ECP 3004: Python for Business Analytics
Department of Economics
College of Business
University of Central Florida
Spring 2021

Assignment 7

Due Sunday, April 11, 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 ECP3004S21!) in a folder called `assignment_07`. In this folder, save your answers to Questions 1 and 2 in a file called `my_A7_functions.py`, following the sample script in the folder `assignment_07` 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_A7_module.py`

Example 1 Consider the function $g(x) = (x - 2)x(x + 2)^2$. Write three Python functions: one for the function `g(x)`, one for the first derivative `g_prime(x)`, and one for the second derivative `g_2prime(x)`.

Example 2 Write a function `newton_g_opt(x_0, maxiter, tol)` that finds the minimum of $g(x)$ using Newton's method. Test your function with examples using three different starting values `x_0` and tolerance levels `tol`.

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.

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:

The sample script `logit_calculation.py` uses the `statsmodels` module to estimate a model for the probability that borrowers will default on their loans. You will calculate the parameter estimates by applying optimization methods in `scipy` to estimate these parameters. The dataset `credit_data.csv` in `demo_19_Classification` includes the following variables.

default:	1 if borrower defaulted on a loan
bmaxrate:	the maximum rate of interest on any part of the loan
amount:	the amount funded on the loan
close:	1 if borrower takes the option of closing the listing until it is fully funded
AA:	1 if borrowers FICO score greater than 760
A:	1 if borrowers FICO score between 720 and 759
B:	1 if borrowers FICO score between 680 and 719
C:	1 if borrowers FICO score between 640 and 679
D:	1 if borrowers FICO score between 600 and 639

In the script `logit_calculation.py`, these data are loaded in and used to estimate a model using `statsmodels`, with only the variables relating to FICO scores to predict `default`. The function `logit_likelihood(beta, y, X)` is the (negative of the) log-likelihood function that is maximized to get the parameter estimates in `statsmodels`. The function `logit_gradient(beta, y, X)` is the (negative of the) first derivative of the log-likelihood function, which is zero at the maximal parameter values. The function `logit_hessian(beta, y, X)` is the (negative of the) matrix of second derivatives of the log-likelihood function. These functions are already defined, since their calculation is the subject of a more advanced course. **No examples are necessary, since I have already tested the functions. All you need to do is obtain the coefficients by optimization, filling in the code in `logit_calculation.py` wherever it is marked** `Code goes here`.

a) Run the script `logit_calculation.py` up to line 140 to see the results for the estimation with `statsmodels`. The goal is to match the parameter estimates in `logit_model_fit_sm.params` and achieve the maximum value of the log-likelihood function shown in the output from `logit_model_fit_sm.summary()`.

b) Calculate the parameter estimates by minimizing `logit_likelihood(beta, y, X)` using the function `minimize()` from the `scipy` module and passing the tuple of arguments `(y, X)`. Implement it several times using the following algorithms.

   i) Use the Nelder-Mead Simplex algorithm algorithm by passing the argument `method = 'nelder-mead'`.

   ii) Use the Davidon-Fletcher-Powell (DFP) algorithm by passing the argument `method = 'powell'`.

   iii) Use the Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS) algorithm by passing the argument `method = 'BFGS'`.

   iv) Use another version of the BFGS algorithm. This time, pass the additional argument `jac = logit_gradient` to use the first derivative to calculate the iterations within the algorithm.

v) Use the Newton-Conjugate-Gradient (NCG) algorithm by passing the argument `method = 'Newton-CG'`. This time, pass the additional arguments `jac = logit_gradient`, to use the first derivative vector, `hessp = logit_hessian`, to use the second derivative matrix, to calculate the iterations within the algorithm.

c) Verify that your parameter estimates and the optimal values of the likelihood function are achieved with the methods in part (b), to match the results from `statsmodels`. You may need to pass additional arguments to the `options` argument, as in:

```
options = {'xtol':  1e-8, 'maxiter':  1000, 'disp':  True}
```

and to adjust the values as necessary. Compare the accuracy and number of iterations.

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 (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.