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

Midterm Examination

Due Monday, March 14, 2022 at 5:55 PM
in your GitHub repository

Instructions:

Complete this examination within the space of your private GitHub repo in a folder called `midterm_exam`. In this folder, save your answers to Questions 1 and 2 in Python scripts, as described in Question 1. When you are finished, submit it by uploading your files to your GitHub repo using any one of the approaches outlined in Question 4. Since this is an examination, you are NOT free to discuss your approach to each question with your classmates and you must upload your own work.

Question 1:

Module `my_CES_midterm.py`:
Collect the following functions from your previous assignments into a module called `my_CES_midterm.py`.

- `CES_utility()` from Assignment 2.
- `CES_utility_valid()` from Assignment 3.
- `CES_utility_in_budget()` from Assignment 3.

Module `my_logit_midterm.py`:
Collect the following functions from your previous assignments into a module called `my_logit_midterm.py`.

- `logit()` from Assignment 3.
- `logit_like()` from Assignment 3.
- `logit_like_sum()` from Assignment 4.

Module `my_production_midterm.py`:
Collect the following functions from your previous assignments into a module called `my_production_midterm.py`.

- `total_revenue()` from Assignment 2.
- `total_cost()` from Assignment 2.

Question 2:

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 Python scripts as described in Question 1.

Under time constraints, the emphasis is on creating function definitions and testing with examples. You are not responsible for error handling and can assume that the user will follow the preconditions and type contract.

Most importantly, take any shortcut you can, using the code from the course repository and any solutions from your previous assignments. The first step is often to locate a similar function and copy it to your module. Then you can begin to either re-purpose that code or call the function in a new function, and save yourself some time.

Exercise 1 Your client is trying to plan the quantity of goods produced by their company. They produce q units of one product at a total cost of $c(q, a, b) = a \times q^2 + b$, which depends on the quantity produced, and sell the units at price p. In Assignment 2, you have already produced functions for total cost $c(q, a, b)$ and total revenue $p \times q$ and called them `total_cost(num_units, multiplier, fixed_cost)` and `total_revenue(num_units, unit_price)`. Now write a function that calculates the profit earned by the company when they produce and sell q units, defined as $\pi(q, p, a, b) = p \times q - c(q, a, b)$. Call your function `total_profit(num_units, unit_price, multiplier, fixed_cost)`, which returns a float that represents the expected profit. Add this function to your module `my_production_midterm.py`. No error messages are necessary, assuming the user will only input values that are `float`s.

Exercise 2 Now write a function that calculates the company's optimal production level, that is, the amount $q^* \geq 0$ that maximizes the profit function $\pi(q, p, a, b) = p \times q - c(q, a, b)$. Using calculus, and assuming $a > 0$, you can show that the following choices maximize the company's profit:

$$q^* = \frac{p}{2a},$$

where a is the cost multiplier in the cost function $c(q, a, b) = a \times q^2 + b$, with fixed cost b. It is possible, however, that this level of production still returns a negative profit, in which case, it would be better not to enter the market in the first place. The full solution is thus

$$q^{max} = \begin{cases} q^*, & \text{if } \pi(q^*, p, a, b) > 0, \\ 0, & \text{if otherwise.} \end{cases}$$

Write a function that returns a float q^{max} called `max_profit_calc(unit_price, multiplier, fixed_cost)`.

Exercise 3 Now verify the derivation of the above solution. Use a grid search on a range of values from zero to `q_max` in increments of size `step`. Write this in a function `profit_max_q(q_max, step, unit_price, multiplier, fixed_cost)`. In this algorithm, you could loop over a single index number `i`, which selects values of `q` from an array `q_list` and evaluates your function `total_profit(num_units = q, unit_price, multiplier, fixed_cost)`.

Exercise 4 Write a function definition `max_CES_util(x_min, x_max, y_min, y_max, step, r, p_x, p_y, w)` that returns the *maximum value* u^* of the CES utility function:

$$u^*(r, p_x, p_y, w) = ((x^*)^r + (y^*)^r))^{\frac{1}{r}}, \quad \text{subject to} \quad p_x \times x^* + p_y \times y^* \leq w,$$

where x^* and y^* are the values of `x` and `y` that maximize `CESutility_in_budget(x, y, r, p_x, p_y, w)` for given r, p_x, p_y, and w. Recall that you may re-use any code or call any function from previous assignments or course material. Add this function to your module `my_CES_midterm.py`.

Exercise 5 Now find values of `beta_0_hat` and `beta_1_hat` that maximize `logit_like_sum(y, x, beta_0, beta_1)` by searching over both `beta_0` and `beta_1` independently. Write a function `max_logit(y, x, beta_0_min, beta_0_max, beta_1_min, beta_1_max, step)` as follows:

a) Find the values of `beta_0` and `beta_1` by evaluating `logit_like_sum(y, x, beta_0, beta_1)` over every combination of of (β_0, β_1) in two lists.

b) Create lists `beta_0_list` and `beta_1_list` from ranges $\beta_0 = \beta_0^{min}, \ldots, \beta_0^{max}$ and $\beta\_1 = \beta_1^{min}, \ldots, \beta_1^{max}$, where the neighboring values of β_0 or β_1 are separated by distance `step`. The `np.arange()` function is useful for this.

c) Initialize the maximized value with `max_logit_sum = float("-inf")`.

d) Loop over the index numbers `i` and `j`, corresponding to lists `beta_0_list` and `beta_1_list`.

   i) For each pair of `i` and `j`, extract the value `beta_0_list[i]` and `beta_1_list[j]`.

   ii) For each pair of `i` and `j`, evaluate `logit_like_sum(y, x, beta_0, beta_1)`.

   ii) If the value is higher than `max_sum_logit`, record the new `i_max = i` and `j_max = j` and update the newest value of `max_sum_logit`.

f) After the loops, return `[ beta_0_list[i_max], beta_1_list[j_max] ]`.

g) Verify that the result matches the values in the test cases supplied with the sample script `my_logit_midterm.py` (up to accuracy `step`). Add this function to your module `my_logit_midterm.py`.

As in the other questions, you may re-use any code or call any function from previous assignments or course material.

Question 3:

For all of the Exercises in Question 2, and the functions listed above, use examples to test the functions you defined. Since the examples will all be contained within the docstrings of your functions, you can use the `doctest.testmod()` function within the `doctest` module to test your functions automatically at the bottom of the modules. Use an `if` statement to determine whether the tests will be run using `testmod`: if the script is executed as a script, i.e. as the `__main__` script, then run the tests; if it is imported, then skip the tests.

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 4:

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 ("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 "'Commit changes," leaving the button 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 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.