Problem Set 1

Quantitative Economics, Fall 2024

January 10, 2025

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

$$v(X, D) = \max_F p \cdot \min\{D, X\} - c \cdot F - K \cdot \mathbb{I}_{F>0} + \beta E_{X,D} v(\min\{\max\{X - D, 0\} + F, \bar{X}\}, D')$$

Problem 1: A simple differential equation

We have

$$\frac{dy}{dx} = y(x),$$

with the initial condition $y(0) = 1$. Our task to find (x) that satisfies the above. It is straightforward to show that it must be $y(x) = \exp(x)$.[1] Your task is to write a function that takes the step size h as an argument and returns the value of y at $x = 1$.

[1] Just differentiate $\exp(x)$ to verify it is true.

Problem 1: Solow model

Recall the law of motion for capital in the Solow model:

$$K_{t+1} = sY_t + (1 - \delta)K_t,$$

where K_t is the capital stock, Y_t is the output, $s \in (0, 1)$ is the saving rate, and δ is the depreciation rate. The output is given by the Cobb-Douglas production function:

$$Y_t = K_t^\alpha (A_t N_t)^{1-\alpha},$$

where A_t is the total factor productivity, N_t is the labor force, and $\alpha \in (0, 1)$ is the output elasticity with respect to capital. In addition g is the growth rate of TFP and n is the growth rate of the labor force.[2]

Recall that capital per unit of effective labor, $k_t \equiv \frac{K_t}{A_t N_t}$ follows[3]

$$ngk_{t+1} = sk_t^\alpha + (1 - \delta)k_t.$$

[2] This means that $A_{t+1} = gA_t$ and $N_{t+1} = nN_t$.

[3] Use $K_{t+t} = sK_t^\alpha (A_t N_t)^{1-\alpha} + (1 - \delta)K_t$. Divide it by $A_t N_t$ and use the definition of k_t together with $A_{t+1} = gA_t$ and $N_{t+1} = nN_t$ to get the formula.

The balance growth path level of capital per unit of effective labor is given by

$$k = \left(\frac{s}{\delta + ng - 1}\right)^{1/(1-\alpha)}$$

Your task is to write a script that illustrates the workings of the Solow model: it is a collection of simple exercises, that one could easily solve using a spreadsheet. Try to recycle functions that you design for one exercise in the next ones: in particular, try to write a function that calculate the capital per unit of effective labor in any period t given the parameters of the model and the initial capital per unit of effective labor. Resist temptation to hard-code parameter values.

1. We will study how the speed of convergence to the balanced growth path depends on the parameters of the model. Suppose that the initial capital per effective labor is $k_0 = \rho k$, with $\rho \in (0, 1)$. After how many periods the gap between k_t and k (relative to k) will be smaller than half of the initial gap? Your code should take the four parameters of the model: α, δ, g, n and ρ as inputs and return the smallest integer number of periods that satisfies the condition above as an output. What is this number for $\alpha = 0.3, \delta = 0.1, g = 1.01, n = 1.005$, and $\rho = 0.75$? Does it increase or decrease if we increase α, δ, g, n by 1% (each separately)?

2. Assume there are two economies that are identical except for ρ. We have $\rho_1 < \rho_2$, so the second economy starts closer to the balanced growth path. Write a function that takes the four parameters of the model and ρ_1 and ρ_2 as inputs and returns the number of periods after which the ratio of capital per effective labor in the first economy to the second economy will be smaller than half of the initial gap. What is this number for $\alpha = 0.3, \delta = 0.1, g = 1.01, n = 1.005$, and $\rho_1 = 0.25, \rho_2 = 0.50$? Does it increase or decrease if we increase α, δ, g, n by 1% (each separately)?

3. Write a function that creates the following plot. On the horizontal axis we will have the number of periods: $t = 0, 1, \ldots$. We will plot the natural logarithm of output in the economy, $\ln(Y_t)$[4] (solid line). Normalize $A_0, N_0 = 1$. In addition, we will plot the natural logarithm of output on the balanced growth path (dotted line).[5] You function should take the four parameters of the model, ρ, and the number of periods T as inputs and return the plot. Make sure that the plot is well-labeled and has a legend.

[4] We have $Y_t = k_t^\alpha A_t N_t$

[5] It is given by $Y_t = k^\alpha A_t N_t$.

Problem 2: Julia basics and Random.jl

Recall the Lindeberg-Levy Central Limit Theorem:[6]

[6] See here for more details.

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

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 $X_i \sim \text{Pois}(\lambda)$.[7] 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).

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

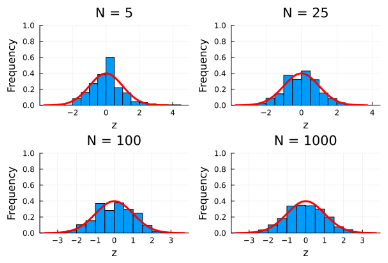

Figure 1: You should produce something like this.

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.[8]

You will have to use the *Random.jl* package to generate the random numbers and *Distributions.jl* to specify the distribution.[9] *StatsPlots.jl* will be useful for plotting the density function.

[8] 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.

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

Problem 3: Fun with Vandermonde matrices

We call a square matrix $\mathbf{V}_n$ a Vandermonde matrix if its columns are powers of a vector $\mathbf{x}$:

$$\mathbf{V}_n = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix}.$$

Observe that $\mathbf{V}_n$ is of size $(n+1) \times (n+1)$. We will encounter these matrices later, when we talk about function interpolation. Suppose that we take a vector of equidistant points, i.e. $x_i = i/n$ for $i = 0, 1, \ldots, n$. You task is to write a function that takes an integer n as an argument and returns the condition number of the corresponding Vandermonde matrix. Please do it without using *SpecialMatrices.jl*. After that, you should plot the natural logarithm of the condition number of the Vandermonde matrix as a function of n for $n = 2, 3, \ldots, 20$.

Problem 4: Some linear algebra

Let

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 0 & \alpha-\beta & \beta \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

The system $\mathbf{Ax} = \mathbf{b}$ is relatively easy to solve by hand. Your first task is to write a function that takes α and β as arguments and returns the exact solution. You will have to do algebra by hand to find it – start from the last equation. Next, write a function that takes α, β as arguments and return the exact solution, the solution obtained by using the backslash operator, the relative residual, and the condition number. Use your function to create the table that shows x_1, the condition number, and the relative residual for $\alpha = 0.1$ and $\beta = 1, 10, 100, \ldots, 10^{12}$.

Problem 5: 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 $g(x) := f(x) + x$

$$x = g(x).$$

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.[10] Sometimes we can even show it is unique and we will converge to it from any guess.[11] 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:

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

[11] 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 $x_0 = 1$?[12]

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

Problem 6: 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:[13]

[13] 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.

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 (appropriately scaled) between the value at the candidate optimum and the best value in the past N_δ iterations is not greater than δ.[14]

[14] 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.