QMB 3311: Python for Business Analytics
Department of Economics
College of Business
University of Central Florida
Spring 2022

Assignment 4

Due Sunday, February 20, 2021 at 11:59 PM
in your GitHub repository

Instructions:

Complete this assignment within the space on your *private* GitHub repo (not a fork of the course repo QMB3311S22!) in a folder called `assignment_04`. In this folder, save your answers to Questions 1 and 2 in a file called `my_A4_functions.py`, following the sample script in the folder `assignment_04` in the course repository. When you are finished, submit it by uploading your files to your GitHub repo using any one of the approaches outlined in Question 3. You are free to discuss your approach to each question with your classmates but you must upload your own work.

Please note: In computer programming, many small details are very important. A file with the wrong name in the wrong folder will not run, even if the functions work perfectly.

Question 1:

Follow the function design recipe to define functions for all of the following Exercises. For each function, create three examples to test your functions. Record the definitions in the sample script `my_A4_functions.py`

Exercise 1 Write a function `matrix_inverse(mat_in)` that replicates the `numpy` method `linalg.inv()` that calculates the inverse of the two-by-two matrix `mat_in`.

The inverse of the matrix A, denoted A^{-1}, is a matrix the same size as A such that $A \cdot A^{-1} = I$ and $A^{-1} \cdot A = I$, where I is the identity matrix with ones on the diagonal and zeroes elsewhere. It can be used to solve the system of equations $A \cdot x = b$ for x by multiplying A^{-1} with b to get $x = A^{-1} \cdot b$.

For a two-by-two matrix A, the inverse can be obtained with the expression

$$A^{-1} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}$$

where the matrix A is defined as

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}.$$

- Your function should take in a two-by-two `numpy` array and output a `numpy` array `mat_out` with the same number of rows and columns as `mat_in`, that is, two-by-two, following the definition of A^{-1} above, when the argument A is passed as `mat_in`.
- It should use two nested loops, one for the row of the output and one for the column of the output.
- The return value should be a matrix that solves the system of equations $A \cdot x = b$ by multiplying the output with b.
- You can use the command `mat_in.dot(mat_out)` to produce the output for two test cases: it should be the identity matrix with ones on the diagonal and zeroes elsewhere. The command `mat_out.dot(mat_in)` will have the same result.
- You can use the command `mat_in.dot(x)` to produce the output for another test case: `mat_in.dot(x)` returns a value of `b`. Then `mat_out.dot(b)` returns the original value of `x`.
- Carefully inspect the expressions in A^{-1} and note any potential problems. Your function should return `None` and print a warning message if the problem occurrs. Use this condition to formulate a fourth test case.

Exercise 2 The likelihood function of the logistic regression model is used to estimate coefficients in logistic regression. Logistic regression is used to model binary events, i.e. whether or not an event occurred. For each observation i, the observation y_i equals 1 if the event occurred and 0 if it did not. Build on the function `logit_like()` from Assignment 3 and write a python function `logit_like_sum()` that calculates the sum of the log-likelihood across all observations (y_i, x_i), $i = 1, \cdots, n$. That is, it returns the sum of either the log of the function $\ell(x_i; \beta_0, \beta_1)$ if $y_i = 1$ or the log of the function $(1 - \ell(x_i; \beta_0, \beta_1))$ if $y_i = 0$, over all observations $i = 1, \cdots, n$:

$$L(y, x; \beta_0, \beta_1) = \sum_{i=1}^{n} L_i(y_i, x_i; \beta_0, \beta_1)$$

where $L_i(y_i, x_i; \beta_0, \beta_1)$ refers to the log of either ℓ or $(1 - \ell)$ as described above, with the logit link function ℓ defined as

$$\ell(x_i; \beta_0, \beta_1) = Prob(y = 1|x) = \frac{e^{x_i\prime\beta}}{1 + e^{x_i\prime\beta}} = \frac{e^{\beta_0 + x_i\beta_1}}{1 + e^{\beta_0 + x_i\beta_1}}.$$

This function `logit_like_sum()` will have the four arguments in $L(y, x; \beta_0, \beta_1)$, in that order, where the sample of observations y and x are both either lists or `numpy` arrays.

Exercise 3 The *gradient vector* of a multivariate function is a vector with each element equal to the derivative of the function with respect to each parameter. In the case of $L(y, x; \beta_0, \beta_1)$, element k is

$$\frac{\partial L(y, x; \beta_0, \beta_1)}{\partial \beta_k} = \sum_{i=1}^{n} \frac{\partial}{\partial \beta_k} L_i(y_i, x_i; \beta_0, \beta_1)$$

for $k = 0$ or $k = 1$, corresponding to β_0 or β_1, where $L_i(y_i, x_i; \beta_0, \beta_1)$ is defined in Exercise 2 above.

Using calculus, one can determine that

$$\frac{\partial}{\partial \beta_k} L_i(y_i, x_i; \beta_0, \beta_1) = \begin{cases} d_i(1 - \ell(x_i; \beta_0, \beta_1)), & \text{if } y_i = 1, \\ d_i(-\ell(x_i; \beta_0, \beta_1)), & \text{if } y_i = 0, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

where

$$d_i = \begin{cases} 1, & \text{for } k = 0, \\ x_i, & \text{for } k = 1. \end{cases}$$

Define a function `logit_like_grad()` that will have the four arguments in $(y, x; \beta_0, \beta_1)$, in that order, and will output a vector of two elements, corresponding to the parameters β_0 and β_1. Your manager consulted an expert in econometrics, who provided some test cases in the script `my_A4_functions.py`.

Exercise 4 For Assignment 3, you wrote a function `CESutility_valid()` that calculated the value of the Constant Elasticity of Substitution utility function $u(x, y; r) = (x^r + y^r)^{\frac{1}{r}}$, for valid values of the parameters x, y, and r. Now, extend this function to evaluate the consumer's utility for more than two goods:

$$u(\mathbf{x}, \mathbf{a}; r) = \left(\sum_{i=1}^{n} a_i^{1-r} x_i^r \right)^{\frac{1}{r}},$$

where $\mathbf{x}$ is a vector of quantities of goods consumed and $\mathbf{a}$ is a vector of weighting parameters for each good and the subscript i indicates the ith element of each vector.

In this function, the first two arguments are $\mathbf{x}$ and $\mathbf{a}$, and the third is still r. Call this function `CESutility_multi(x, a, r)` and make sure to include the logic to determine whether the inputs are valid, as in `CESutility_valid()`, except all elements of $\mathbf{x}$ and $\mathbf{a}$ must be nonnegative, and return `None` otherwise.

Question 2:

For all of the Exercises in Question 1, use your examples to test the functions you defined. Since the examples are all contained within the docstrings of your functions, you can use the `doctest.testmod()` function within the `doctest` module to test your functions automatically. Add some code to the sample script `my_A4_functions.py`, run the entire script, and paste the output to a file called `my_A4_functions.out`.

Don't worry about false alarms: if there are some "failures" that are only different in the smaller decimal places, then your function is good enough. It is much more important that your function runs without throwing an error.

Question 3:

Push your completed files to your GitHub repository following one of these three methods.

Method 1: In a Browser

Upload your code to your GitHub repo using the interface in a browser.

1. Browse to your `assignment_0X` folder in your repository (the "X" corresponds to Assignment X.).

2. Click on the "Add file" button and select "Upload files" from the drop-down menu.

3. Revise the generic message "Added files via upload" to leave a more specific message. You can also add a description of what you are uploading in the field marked "Add an optional extended description..."

4. Press the button "'Commit changes," leaving the buton set to "Commit directly to the `main` branch."

Method 2: With GitHub Desktop

Upload your code to your GitHub repo using the interface in GitHub Desktop.

1. Save your file within the folder in your repository within the folder referenced in GitHub Desktop.

2. When you see the changes in GitHub Desktop, add a description of the changes you are making in the bottom left panel.

3. Press the button "Commit to main" to commit those changes.

4. Press the button "Push origin" to push the changes to the online repository. After this step, the changes should be visible on a browser, after refreshing the page.

Method 3: At the Command Line

Push your code directly to the repository from the command line in a terminal window, such as GitBash on a Windows machine or Terminal on a Mac.

1. Open GitBash or Terminal and navigate to the folder inside your local copy of your git repo containing your assignments. Any easy way to do this is to right-click and open GitBash within the folder in Explorer. A better way is to navigate with UNIX commands, such as `cd`.

2. Enter `git add .` to stage all of your files to commit to your repo. You can enter `git add my_filename.ext` to add files one at a time, such as `my_functions.py` in this Assignment.

3. Enter `git commit -m "Describe your changes here"`, with an appropriate description, to commit the changes. This packages all the `add`ed changes into a single unit and stages them to `push` to your online repo.

4. Enter `git push origin main` to push the changes to the online repository. After this step, the changes should be visible on a browser, after refreshing the page.