Problem Set 2

ECON 6343: Econometrics III
Prof. Tyler Ransom
University of Oklahoma

Directions: Answer all questions. Each student must turn in their own copy, but you may work in groups. You are encouraged to use any and all Artificial Intelligence resources available to you to complete this problem set. Clearly label all answers. Show all of your code. Turn in jl-file(s), output files and writeup via GitHub. Your writeup may simply consist of comments in jl-file(s). If applicable, put the names of all group members at the top of your writeup or jl-file.

Before starting, you will need to install and the following packages:

```
Optim

HTTP

GLM
```

You will also need to load the following packages:

```
LinearAlgebra

Random

Statistics

DataFrames

CSV

FreqTables
```

On Github there is a file called `PS2_starter.jl` that has the code blocks below already created.

1. **Basic optimization in Julia.** We'll start by finding the value of x that maximizes the function

$$f(x) = -x^4 - 10x^3 - 2x^2 - 3x - 2.$$

   In more formal math terms, our objective is

$$\max_x f(x) = -x^4 - 10x^3 - 2x^2 - 3x - 2.$$

While we could probably solve this by hand, the goal of today's problem set is to introduce you to Julia's nonlinear optimization tools.

We will use Julia's `Optim` package, which is a function *minimizer*. Thus, if we want to find the maximum of $f(x)$, we need to minimize $-f(x)$.

The `Optim` package provides a function called `optimize()`. This function requires three inputs: the objective function, a starting value, and an optimization algorithm. We will not get too deep into optimization algorithms in this course, but for now just use `LBFGS()`.

Below is some code that shows how we can solve the objective function written above. You should copy and paste this code into your Julia script for this problem set. You should also copy, paste and run it in the REPL.

```
using Optim
f(x) = -x[1]^4-10x[1]^3-2x[1]^2-3x[1]-2
negf(x) = x[1]^4+10x[1]^3+2x[1]^2+3x[1]+2
startval = rand(1)  # random number as starting value
result = optimize(negf, startval, LBFGS())
```

The output printed in the REPL will be something like (but not exactly, since starting values may differ)

```
 * Status: success (objective increased between iterations)

 * Candidate solution
    Final objective value:     -9.643134e+02

 * Found with
    Algorithm:     BFGS

 * Convergence measures
    |x - x'|               = 1.81e-11 > 0.0e+00
    |x - x'|/|x'|          = 2.45e-12 > 0.0e+00
    |f(x) - f(x')|         = 3.41e-13 > 0.0e+00
    |f(x) - f(x')|/|f(x')| = 3.54e-16 > 0.0e+00
    |g(x)|                 = 8.91e-09 < 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    6
    f(x) calls:    27
    Nablaf(x) calls:   27
```

And we can see the maximum is - $\left(\text{-}9.643 \times 10^2\right) = 964.3$. To get the optimizer, we have to issue a call at the REPL:

```
julia> println(result.minimizer)
[-7.378243405529116]
```

which shows that the argmax is at ≈-7.38.

2. Now that we're familiar with how `Optim`'s `optimize()` function works, lets try it on some real-world data.

   Specifically, let's use `Optim` to compute OLS estimates of a simple linear regression using actual data. The process for passing data to `Optim` can be tricky, so it will be helpful to go through this example.

   First, let's import and set up the data. Note that you will need to put the URL all on one line when executing this code in Julia.

```
using DataFrames
using CSV
using HTTP
url = "https://raw.githubusercontent.com/OU-PhD-Econometrics/fall-2024/
master/ProblemSets/PS1-julia-intro/nlsw88.csv"
df = CSV.read(HTTP.get(url).body, DataFrame)
X = [ones(size(df,1),1) df.age df.race.==1 df.collgrad.==1]
y = df.married.==1
```

   Now let's use `Optim` to solve our objective function:

$$\min_{\beta} \sum_i (y_i - X_i\beta)^2$$

   This estimates the linear probability model

$$married_i = \beta_0 + \beta_1 age_i + \beta_2 1[race_i = 1] + \beta_3 1[collgrad_i = 1] + u_i$$

   A tricky thing with using `Optim` is that it requires something called a *closure* to be able to pass data into the function.

```
function ols(beta, X, y)
    ssr = (y.-X*beta)'*(y.-X*beta)
    return ssr
end

beta_hat_ols = optimize(b -> ols(b, X, y), rand(size(X,2)), LBFGS(),
                        Optim.Options(g_tol=1e-6, iterations=100_000,
                        show_trace=true))
println(beta_hat_ols.minimizer)
```

   We can check that this worked in a few different ways:

```
using GLM
bols = inv(X'*X)*X'*y
df.white = df.race.==1
bols_lm = lm(@formula(married ~ age + white + collgrad), df)
```

Indeed, all three ways give the same estimates.

3. Use `Optim` to estimate the logit likelihood. Some things to keep in mind:
   - To maximize the likelihood, you will need to pass `Optim` the *negative* of the likelihood function, since `Optim` is a minimizer
   - The likelihood function is included in the Lecture 4 slides
4. Use the `glm()` function from the `GLM` package to check your answer. (Example code for how to do this is in the Lecture 3 slides.)
5. Use `Optim` to estimate a multinomial logit model where the dependent variable is `occupation` and the covariates are the same as above.

   Before doing this, clean the data to remove rows where `occupation` is missing. We also need to aggregate some of the occupation categories or else we won't be able to estimate our multinomial logit model:

```
using FreqTables
freqtable(df, :occupation) # note small number of obs in some occupations
df = dropmissing(df, :occupation)
df[df.occupation.==8 ,:occupation] .= 7
df[df.occupation.==9 ,:occupation] .= 7
df[df.occupation.==10,:occupation] .= 7
df[df.occupation.==11,:occupation] .= 7
df[df.occupation.==12,:occupation] .= 7
df[df.occupation.==13,:occupation] .= 7
freqtable(df, :occupation) # problem solved
```

   Since we changed the number of rows of `df`, we also need to re-define our `X` and `y` objects:

```
X = [ones(size(df,1),1) df.age df.race.==1 df.collgrad.==1]
y = df.occupation
```

Hints:

- With 7 choice alternatives, you will have $K \cdot 6$ coefficients, where K is the number of covariates in X. It may help to transform the parameter vector into a $K \times 6$ matrix (to more easily reference the α_j's for each j)
- You should reset the tolerance of the gradient (`g_tol`) to be 10^-5. This will help the estimation converge more quickly, without losing too much precision

- You may need to try different sets of starting values. Some candidates to consider are:
  - a vector of 0s
  - a vector of $U[0,1]$ random numbers
  - a vector of $U[-1,1]$ random numbers
  - the estimated values from Stata or R (see below)

Notes:

- If you have access to Stata, you can check your answers with the following code:

```
webuse nlsw88
drop if mi(occupation)
recode occupation (8 9 10 11 12 13 = 7)
gen white = race==1
mlogit occupation age white collgrad, base(7)
```

- In general it is a good strategy to run your model(s) through a more user-friendly interface like Stata or R before trying to implement them in Julia. But you might ask, "Why don't we just use Stata or R, then?" The reason is because the models we will get to later in the course are much more difficult to implement in those languages, because they can't just be taken off the shelf.

6. Wrap all of your code above into a function and then call that function at the very bottom of your script. Make sure you add `println()` statements after obtaining each set of estimates so that you can read them.

7. Have an AI write unit tests for each of the functions you've created (or components of each) and run them to verify that they work as expected. Best practice is to provide unit tests in a separate script that first reads in the source code before running the tests.