
    'beta': 0.05,
    'theta': 0.01,
    'rho': 0.3,
    'sigma_u2': 0.6,
    'sigma_nu2': 0.5,
    'sigma_u_nu': -0.5
}

T = 100

#after defining parameters and sample size we can simulate the system and then␣
↪simply plot the time series side by side
x_series, y_series = simulate_system(params, T)


plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(x_series, label='x_t')
plt.title('Simulated x_t (T = 100)')
plt.xlabel('Time')
```

```
plt.ylabel('x_t')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(y_series, color='orange', label='y_t')
plt.title('Simulated y_t (T = 100)')
plt.xlabel('Time')
plt.ylabel('y_t')
plt.legend()

plt.tight_layout()
plt.show()
```



2 Question 2

Here we will create a function that given simulated data, estimates all of the parameters of the system above using OLS.

So the idea is to estimate the two separate regressions using OLS, considering that both regressions use the same regressor: the lagged value of x.

I am going to use the **statsmodels package** as it can be useful to have more details and not only the parameters, especially in other questions.

[5]:
```
import statsmodels.api as sm

def estimate_parameters_sm(x, y): #the two arguments are the numpy arrays
↪representing the simulated time series with T+1 opbservations including the
↪initial value
    T = len(x) - 1 #number of regression observation excluding the initial value
```

```
    X_lag = sm.add_constant(x[:T])  # independent variable matrix, adds a␣
 ↪column of ones to x[:T] to account for intercepot term
    #X_lag is now a matric with one constant column for the intercept and one␣
 ↪for the lagged x (x_t)

#the regression will use x[0] to x[T-1] as regressor (lagged x) and y[1:] or␣
 ↪x[1:] as dependent variable

    # Regression for y_{t+1}
    model_y = sm.OLS(y[1:], X_lag) #y[1:] gives the dependent variable
    results_y = model_y.fit()

    # Regression for x_{t+1}
    model_x = sm.OLS(x[1:], X_lag)
    results_x = model_x.fit()

    estimates = {
        'alpha': results_y.params[0],
        'beta': results_y.params[1],
        'theta': results_x.params[0],
        'rho': results_x.params[1],
        'y_summary': results_y.summary(),
        'x_summary': results_x.summary()
    }

    return estimates #we will get not only the parameters but also the full␣
 ↪summaries
```

3 Question 3

We fix a sample size of T = 100 and perform N = 10,000 simulations of the system above.

For every simulation we estimate β.

And finally plot the distribution of $\hat{\beta}$ and show graphically how the estimator is biased.

We have already define from question 1 and 2 the simulation of the system and the estimation of the parameters functions.

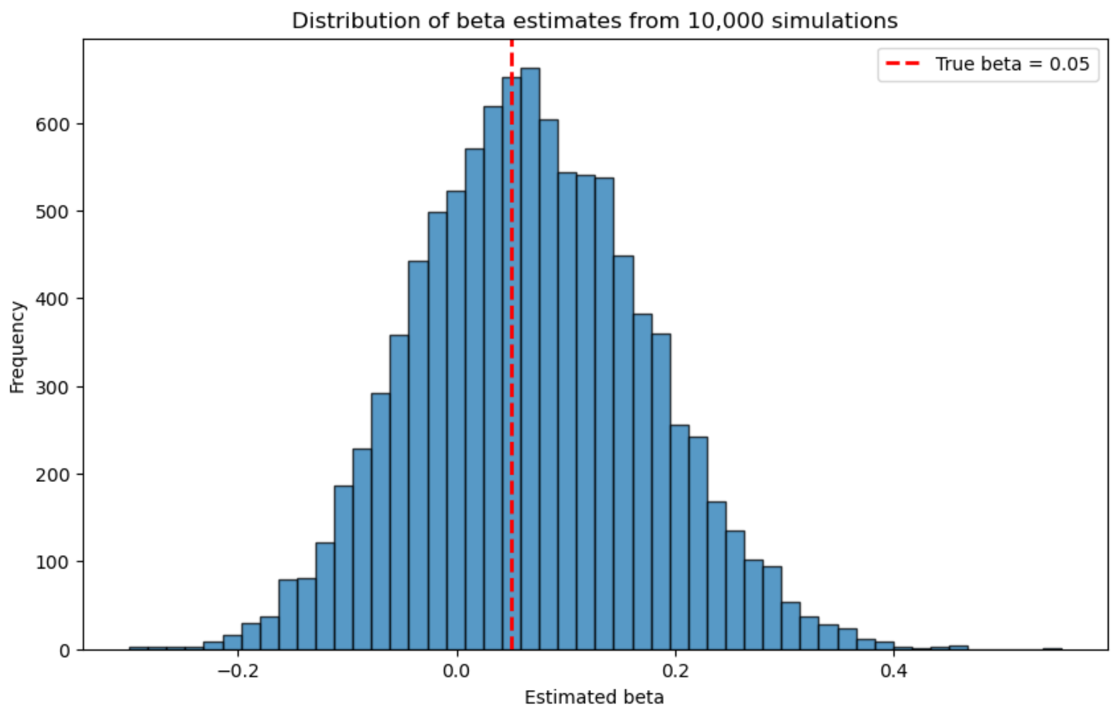
Now we just need to create a **loop of N simulations** and for each simulation (building on Q1) we use the estimated parameters function and we extract β, the coefficient from the regression of y_{t+1} on x_t and store it.

[6]:
```
N = 10000
T = 100     # Sample size for each simulation

beta_estimates = np.zeros(N) #we need an empty arrays of size N to store the␣
 ↪betas
```

```
for i in range(N):
    x_sim, y_sim = simulate_system(params, T)
    estimates = estimate_parameters_sm(x_sim, y_sim)
    beta_estimates[i] = estimates['beta']
```

We can now ***plot the distribution*** using matplotlib to plot the histogrram of beta estimates. The red dashed lines is there to mark the true value of beta as stated in the problem set setup: $\beta = 0.05$

[7]:
```
plt.figure(figsize=(10, 6))
plt.hist(beta_estimates, bins=50, edgecolor='black', alpha=0.75)
plt.axvline(x=params['beta'], color='red', linestyle='dashed', linewidth=2,␣
 ↪label='True beta = 0.05')
plt.title('Distribution of beta estimates from 10,000 simulations')
plt.xlabel('Estimated beta')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



[8]:
```
print("Mean beta estimate:", np.mean(beta_estimates))
```

```
Mean beta estimate: 0.06827930544385082
```

We can see graphically that the β estimate is slightly biased, the mean of the β estimate shows the bias more clearly

4 Question 4

We will now fix N = 100 and compute the bias of $\hat{\beta}$ for different sample sizes (T). Finally we will plot the bias for 500 different points in the interval [50, 1000]

So the idea is to look at how the bias in $\hat{\beta}$ varies with the sample size T. We want to have 100 simulations for each sample size T using 500 different sample sizes in the interval [50, 1000]. For each T we run 100 simulation, obtain the average $\hat{\beta}$, compute the bias and then plot these as a function of T.

[9]:
```
'''
N = 100  # simulations for each T
T_values = np.linspace(50, 1000, 500).astype(int) #500 sample sizes between␣
↪[50, 1000]

bias_values = []

# Loop over the different sample sizes
for T in T_values:
    beta_estimates = []
    # For each T, run N simulations, exaclty as in Q3
    for i in range(N):
        x_sim, y_sim = simulate_system(params, T)
        estimates = estimate_parameters_sm(x_sim, y_sim)
        beta_estimates.append(estimates['beta'])

    # Calculate the bias (average beta estimate - true beta)
    avg_beta = np.mean(beta_estimates)
    bias = avg_beta - params['beta']
    bias_values.append(bias)
'''
```

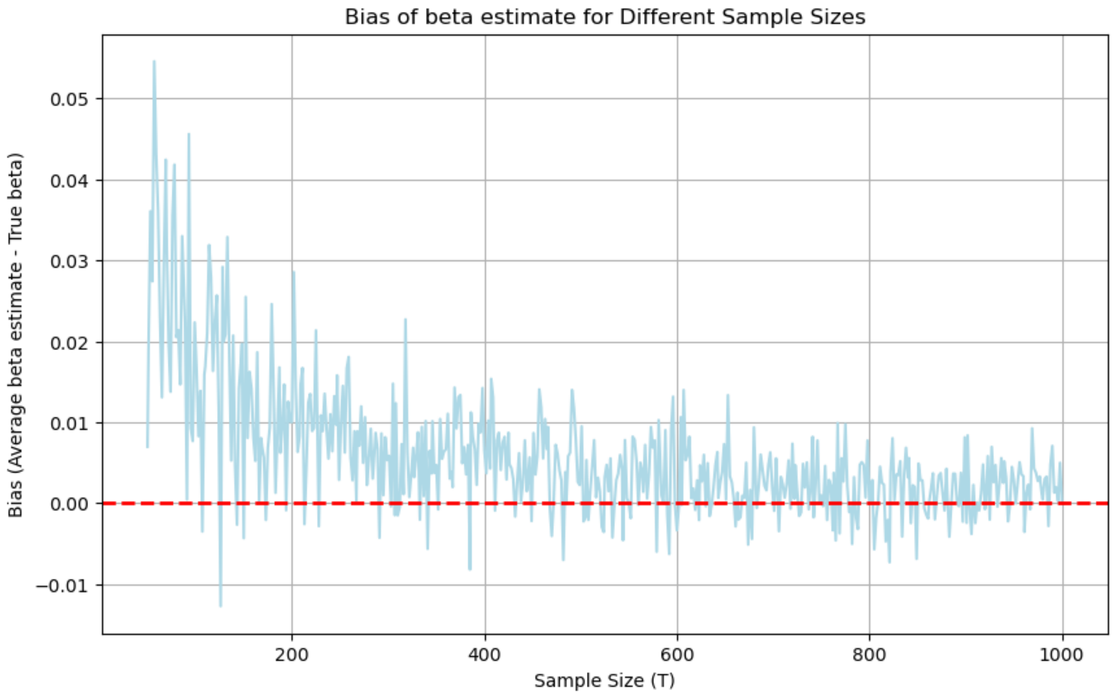
[9]:
```
"\nN = 100  # simulations for each T\nT_values = np.linspace(50, 1000,
500).astype(int) #500 sample sizes between [50, 1000]\n\nbias_values = []\n\n#
Loop over the different sample sizes\nfor T in T_values:\n    beta_estimates =
[]\n    # For each T, run N simulations, exaclty as in Q3\n    for i in
range(N):\n        x_sim, y_sim = simulate_system(params, T)\n        estimates
= estimate_parameters_sm(x_sim, y_sim)\n
beta_estimates.append(estimates['beta'])\n    \n    # Calculate the bias
(average beta estimate - true beta)\n    avg_beta = np.mean(beta_estimates)\n
bias = avg_beta - params['beta']\n    bias_values.append(bias)\n"
```

It took 12 minutes for all the simulations to complete and calculate all biases, in the next code I will try to use python joblib library parallel anbd delayed functions to reduce runtime, splitting the work through multiple CPU cores

```
[10]: from joblib import Parallel, delayed
# Function to run N simulations for a given T and return the bias in beta
def simulate_bias_for_T(T, N=100):
    beta_estimates = []
    for i in range(N):
        x_sim, y_sim = simulate_system(params, T)
        estimates = estimate_parameters_sm(x_sim, y_sim)
        beta_estimates.append(estimates['beta'])
    avg_beta = np.mean(beta_estimates)
    bias = avg_beta - params['beta']
    return bias

# Settings: 500 sample sizes between 50 and 1000, and N simulations per T.
T_values = np.linspace(50, 1000, 500).astype(int)
N = 100  # simulations for each T

# Use joblib's Parallel to process each T value in parallel.
bias_values = Parallel(n_jobs=-1)(
    delayed(simulate_bias_for_T)(T, N) for T in T_values
)
```

Once the loop is completed, and the biases have been calculated, we can plot the bias as a function of T

```
[11]: plt.figure(figsize=(10, 6))
plt.plot(T_values, bias_values, linestyle='-', color='lightblue')
plt.axhline(0, color='red', linestyle='--', linewidth=2)
plt.title("Bias of beta estimate for Different Sample Sizes")
plt.xlabel("Sample Size (T)")
plt.ylabel("Bias (Average beta estimate - True beta)")
plt.grid(True)
plt.show()
```



5 Question 5

Using the results from the question above, here we will fit the below regression using OLS:

$$\text{bias}_i = \gamma_0 + \gamma_1 \left(\frac{1}{T_i}\right) + \gamma_2 \left(\frac{1}{T_i^2}\right) + \varepsilon_i$$

We will compute the t-statistics of all coefficients and compare ν_1 with the equivalent term in the Stambaugh's bias definition.

In question 4 we obtained for each sample size T (over 500 values between 50 and 1000), an estimate of the bias defined as

$$\text{bias}_i = \left(\text{average } \hat{\beta} \text{ at sample size } T_i\right) - \beta_{\text{true}}$$

Stambaugh (1999) shows that the bias in the OLS estimator of β can be approximated by terms that are proportional to $\left(\frac{1}{T}\right)$ and $\left(\frac{1}{T^2}\right)$.

Here, we have to fit the regression

$$\text{bias}_i = \gamma_0 + \gamma_1 \left(\frac{1}{T_i}\right) + \gamma_2 \left(\frac{1}{T_i^2}\right) + \varepsilon_i$$

using OLS. The estimated ν_1 should capture the $\left(\frac{1}{T}\right)$ bias.

[12]:
```python
# first thing we have to create thet two regressors
T_inv = 1 / T_values
T_inv2 = 1 / pow(T_values,2)

# similar to Q2 we add a constant for the intercept
X = np.column_stack([np.ones(len(T_values)), T_inv, T_inv2])
y_bias = np.array(bias_values)  # dependent variable (bias at each T), as
↪calculated before

model = sm.OLS(y_bias, X) #create our model with the dependent variable and the
↪two regressor as we created before the "X"
results = model.fit() #we simply fit the regression to the model above

print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.509
Model:                            OLS   Adj. R-squared:                  0.507
Method:                 Least Squares   F-statistic:                     257.4
Date:                Wed, 26 Feb 2025   Prob (F-statistic):           1.91e-77
Time:                        09:38:40   Log-Likelihood:                 1859.1
No. Observations:                 500   AIC:                            -3712.
Df Residuals:                     497   BIC:                            -3699.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0011      0.001     -1.903      0.058      -0.002    3.47e-05
x1             2.4778      0.246     10.071      0.000       1.994       2.961
x2           -43.1307     15.517     -2.780      0.006     -73.618     -12.643
==============================================================================
Omnibus:                       46.315   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              243.217
Skew:                          -0.097   Prob(JB):                     1.53e-53
Kurtosis:                       6.411   Cond. No.                     5.89e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.89e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

More specifically we can look directly at it extracting coefficients and t-statistics:

[13]:

```
gamma0, gamma1, gamma2 = results.params
tstats = results.tvalues
print(f"gamma_0 = {gamma0:.4f}, t-stat = {tstats[0]:.4f}")
print(f"gamma1 = {gamma1:.4f}, t-stat = {tstats[1]:.4f}")
print(f"gamma2 = {gamma2:.4f}, t-stat = {tstats[2]:.4f}")
```

```
gamma_0 = -0.0011, t-stat = -1.9025
gamma1 = 2.4778, t-stat = 10.0711
gamma2 = -43.1307, t-stat = -2.7795
```

Intercept (γ_0) estimated is very close to zero and not statistically significant. This is expected because, as per the theory, as T grows large the bias should vanish, leaving no constant bias term.

The coefficient on $\left(\frac{1}{T^2}\right)$ ($\gamma_2 \approx -43.13$) is intended to capture higher-order corrections to the bias. Its estimation is not statistically significant, which can be attributed to the high multicollinearity between $\left(\frac{1}{T}\right)$ and $\left(\frac{1}{T^2}\right)$ over the range of T values.

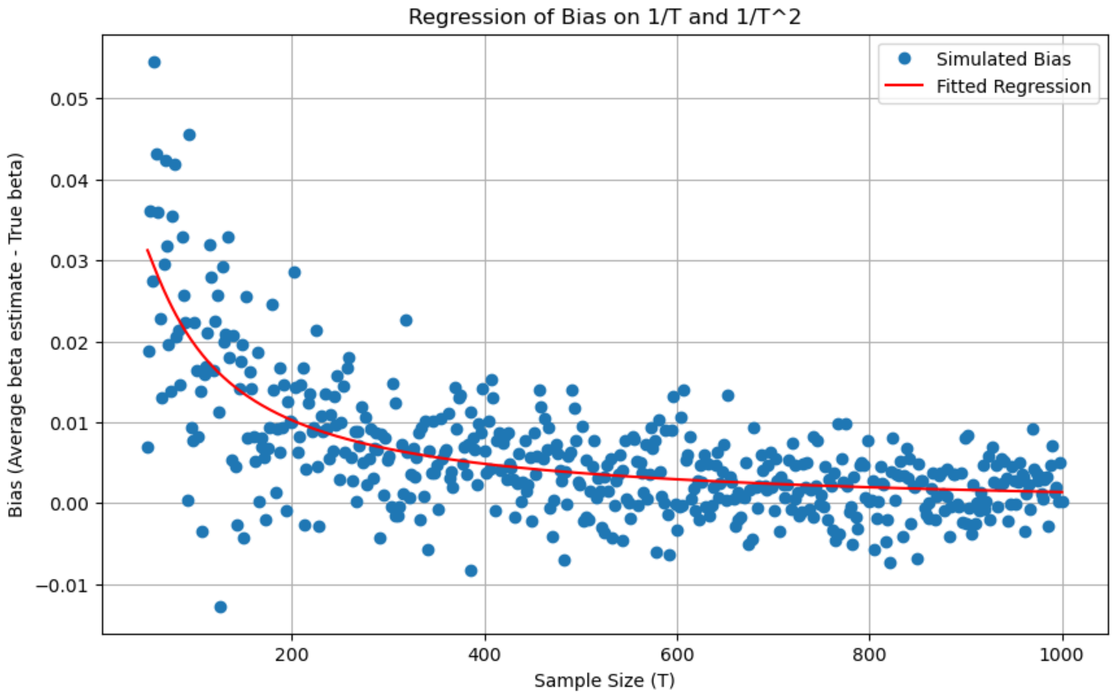
The sign of (γ_1) (positive) does match the sign implied by: $-\left(\frac{\sigma_{u\nu}}{\sigma_u^2}\right)(1-\rho)$

given $\sigma_{u\nu} < 0$. So our results are consistent with Stambaugh's insight that negative $\sigma_{u\nu}$ can produce an upward small-sample bias in $\hat{\beta}$. However, because we are not in a "large-T, $\rho \approx 1$" setting and we have a second regressor $\left(\frac{1}{T^2}\right)$, the magnitude (2.48) vs. the theoretical (0.58) is inflated.

Below I have added a **visualization** of the simulated bias for all the different sample sizes that we have used within [50, 1000] and the fitted regression we just created.

[14]:

```
fitted_bias = results.predict(X)
plt.figure(figsize=(10, 6))
plt.plot(T_values, y_bias, 'o', label='Simulated Bias')
plt.plot(T_values, fitted_bias, 'r-', label='Fitted Regression')
plt.xlabel('Sample Size (T)')
plt.ylabel('Bias (Average beta estimate - True beta)')
plt.title('Regression of Bias on 1/T and 1/T^2')
plt.legend()
plt.grid(True)
plt.show()
```



Regression of Bias on 1/T and 1/T^2
Simulated Bias
Fitted Regression
Bias (Average beta estimate - True beta)
Sample Size (T)
0.05
0.04
0.03
0.02
0.01
0.00
−0.01
200
400
600
800
1000