Warm-Up: Solution

Econ 350: The University of Chicago

This Draft: December 12, 2013

Abstract

The objective of this exercise is for you to become familiar with the basic requirements to proceed with the rest of the empirical exercise. This lets you make sure that your software is installed and try optimization exercises . Also, it teaches you how to do numerical integration, which is useful in the rest of the empirical project. Be sure to get this exercise correct so that you can be confident to proceed with the rest of the project.

1 How to Calculate a Numerical Integral

This exercise is based on Langtangen (2011). This is a useful source for the rest of the course. Try to solve the exercises first and then look at the book.

1.1 Interpolation

The information that a real-valued continuous function contains is actually infinite (in the not-countable sense). The computer, however, works in $\aleph_0$. This is not trivial, the computer can go ahead and count forever. How does a computer plots functions?

Exercise 1.1 *(Plot a basic continuous function through a grid) Create a grid of* $1,000$ *points to plot one cycle in the unitary circle of the sine function. Hint: use Google.*
Answer: see code "integration.py".

Roughly, with $1,000$ points, the computer: (i) evaluates the sine function, (ii) puts together coordinates, (iii) scatters them, (iv) joins them through lines. This method has a name: *linear interpolation*. This is how linear interpolation happens. A given point x lies in the interval $[x_k, x_{k+1}]$, for an arbitrary k. In that interval, the computer defines a function that passes through the points (x_k, s_k) and (x_{k+1}, s_{k+1}):

$$S_k(x) = s_k + \frac{s_{k+1} - s_k}{x_{k+1} = x_k}(x - x_k). \tag{1}$$

$S_k(x)$ is identical to $sin(x)$ at x_k and x_{k+1}; between these points, $S_k(x)$ is linear.

1.2 Trapezoidal Integration

Some functions have *primitive* form integrals and some do not. How to integrate the sine function on the interval $(0, \pi)$ without a primitive form? For example,

$$\int_0^\pi \sin(x)dx. \tag{2}$$

From basic calculus it is possible to partition this integral in an arbitrary sum of integrals:

$$\int\limits_{0}^{\pi} \sin(x)dx = \sum_{k=0}^{n-1} \int\limits_{x_k}^{x_{k+1}} \sin(x)dx \tag{3}$$

where n is the number of partitions and $x_n \equiv \pi$. This is useful because it is more precise to approximate it "by pieces". Precisely, the task is to compute the following:

$$\int\limits_{x_k}^{x_{k+1}} \sin(x)dx \tag{4}$$

…but $sin(x) \approx S_k(x)$. Then,

$$\int\limits_{x_k}^{x_{k+1}} \sin(x)dx = \int\limits_{x_k}^{x_{k+1}} S_k(x)dx. \tag{5}$$

Now, note that: (i) a definite integral is the surface under the curve in certain interval; (ii) the interpolation function is linear. Thus, the approximation of the integral in the interval of interest implies the calculation of the area of a trapezoid. In particular:

$$\int\limits_{x_k}^{x_{k+1}} \sin(x)dx \approx \frac{1}{2}(s_{k+1} + s_k)(x_{k+1} - x_k). \tag{6}$$

It is almost always the case that the spacing between the points in the grid in which functions are worked out is uniform. Thus, $(x_{k+1} - x_k) = h$ for some elected h and for $k = 0, \ldots, n-1$. The integral becomes a sum:

$$\begin{aligned} \int\limits_{0}^{\pi} \sin(x)dx &\approx \sum_{k=0}^{n-1}(s_{k+1} + s_k) \\ &= \frac{h}{2}\left[s_0 + 2\sum_{k=1}^{n-1} s_k + s_n\right] \end{aligned} \tag{7}$$

where you can shows that the second equality holds by the principle of mathematical induction.

Clue 1.2 *Do you remember the name of the functions for which the sum on (8) converges to a finite value?*
Answer: the name of these functions is Riemann Integrable.

1.2.1 Generalization

It is possible to generalize this reasoning as follows. Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous function in the interval $[a, b]$. The "grid version" of f is given by the sets of lines that connect the sequence of coordinates $(x_i, y_i)_{i=0}^{n}$ where $x_i = a + ih$ and $y_i f(x_i)$ for $i = 0, \ldots, n$, $n \geq 1$, $h = (b-a)/n$. Then,

Definition 1.3 *(Trapezoidal Integral)*

$$\int\limits_{a}^{b} f(x)dx \approx \frac{h}{2}\left[y_0 + 2\sum_{k=1}^{n-1} y_k + y_n\right]. \tag{8}$$

Exercise 1.4 *(Trapezoidal Integration) Write down a code that has a general function for trapezoidal integration.*
Answer: see code "integration.py".

Exercise 1.5 *(Trapezoidal Integration of the Sine Function) Integrate (2) with* $n = 10, 50, 100, 500$ *using your code of Exercise 1.4. How do these values differ from the values that you can get in your calculator. Hint: switch to radians.*
Answer: see code "integration.py". Note that the desired integral is identical to 2 *and that the method is less imprecise as* n *increases.*

1.3 Monte-Carlo Integration

Take the same framework as in Section 1.2.1. The first mean value theorem for integrals establishes that the integral of a function over the interval $[a, b]$ is the mean value of f over the interval $[a, b]$ multiplied by the length of the interval, $b - a$. Then, a simulated integral is calculated as follows: (i) take n draws from a uniform distribution over $[a, b]$; (ii) evaluate f at each of them; (iii) take an average; (iv) multiply the average by the length of the interval. Put differently, if x_i is a typical random draw,

$$\int_a^b f(x)dx \approx (b-a)\frac{1}{n}\sum_{i=1}^{n} f(x_i). \tag{9}$$

This approach to numerical integration is called Monte Carlo integration.

Exercise 1.6 *(Monte Carlo Integration) Write down a code that has a general function for Monte Carlo Integration.*
Answer: see code "integration.py".

Exercise 1.7 *(Monte Carlo Integration of the Sine Function) Integrate (2) with* $n = 10, 50, 100, 500$ *using your code of Exercise 1.6. Use the random package in Python and set the seed to zero. How do these values differ from the values that you can get in your calculator and in Exercise 1.4. Why?*
Answer: see code "integration.py". For small values of n the trapezoidal integration is more precise. This is because Monte Carlo integration may have a couple of draws that give extreme values of the function to be integrated. As n grows, Monte Carlo integration is more precise. This is specially important because for some functions linear interpolation is inaccurate.

2 How to Maximize a Likelihood

Exercise 2.1 *(Basic Optimization) The Rosenbrock function helps to illustrate how to maximize a function that involves sums. This is useful in this context because log-likelihoods are sums. Write down a code that minimizes the Rosenbrock function with the Broyden-Fletcher-Goldfarb-Shanno algorithm in Python. Examine the function and provide an initial condition (the length of this vector provides the dimension of the elements in the domain of the function. Use* 10*). Hint: use Google.*
Answer: see the code "likelihood.py".

Consider a sample of N i.i.d. observations:

$$x_i \sim \mathcal{N}(0, \theta) \tag{10}$$

for $i = 1, \ldots, N$. θ is the variance and you want to estimate it.

Exercise 2.2 *(Basics) (i) Write down the individual likelihood, the sample likelihood, and the sample log-likelihood; (ii) write down the sample score; (iii) calculate the Maximum Likelihood Estimator (MLE) of θ as a function of the data. Answer:*

(i) The individual likelihood is: $L_i(\theta|x_i) = \frac{1}{\sqrt{2\pi\theta}} \exp^{\left(-\frac{(x_i-0)^2}{2\theta}\right)}$ *. The sample likelihood is:* $L(\theta|x_1,\ldots,x_n) = \prod_{i=1}^{n} L_i(\theta|x_i)$ *. The sample log-likelihood is* $l(\theta|x_1,\ldots,x_n) = -\frac{1}{2}\log(2\pi) - \frac{1}{2}\log\theta - \frac{1}{n}\sum_{i=1}^{n}\frac{(x_i-0)^2}{2\theta}$*.*

(ii) The sample score is $s(\theta|y) = \frac{\partial l(\theta|x_1,\ldots,x_n)}{\partial\theta} = -\frac{1}{2\theta} + \frac{1}{n}\sum_{i=1}^{n}\frac{(x_i-0)^2}{2\theta^2}$*.*

(iii) To calculate the MLE set the score to zero and solve for θ: $\hat{\theta}^{MLE} = \frac{1}{n}\sum_{i=1}^{n}(x_i-0)^2$*.*

Exercise 2.3 *(Estimation I) In Exercise 2.2 you find that the MLE estimator for θ has a closed and easy functional form. Actually, you do not need numeric maximization to obtain it. Simulate your own data with $\theta = 1$ and then recover the parameter by ML. Use the random package in Python and set the seed to zero. Set the Do not use a maximization routine. Use sums and multiplications. Answer: see the code "likelihood.py".*

Exercise 2.4 *(Estimation II) Use the data that you simulated to estimate θ by ML. Use the Broyden-Fletcher-Goldfarb-Shanno algorithm in Python. Hint: recall that this algorithm minimizes.*

Exercise 2.5 *Compare the estimate in Exercise 2.3 to the estimate in Exercise 2.4. Do they differ? Why? Which one is better?*
Answer: although the optimization routine is powerful, the second estimate is an approximation of the first. The more complex the likelihood function, the bigger the difference between these two estimates. Unfortunately, the more complex the likelihood function, the more difficult is to obtain close form MLE estimators.

References

Langtangen, H. P. (2011). *A primer on scientific programming with Python*, Volume 6. Springer.