Python Assignment
DUE: DECEMBER 5TH 2023 @23:59

Note: This assignment makes use of the `CodeGrade` tool that allows you to receive instantaneous feedback on your code while you work on this assignment. You will also use this tool to hand in your final solution. In the lecture on *Tuesday, 21st November*, a short demonstration will be provided on how you can use CodeGrade to receive feedback and submit your solution.

Instructions

- Download the file `period2_assignment.zip` from Canvas and extract it. In this zip-file, you can find three files: `assignment.py`, `Q2_data.csv` and `Q3_data.csv`. While you work on the assignment, keep these files in the same folder and don't rename any of them. When you submit on CodeGrade, you should only submit `assignment.py`, don't upload the datasets.
- The questions in this assignment will ask you to complete the code of different functions provided in the file `assignment.py`. **You must work in this file. Do not rename the file.** You can edit this file using an editor of your choice (e.g. Spyder, VS Code etc.).
- To answer the questions in this assignment, add your code in between the answer blocks in each function (see below). **Do not modify any of the lines outside these blocks**, otherwise you risk your code not running and you might not receive points for the questions for which your code crashes.

```
## BEGIN ANSWER
# ...
# TODO: Add your code here
# ...
## END ANSWER
```

- To answer the questions in this assignment, you are only allowed to use basic Python functions, as well as the libraries imported in `assignment.py`. As a hint, some useful functions from these libraries have already been imported (see the first lines of `assignment.py`). In addition, the exercises below might specify some restrictions on which functions/libraries you are allowed to use for each question. If you use libraries or functions not permitted for the exercise, you will not receive any points for the questions for which you used them.
- **CodeGrade instantaneous feedback:** You can receive feedback on whether your code is working correctly by copy and pasting your code into the editor in CodeGrade and clicking the `Run` button. This will run your code through a series of functionality checks which check whether your code computes the right answers for each question. You will see for which questions your code passes the checks and which questions you still need to work on. *You can hand-in your code as many times as you want*, so make sure that your code passes as many of the checks as possible in the final version that you submit.
- **Questions/doubts:** If you have any doubts about the questions or you believe that your code is working correctly, but it is not passing the functionality checks, please send a Canvas message to the TA (*Boaz van Bokhorst*), attaching your current version of `assignment.py`. Make sure you clearly state your group name and *specific (!) question.* General questions on whether your code is correct, how to solve the questions etc. will be ignored.

1 Random Number Generation

Note: The questions below will ask you to generate pseudo-random numbers from different distributions. In your code, you should use the `np.random.rand()` or `np.random.uniform()` functions to generate random numbers from the uniform distribution (i.e. don't code a uniform RNG yourself). However, you may not use any other random number generators (e.g. for different distributions, from other libraries etc.).

The *Cauchy distribution* has support $x \in \mathbb{R}$ and has parameters location $\mu \in \mathbb{R}$ and scale $\sigma > 0$. It has the probability density function (pdf)

$$f_X(x) = \frac{1}{\pi\sigma\left(1 + \left(\frac{x-\mu}{\sigma}\right)^2\right)} \tag{1}$$

The *Laplace distribution* has support $x \in \mathbb{R}$ and has parameters location $\mu \in \mathbb{R}$ and scale $b > 0$. It has the probability density function (pdf)

$$f_X(x) = \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right) \tag{2}$$

(a) Complete the code for the function `pdf_cauchy(x, mu, sigma)` which accepts a scalar input x and returns the scalar density value $f_X(x)$.

(b) Complete the code for the function `pdf_laplace(x, mu, b)` which accepts a scalar input x and returns the scalar density value $f_X(x)$.

(c) Complete the code for the function `rng_cauchy(n, mu, sigma, seed)`. Your code should return a vector of n pseudo-random numbers from the Cauchy(μ, σ) distribution. To generate these numbers, you should use the *Inverse CDF method*.

(d) Complete the code for the function `rng_std_laplace(n, seed)`. Your code should return a vector of n pseudo-random numbers from the Laplace$(0, 1)$ distribution. To generate these numbers, you should use the *Rejection-Sampling method* using a Cauchy$(0, 1)$ proposal density. To draw Cauchy random numbers, you should use your function from (c).

Note: It should be clear from your code what value of M you used, i.e. define `M=...`.

(e) Complete the code for the function `hist_std_laplace()` such that it draws a histogram of 50,000 draws from the Laplace$(0, 1)$ distribution using your function from (d). To generate the plot, use `plt.hist()`. The histogram should show the density of the Laplace random draws using 200 bins. The x limits of the plot should be fixed to $[-7, 7]$. The histogram should be overlaid with a plot of the theoretical Laplace density in the appropriate range using your pdf function from (b).

2 Numerical Optimization

Note: Some of the questions below will ask you to optimize objective functions using numerical optimization methods. In question (e), you should code up the optimization procedure yourself following the steps laid out in the exercise. Only in question (g) are you allowed to use the `minimize()` function from the `scipy` library.

This question concerns the simple linear regression model you discussed in Econometrics I.

$$y_i = \beta_0 + \beta_1 x_i + u_i, \qquad i = 1, 2, \ldots, n \tag{3}$$

Let $\boldsymbol{y}$ be the n-dimensional vector with entries y_i and $\boldsymbol{x}$ be the n-dimensional vector with entries x_i. In addition, let $\boldsymbol{X}$ be the $(n \times 2)$ matrix whose first column contains 1 in every row and whose second column is $\boldsymbol{x}$.

The **Ordinary Least Squares (OLS)** estimator in this model $\hat{\beta}_{OLS}$, which minimizes the sum of squared residuals (SSR) of the model, is the 2-dimensional vector given by

$$\hat{\beta}_{OLS} := \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \left(\boldsymbol{X}^\top \boldsymbol{y}\right) \tag{4}$$

In contrast, the **Least Absolute Deviation (LAD) estimator** in this model $\hat{\beta}_{LAD}$ is defined as the solution to the following optimization problem

$$\hat{\beta}_{LAD} := \underset{\boldsymbol{b} \in \mathbb{R}^2}{\arg\min} \quad SAR(\boldsymbol{b}, \boldsymbol{y}, \boldsymbol{x})/n \tag{5}$$

$$SAR(\boldsymbol{b}, \boldsymbol{y}, \boldsymbol{x}) = \sum_{i=1}^{n} \left| y_i - b_0 - b_1 x_i \right| \tag{6}$$

For the questions below, you will work with the dataset `Q2_data.csv`. Make sure this dataset is placed in the same folder as the file `assignment.py` that you are editing. For testing your functions while you develop them, you can load the data using the function supplied in the template by running `y, x = load_data_q2()`.

(a) Using the formula in eq. (4), complete the code for the function `ols_estimator(y, x)` which should compute and return $\hat{\beta}_{OLS}$.

(b) Complete the function `ols_scatterplot()` such that it plots a scatter plot of the data and overlays it with the fitted regression line given by $\hat{\beta}_{OLS}$ (black solid line).

(c) Complete the function `sar(b,y,x)` such that it computes the sum of absolute residuals given in equation (6).

(d) Complete the function `sar_grad(b,y,x)` such that it computes the gradient of the sum of absolute residuals with respect to $\boldsymbol{b}$ i.e. $\nabla_{\boldsymbol{b}}\, SAR(\boldsymbol{b}, \boldsymbol{y}, \boldsymbol{x})$.

(e) Consider the implementation of a gradient-based optimization algorithm called **Gradient-descent** that is described below. This algorithm is similar to the Newton-Raphson procedure with line search we discussed in the lecture, but does not make use of Hessian information. It also uses a different backtracking procedure than the one we discussed in the lecture.

Gradient-Descent

- Start with an initial guess $\boldsymbol{b^{(0)}}$.
- For $k = 0, 1, \ldots,$ loop until the convergence criterion $|f(\boldsymbol{b^{(k-1)}}) - f(\boldsymbol{b^{(k)}})| < f_{tol}$ holds or the maximum number of iterations is exceeded.
  - Compute the gradient $G(\boldsymbol{b}^{(k)})$ and updating direction $\boldsymbol{h}^{(k)} = -G(\boldsymbol{b}^{(k)})$.
  - Update $\boldsymbol{b}^{(k+1)} = \boldsymbol{b}^{(k)} + \alpha_k \cdot \boldsymbol{h}^{(k)}$ where α_k is determined using the following *backtracking procedure* for $c_1 = 1e-4$ and $\rho = 0.95$.
    (a) Start with $\alpha_k = 1.0$
    (b) Check if the Armijo rule $f(\boldsymbol{b}^{(k)} + \alpha_k \boldsymbol{h}^{(k)}) \leq f(\boldsymbol{b}^{(k)}) + c_1 \alpha_k \cdot \boldsymbol{h^{(k)}} G(\boldsymbol{b}^{(k)})^\top$ holds.
    (c) If yes, use this value of α_k. If not, update $\alpha_k = \rho \cdot \alpha_k$ and go to (b).
- Return $\boldsymbol{b}^{(k+1)}$.

Complete the function `gradient_descent(f, grad, b0, y, x, max_iter, f_tol)` so that it implements this algorithm.

(f) Complete the function `lad_scatterplot()` such that it plots a scatter plot of the data and overlays it with the fitted regression line given by $\hat{\beta}_{OLS}$ (black solid line) and $\hat{\beta}_{LAD}$ (red dashed line). To compute $\hat{\beta}_{LAD}$, you should use the `gradient_descent()` function from (e) with a starting value of $\boldsymbol{b_0} = (0.0, 0.0)^\top$.

(g) Compute $\hat{\beta}_{LAD}$ again for the dataset by completing the code for the function `lad_nelder_mead()`, but this time use the `minimize()` function from `scipy.optimize` to minimize the sum of absolute residuals using the **Nelder-Mead algorithm** with a starting value of $\boldsymbol{b_0} = (0, 0)^\top$.

3 Solving Linear Equations & Eigenvalues

Note: Some of the questions below will ask you to solve linear equation systems or to compute eigenvalues and eigenvectors using numerical methods. For these exercises, you should use the functions `np.linalg.solve()` and `np.linalg.eig()`, instead of coding up the algorithms that were discussed in the lecture yourself.

This exercises concerns the **Ridge estimator** in the multiple linear regression model (without constant) given by

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + u_i, \qquad i = 1, 2, \ldots, n \tag{7}$$

Let $\boldsymbol{y}$ be the n-dimensional vector with entries $y_1, y_2, \ldots, y_n$ and $\boldsymbol{x_j}$ be the n-dimensional vector with entries $x_{1j}, x_{2j}, \ldots, x_{nj}$. Further, let $\boldsymbol{X}$ be the $(n \times k)$ matrix with columns corresponding to $\boldsymbol{x_j}$ for $j = 1, \ldots, k$. As you discussed in Econometrics I, the Ridge estimator is the solution to the following optimization problem for some choice of $\alpha \geq 0$.

$$\hat{\beta}_{ridge} := \underset{\boldsymbol{b} \in \mathbb{R}^k}{\arg\min} \quad Q(\boldsymbol{b}, \boldsymbol{y}, \boldsymbol{X}) \tag{8}$$

$$\begin{aligned} Q(\boldsymbol{b}, \boldsymbol{y}, \boldsymbol{X}) &:= \big(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{b}\big)^\top \big(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{b}\big) \; + \; \alpha \cdot \big(\boldsymbol{b}^\top \boldsymbol{b}\big) \\ &= \boldsymbol{y}^\top \boldsymbol{y} - 2\boldsymbol{b}^T \boldsymbol{X}^\top \boldsymbol{y} \; + \; \alpha \cdot \boldsymbol{b}^T \boldsymbol{b} \end{aligned} \tag{9}$$

(a) Analytically derive an expression for $\nabla_{\boldsymbol{b}}\, Q(\boldsymbol{b}, \boldsymbol{y}, \boldsymbol{X}) = 0$. Then, bring this equation into the form $\boldsymbol{M}\boldsymbol{b} = \boldsymbol{z}$. Complete the code for the functions `ridge_M(X,alpha)` and `ridge_z(y,X)` so that they implement the expressions for $\boldsymbol{M}$ and $\boldsymbol{z}$.

(b) Complete the code for the function `ridge_estimator(y,X,alpha)` so that it computes $\hat{\beta}_{ridge}$ by solving the equation from (a) using `np.linalg.solve()`.

(c) Let $\boldsymbol{w}$ be the k-dimensional vector that collects the eigenvalues of $\boldsymbol{X}^\top \boldsymbol{X}$. Further, let $\boldsymbol{V}$ be the $(k \times k)$ matrix whose columns are the eigenvectors associated with eigenvalues $w_1, w_2, ..., w_k$. Complete the code for the function `ridge_ev_decomp_XtX(X)` so that it returns $\boldsymbol{w}$ and $\boldsymbol{V}$.

(d) Note that using the equation in (a), we can also compute the ridge estimator as $\boldsymbol{b} = \boldsymbol{M}^{-1}\boldsymbol{z}$. In this question, we will directly compute $\boldsymbol{M}^{-1}$ via an eigenvalue decomposition of the matrix $\boldsymbol{X}^\top \boldsymbol{X}$. Complete the code for the function `ridge_Minv(X,alpha)` so that it computes $\boldsymbol{M}^{-1}$ via the following steps.

- Use your function from (c) to compute the vector of eigenvalues $\boldsymbol{w}$ and the matrix of eigenvectors $\boldsymbol{V}$ of the matrix $\boldsymbol{X}^\top \boldsymbol{X}$.
- Compute the modified vector of eigenvalues $\tilde{\boldsymbol{w}}$ where $\tilde{w}_j = 1/(w_j + \alpha)$ for $j = 1, 2, \ldots, k$.
- Compute $\boldsymbol{M}^{-1} = \boldsymbol{V}\tilde{\boldsymbol{W}}\boldsymbol{V}^\top$ where $\tilde{\boldsymbol{W}}$ is a diagonal matrix with $\tilde{\boldsymbol{w}}$ on the diagonal.

(e) Complete the code for the function `ridge_estimator_via_inv(y,X,alpha)` so that it computes the ridge estimator as $\boldsymbol{M}^{-1}\boldsymbol{z}$. You may not compute $\boldsymbol{M}^{-1}$ using `numpy`, but you should use your function from (d). For computing $\boldsymbol{z}$, use your function from (a).