**Integration via software**

Let the computer do the integration for us!

Example: Transaction time (Transaction.R)

The number of seconds between transactions (e.g., purchases) on a website can be represented by a PDF. Let X be a random variables representing the seconds. Suppose the PDF for X is





Find the probability that the time between a set of randomly selected consecutive transactions will be longer than 30 seconds.

= e-1 ≈ 0.367879

Find the probability the time between a set of randomly selected consecutive transactions will be between 25 and 100 seconds.

 = e-25/30 – e-100/30

 ≈ 0.3989

These probabilities can be found in R using numerical integration via adaptive quadrature. You are only responsible for implementation rather than the details of how it works. Essentially, many different rectangles are formed to approximate the area under the plotted curve (rectangles drawn from 0 to f(x) relative to the y-axis). The sum of the area for these rectangles is the approximate area under the curve.

The PDF needs to be written in terms of a R function where the first argument corresponds to the random variable.

> # Numerical integration

> pdf <- function(x) {

 1/30\*exp(-x/30)

 }

> # Examples

> pdf(x = 0)

[1] 0.03333333

> 1/30\*exp(-0/30)

[1] 0.03333333

> pdf(x = 10)

[1] 0.02388438

> 1/30\*exp(-10/30)

[1] 0.02388438

The integrate() function performs the numerical integration.

> # P(X > 30)

> integrate(f = pdf, lower = 30, upper = Inf)

0.3678796 with absolute error < 8.2e-05

> # P(25 < X < 100)

> integrate(f = pdf, lower = 25, upper = 100)

0.3989242 with absolute error < 4.4e-15

If the error is too large, one can change the rel.tol argument for a more precise answer.

Find the time between transactions such that 95% are less than this value.

 ⇒ c = 30ln(20) ≈ 89.87

We need to use integrate() again but now with the uniroot() function.

> # Calculations to help understand code in find.root()

> save.res <- integrate(f = pdf, lower = 0, upper =

 89.87)

> names(save.res)

[1] "value" "abs.error" "subdivisions"

 "message" "call"

> save.res$value

[1] 0.9499967

> # Extract only the value from integration

> integrate(f = pdf, lower = 0, upper = 89.87)$value

[1] 0.9499967

> # Shows value is 0

We want to solve for c in . Equivalently, solve for c in

.

> 0.95 - integrate(f = pdf, lower = 0, upper =

 89.87)$value

[1] 3.280452e-06

> # Function want to find root for (the "c" where

 expression is 0)

> find.root <- function(c.val) {

 0.95 - integrate(f = pdf, lower = 0, upper =

 c.val)$value

 }

> find.root(c.val = 89.87)

[1] 3.280452e-06

> # Find c value

> uniroot(f = find.root, interval = c(0,300))

$root

[1] 89.87197

$f.root

[1] 4.624123e-11

$iter

[1] 9

$init.it

[1] NA

$estim.prec

[1] 6.103516e-05

Notes:

* Why did I use c.val as the argument in find.root() rather than c? Because c() is a function to create a vector. While it still would work c as the argument, it is good coding practice not to common function names as arguments or object names.
* The uniroot() function requires the use of a function for which the expression of interest is returned. The function, find.root() in this case, needs to have the quantity of interest as its first argument.

R works nice to find the numerical quantities of interest when working with integrals. It does not work well with symbolic forms of integration. For this purpose, there are a number of other tools to choose from.

Example: Transaction time (PDF.ipynb, PDF.sagews)

SageMath is a free open-source software package that performs symbolic mathematical calculations. Important aspects of it:

1. Most people refer to it as Sage, so I will as well.
2. It is meant for Linux operating systems. However, Windows users can install it on their computer too by using software that simulates the Linux environment (e.g., VirtualBox). Newer versions of Sage come with this automatically to help Windows users. Sage can be downloaded from <https://www.sagemath.org>.
3. Cloud computing implementations are available so that users do not need to install on their own computer.

I will focus here on the cloud computing implementation available for free from CoCalc at <https://cocalc.com>. This company provides a web-browser way to use Sage via a Jupyter notebook.

Eventually, you will get to a Jupyter notebook interface similar to the one below.



What is a Jupyter notebook?

Project Jupyter (https://jupyter.org) provides free, open-source software for individuals to write and run computer programs. The name “Jupyter” comes from the three main languages used with it - Julia, Python, and R.

Most individuals will use the Jupyter Notebook program of the project. The actual coding environment is somewhat different than what is experienced in the most often with R. Jupyter Notebook uses a notebook interface that is similar to what is found in mathematical software, like Maple and Mathematica. In this setting, code/output is presented in the following form:

Code (In)

Output (Out)

Code

Output

Code

Output

...

all within the program editor. Thus, it is like using the R Console window directly, but one can easily re-run previous parts of the code by simply moving the cursor back to that portion of the code.

While it is nice to see the output right after the code, problems can be encountered! In particular, code from earlier portions of the program could be changed and re-run, but later portions of the output would not change.

Below is how I used Sage for some of our examples.











Notes:

1. Each In[], Out[] portion is a cell
2. The numbers within brackets correspond to the order the cells are run.
3. To run a cell, select  from the toolbar. To run more than one cell, select *Cell >* from the main menu bar
4. The n(prob,digits = 4) and prob.n(digits = 4) code produces the same result. The n() function is known as a method function. There are other method functions that can be put at the end of expression as well.
5. The work performed is automatically saved as file. If you obtain a free account, these are saved on a server for you. The file type is .ipynb (Jupyter was originally part of iPython). These are simply plain text files, but they are not as easy to examine in a text editor as an R program.
6. To download the .ipynb, you need to be in the main folder directory. Select  when in a Jupyter notebook. Checkmark the particular file and select Download.
7. Jupyter notebooks can be uploaded as well by selecting Upload in the main folder directory.
8. The code/output shown in my notes here were obtained by screen captures using the Snipping Tool and the Snip & Sketch apps in Windows.
9. A paid version exists that CoCalc touts as being faster.

Example: Insurance example (insurance.ipynb)

Suppose probabilities for the number of injury claims per month for a company can be modeled by a random variable X with

f(x) =  for x = 0, 1, 2, … .

as the PDF.



Sage also allows one to run its code outside of a Jupyter notebook. For example, Sage has its own worksheet format (.sagews). One nice aspect of Sage’s worksheet is that it can be a little easier to find help. For example, one can type an item, like f(x).<tab> (<tab> means to press the Tab key), to see a list of method functions that can be used with the quantity.

Other software packages for symbolic integration

1. Maple – Inexpensive for students, but expensive for everyone else. Examples from the classic worksheet version of Maple



Rather than screen captures, one can use regular copy/paste methods:

> P(X>30);



> evalf(P(X>30));



> evalf(P(X>30),4);



> int(f,x=30..infinity);



> int(f,x=25..100);



> evalf(int(f,x=25..100));



> evalf(int(f,x=25..100),4);



> solve(0.95=int(f,x=0..c));



> int(f,x=0..a);



> F:=int(f,x=0..a);



> diff(F,a);



1. Mathematica – Same type of pricing structure. There is a nice free web interface for integration at <https://www.wolframalpha.com/calculators/integral-calculator>.

