Problem Set 1

Quantitative Economics, Fall 2023

October 22, 2023

This problem set consists of three problems. You have two weeks to solve them and submit your solutions (Nov. 6th 11:59 PM). You can work in teams of up to three students.

Problem 1: Julia basics and Random.jl

Recall the Lindeberg-Levy Central Limit Theorem:[1]

$$\frac{\bar{X}_n - \mu}{\sigma\sqrt{n}} \xrightarrow{d} N(0,1)$$

for $\{X_1, X_2, \ldots\}$ i.i.d. with $E[X_i] = \mu$ and $Var[X_i] = \sigma^2 < \infty$. Here $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$.

Your task is to illustrate it for $X_i \sim \text{Pois}(\lambda)$.[2] You will have to plot four histograms of 1000 realizations of $\bar{X}_n$, for $n = 5, 25, 100, 1000$, (in blue) and superimpose the density of $N(0,1)$ on each of them (in red).

We will set $\lambda = 1$ for simplicity. Your task is to write code that will generate the requested plots. There is more than one solution! For example: you can calculate averages by summation or by using a function that already exists in Julia.[3]

You will have to use the *Random.jl* package to generate the random numbers and *Distributions.jl* to specify the distribution.[4] *StatsPlots.jl* will be useful for plotting the density function.

Problem 2: Iterative solver for nonlinear equations

In this problem you will have to code up a simple function that we can use to solve nonlinear equations in one unknown by using an iterative method. Consider the following problem:

Let $f : R \to R$. We want to find x such that

$$0 = f(x).$$

Our task is thus to find a root (because there are possibly many) of $f(x)$. Notice that it is the same as

$$x = f(x) + x$$

or, if we define $g(x) := f(x) + x$

$$x = g(x).$$

[1] See here for more details.

[2] Recall that for $X \sim \text{Pois}(\lambda)$, $E[X] = \lambda$ and $Var[X] = \lambda$.

[3] I do not ask you to write first a function that works for arbitrary argument values (including even the density function). I however encourage you to think how to make your code well-organized, easy to read and modular. Use this simple exercise as an opportunity to think about these issues.

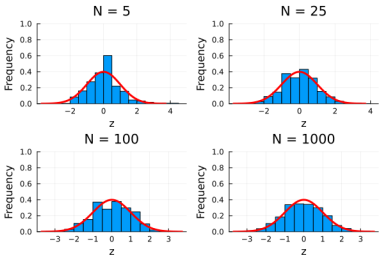

Figure 1: You should produce something like this.

[4] See the *ps1_hint.jl* file for a hint.

We look for a fixed point of $g(x)$: a value of x such that $g(x)$ is equal to x. It may or may not exist. Sometimes we can prove a fixed point exists.[5] Sometimes we can even show it is unique and we will converge to it from any guess.[6] Suppose that we start with some $x = x_0$ and we calculate $x_1 = g(x_0)$. We can then check if x_1 is close to x_0. If it is, i.e. $\frac{|x_1 - x_0|}{1+|x_0|} < \epsilon$ for some small $\epsilon > 0$ we (approximately) found a fixed point. If not, then we calculate $x_2 = g(x_1)$ and check again. We proceed in this fashion using the formula $x_{n+1} = g(x_n)$. You task is to write a *function* that takes three arguments:

[5] Brouwer's, Kakutani's and Tarski's fixed point theorems are among those, which are often seen in economics

[6] See Contraction Mapping Theorem by Stefan Banach. We will talk about it in detail.

1. a one-dimensional *function* f,
2. a starting guess x_0,
3. an extra parameter α to be explained shortly below,
4. a tolerance parameter ϵ,
5. a maximum number of iterations *maxiter*.

This function is supposed to return:

1. an integer $flag$ equal to 0 if the solution has been found before the maximum number of iterations has been reached
2. the point that is the solution (or NaN if it has not been found),
3. the value of that solution (or NaN if it has not been found),
4. an absolute value of a difference between the solution x and $g(x)$,
5. a vector of all $\{x_0.x_1, x_2, \cdots\}$
6. a vector of all residuals $\{|x_1 - x_0|, |x_2 - x_1|, \ldots\}$.

We will actually work with a modified version of the algorithm:

$$x_{n+1} = (1 - \alpha)\, g(x_n) + \alpha x_n.$$

The parameter α is responsible for dampening: if it's close to 1, then we will update our guesses only slowly.

Once you write the function, test it by find the root of $f(x) = (x+1)^{\frac{1}{3}} - x$ starting from $x_0 = 1$ and using $\alpha = 0$. Does it work for different values of α and/or different starting points? Convince yourself that finding the root of $f(x)$ is the same as finding the root of $h(x) = x^3 - x - 1$. Can you find α such that the algorithm converges to the root starting from $x_0 = 1$?[7]

[7] What if you start from a point slightly different from the root of $f(x)$?

Problem 3: Stochastic optimization

In this problem you will have to code up a simple function that we can use to find the global **maximum** of a nonnegative univariate function. The idea is that we will explore a function by evaluating it at various randomly chosen points.

The algorithm proceeds as follows:[8]

1. Given a current point x_n, draw a random number ε from a standard normal distribution, $\varepsilon \sim N(0,1)$.
2. Calculate $x^* = x_n + \lambda\varepsilon$, where λ is a step size parameter.
3. Calculate $\alpha = \min\left\{1, \frac{f(x^*)}{f(x_n)}\right\}$.
4. Set $x_{n+1} = \begin{cases} x^* & \text{with probability } \alpha \\ x_n & \text{with probability } 1-\alpha \end{cases}$.
5. Keep stored x that yields the highest value of $f(\cdot)$.
6. Terminate if the point that yields the highest value has not changed for N_δ iterations or if the maximum number of iterations has been reached. We will say that the point has not been changed if the difference (appropriaterly scaled) between the value at the candidate optimum and the best value in the past N_δ iterations is not greater than δ.[9]

[8] This method can be easily generalized to multidimensional case (for example cycle over coordinates) or to allow for changing λ over time. We can allow for large steps initially to explore the space and then decrease the step size as we think we are getting closer to the optimum.

[9] We can also add an extra criterion that check is the point that yields the highest value is not too far from the second-best point over the past N_γ iterations.

The idea is that we test a point x^* and if it is better than the current point x_k, we move to x^* for sure. If it is worse, we can still move to it but with probability α. The probability of moving is lower if x^* is much worse. The parameter λ controls the size of the steps we take.

You task is to write a *function* that implements the above algorithm. It should take the following arguments:

1. a one-dimensional nonnegative *function* f,
2. a starting guess x_0,
3. λ, the step-size parameter,
4. a maximum number of iterations *maxiter*,
5. δ, the tolerance parameter,
6. N_δ, explained above.

This function is supposed to return:

1. an integer $flag$ equal to 0 if the solution has been found before the maximum number of iterations has been reached (this means that the condition on the difference between the value at the candidate optimum and the best value in the past N_δ iterations has not been violated),
2. the point that is the solution (or NaN if it has not been found),
3. the value of that solution (or NaN if it has not been found),
4. a vector of all *accepted* points $\{x_0.x_1, x_2, \cdots\}$
5. a vector of all *accepted* values $\{f(x_0).f(x_1), f(x_2), \cdots\}$
6. the average rate of acceptance, i.e. the average of α over all iterations.