`Assignment 7 --- QMB 3311 --- Spring 2023 --- Due:  April 9, 2023`

Instructions:

Complete this assignment within the space on your *private* GitHub repo in a folder called `Assignment_07`. In this folder, save your answers to a file called `my_logit_A7.py` in the course repository. In the same folder, submit the script `logit_calculation_A7.py`When you are finished, submit your files to your repository and upload the link to Webcourses.

For all of the exercises, 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.

1. Create the following functions from your previous assignments into a module called `my_logit_A7.py`.
   - `logit()` from Assignment 3.
   - `logit_like()` from Assignment 3.
   - `logit_like_sum()` from Assignment 4.
   - `logit_di()` from Assignment 5.
   - `logit_dLi_dbk()` from Assignment 5.

2. Follow the function design recipe to define functions for all of the following parts. For each function, create three examples to test your functions.

   (a) Write a function `logit_like_sum_opt(beta, y, x)` that is a wrapper function for the function called `logit_like_sum()` that can be used for optimization. The parameter for optimization `beta` should be a single parameter containing both `beta_0` and `beta_1` and it should be the first parameter. Alsom the function should return the negative of `logit_like_sum()` since the `scipy` algorithms minimize the objective function and the estimates of `beta` are those that maximize `logit_like_sum()`.

   (b) Write a function `logit_like_grad(beta, y, x)` to calculate the gradient vector of the like- lihood function `logit_like_sum_opt()`. It should take the same arguments as the function `logit_like_sum_opt()`, in the same order. 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_0, where $L_i(y_i, x_i; \beta_0, \beta_1)$ 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

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

We coded these functions in Assignment 5. Using these expressions, you can define this function `logit_like_grad()` with the sum of the calculations of `logit_dLi_dbk(y_i, x_i, k, beta_0, beta_1)` from Assignment 5. Your function will output a vector of two elements, corresponding to the parameters β_0 (for `k` = 0) and β_1 (for `k` = 1). Like the objective function above, this new function will take the negative of the sum of the terms in `logit_dLi_dbk(y_i, x_i, k, beta_0, beta_1)`, where `i` loops over the elements in the lists `y` and `x`.

3. The sample script `logit calculation A7.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` 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 A7.py`, these data are loaded in and used to estimate a model using statsmodels, with only the variable bmaxrate. We use the functions defined above in exercise 1 and 2 above. The function `logit_like_sum_opt(beta, y, x)` is the (negative of the) log-likelihood function that is maximized to get the parameter estimates in statsmodels. The function `logit_like_grad(beta, y, x)` is the (negative of the) first derivative of the log-likelihood function, which is zero at the maximal parameter values. No examples are necessary, since you have already tested the functions. All you need to do is obtain the coefficients by optimization, filling in the code in `logit_calculation_A7.py` wherever it is marked `Code goes here`.

   (a) Run the script `logit_calculation_A7.py` up to line 150 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_like_sum_opt(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.

1. Use the Nelder-Mead Simplex algorithm algorithm by passing the argument `method = 'nelder-mead'`.
2. Use the Davidon-Fletcher-Powell (DFP) algorithm by passing the argument `method ='powell'`.
3. Use the Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS) algorithm by passing the argument `method = 'BFGS'`.
4. 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 = f'xtol': 1e-8, 'maxiter': 1000, 'disp': True`

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