`Assignment 4 --- QMB 3311 --- Spring 2023 --- Due: February 19, 2023`

Instructions:

Complete this assignment within the space on your *private* GitHub repo in a folder called `Assignment_04`. In this folder, save your answers to 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 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. 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.

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`

   (a) Write a function `matrix_inverse(mat_in)` that replicates teh `numpy` method `linalg.inv()` that calculates the inverse of a 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 $A \cdot A^{-1} = I$, where I is the identity matrix with ones on the diagonal and zeros elsewhere. It can be used to solve the systems of equations $A \cdot x = b$ 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 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 arguement 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 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 occurs. Use this condition to formulate a fourth test case.

(b) 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 teh log-likelihoond across all observation $(y_i, x_i, i = 1, ..., n$. That is, it returns the sum of either the log of the function $\ell(x; \beta_0, \beta_1)$ if $y_i = 1$ or the log of the function $(1 - \ell(x; \beta_0, \beta_1))$ if $y_i = 0$, over all observation $i = 1, ..., n$:

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

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

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

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

(c) The *gradient vector* of a multivariate function is a vector with each element equal to the derivatrive of the function with respect to each parameter. It 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 the previous exercise above.

Using calculus, one can determine that

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

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 arguements in $(y, x; \beta_0, \beta_1)$, in that order, and will output a vector of two elements corresponding the 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`

(d) In a previous assignment, 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 valud 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 arguements 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.