Integrating with Python
Ethan Bolker
March 2, 2015

1 Riemann sums

These are approximate lecture notes from the fifth class of the semester (February 26, 2015). I took advantage of the fact that I had not prepared the material to model the process of building a simple program from scratch – something I'd talked about but never done in front of the students.

I began by asking how they would compute

$$\int_0^1 \sqrt{1-x^2}dx \quad .$$

I got several blank looks, one person suggesting a trig substitution. When I drew the picture of a quarter of a circle they all agreed that the answer should be $\pi/4$.

Then I asked about

$$\int_0^1 \exp(-x^2)\log(1+x)\sin(x^3)dx \quad . \tag{1}$$

Some said look at a table of integrals. When I told them there was a theorem showing that function could not be integrated (using the kind of functions they studied in calculus) they were stuck.

At that point I hinted at the definition of the definite integral and they all remembered something about summing the areas of rectangles. The standard blackboard picture of Riemann sums did the trick.

After a brief but important digression on the Fundamental Theorem of Calculus I said we'd do the numerical work in Python. I began by writing and testing a program that simply defined the function we wanted to integrate and a function that accepted the limits of integration as parameters, using a Python list just to prove that I'd passed the arguments correctly.

```
1  # name, date and purpose here
2  #
3
4  import math
5
6  def f(x):
7      return math.sqrt(1 - x*x)
8
9  def integrate( b, t ):
10     """ documentation should go here, but this is just lecture ..."""
11     return [b,t]
12
13 print( f( 0.5))
14 print( integrate(0,1))
```

```
myshell> python integrate0.py
0.8660254037844386
[0, 1]
```

Since the function is computing $\sqrt{3/4}$, 0.87 is believable as the right answer. The second line of output shows I read the parameters correctly.

Next we decided to pass as a parameter the number of subdivisions of the interval, and to calculate the points of division. List comprehension (which we talked about at the end of the last class) was just the way to translate this mathematics

$$\left\{ b + j\frac{t-b}{n} \mid j = 0, 1, \ldots, n-1 \right\}$$

into Python.

I took the opportunity to sneak in the ability to provide default parameter values, and tested that too.

```
# name, date and purpose here
#

import math

def f(x):
    return math.sqrt(1 - x*x)

def integrate( b, t, n=10 ):
    """ documentation should go here, but this is just lecture ..."""
    divisions = [b + j*(t-b)/n for j in range(n)]
    return [divisions]

# print( f( 0.5))
print( integrate(0,1))
print( integrate(0,1,5))
```

```
myshell> python integrate1.py
[[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]]
[[0.0, 0.2, 0.4, 0.6, 0.8]]
```

Now we were ready for the final iteration. One of the students beat me to the idea that the function to be integrated should also be passed to the `integrate()` function. I knew that was possible in Python, but had never actually tried at, and didn't know whether any special syntax would be required. Given the choices *just try it* or *look it up* I always go for the try, so we did. I renamed the function defining the quarter circle, wrote the weird one in Equation 1.

I also noted the number of times I was calculating `j*(t-b)/n`, both in creating `divisions` and in the summation, so created the variable `delta` to save the value computed just once.

To test, I integrated three times. The first call found the area of the quarter circle with 10 subdivisions, the second with 100. I multiplied the answers by 4 so I could easily check the approximation to π. The third call computes the area under the weird function. It's pretty small, since each of its three factors is less than 1 on the unit interval.

```
myshell> python integrate2.py
3.304518326248318
3.160417031779047
0.06827253318842395
```

Here's the code:

```
# name, date and purpose here
#

import math

def g(x):
    return math.sqrt(1 - x*x)

def h(x):
    return math.exp(-x*x)*math.log(1+x)*math.sin(x**3)

def integrate( f, b, t, n=10 ):
    """ documentation should go here, but this is just lecture ..."""
    delta = (t-b)/n
    divisions = [b + j*delta for j in range(n)]
    sum = 0
    for x in divisions:
        sum += f(x)*delta
    return sum

```

```
21 print( 4*integrate(g, 0, 1))
22 print( 4*integrate(g, 0, 1, 100))
23 print( integrate(h, 0, 1, 100))
```

2 Doing it better

This code isn't robust. When I accidentally coded the weird function with `log(x)` instead of `log(1+x)` Python raised a `ValueError` because log(0) is undefined.

I don't know what would happen if I tried a value of n that wasn't a positive integer, or a value of f that wasn't a Python function. Would the code work if $b > t$? I'm sure it would be fine even if the function took on negative values.

The approximation would also be better if we used the trapezoid rule

The website `http://rosettacode.org/wiki/Numerical_integration#Python` does a better job in both precision and testing.

3 Trapezoid rule; converging to a better answer

```
# Integration
#
# Ethan Bolker
# March 2015, for Math 480

import math

def g(x):
    return math.sqrt(1 - x*x)

def h(x):
    return math.exp(-x*x)*math.log(1+x)*math.sin(x**3)

# This is the first successful integration routine we wrote
def riemann( f, b, t, n=10 ):
    """ compute integral_b^t f with vanilla Riemann sums """
    delta = (t-b)/n
    divisions = [b + j*delta for j in range(n)]
    sum = 0
    for x in divisions:
        sum += f(x)*delta
    return sum

# The trapezoid rule is more accurate, for about the same amount
# of arithmetic. It's inefficient since it computes f twice
# at the internal division points. I could make it run twice as fast.
def trapezoid0 ( f, b, t, n=10 ):
    """ compute integral_b^t f with trapezoid rule """
    delta = (t-b)/n
    divisions = [b + j*delta for j in range(n)]
    sum = 0
    for x in divisions:
        sum += (f(x) + f(x+delta))*delta/2
    return sum

# Redoing the trapezoid rule using the sum function
# and list comprehension.
def trapezoid ( f, b, t, n=10 ):
    """ compute integral_b^t f with trapezoid rule """
    delta = (t-b)/n
    divisions = [b + j*delta for j in range(n)]
    return sum([(f(x) + f(x+delta))*delta/2 for x in divisions])

#     height = [f(b + j*delta) for j in range(n+1)]
#     print( height)
#     value = [[height[j]+height[j+1] for j in range(n)]]
#     print( value )
#     return sum(value)
#     return sum([[height[j]+height[j+1] for j in range(n)]])

def integrate ( f, b, t, epsilon=0.01, debug = False ):
    """ compute integral_b^t f with trapezoid rule
        to within epsilon """
    scale = 1.1
#     scale = 1.2
    n = 1000
    previous_sum = trapezoid(f, b, t, n)
    n = math.ceil(scale*n)
```

```
    next_sum = trapezoid(f, b, t, n)
    if debug:
        print(epsilon)
        print( str(previous_sum) + "  " + str(next_sum))
    while abs(previous_sum - next_sum) > epsilon:
        previous_sum = next_sum
        n = math.ceil(scale*n)
        next_sum = trapezoid(f, b, t, n)
        if debug:
            print(n)
            print( str(previous_sum) + "  " + str(next_sum))
    return next_sum

print( 4*riemann(g, 0, 1, 1000))
print( 4*trapezoid0(g, 0, 1, 1000))
print( 4*trapezoid(g, 0, 1, 1000))
print( 4*integrate(g, 0, 1, 0.00000001, True))
print( 4*integrate(g, 0, 1, 0.00000001))
```

```
python integrate3.py
3.143555466911028
3.141555466911024
3.141555466911024
1e-08
0.785388866727756  0.7853901051688568
1210
0.7853901051688568  0.7853911786347608
1331
0.7853911786347608  0.7853921091017629
1465
0.7853921091017629  0.7853929204538184
1612
0.7853929204538184  0.7853936210024427
1774
0.7853936210024427  0.7853942287775434
1952
0.7853942287775434  0.785394754498148
2148
0.785394754498148  0.7853952102638213
2363
0.7853952102638213  0.7853956039896094
2600
0.7853956039896094  0.7853959458364763
2861
0.7853959458364763  0.7853962422563087
3148
0.7853962422563087  0.7853964988951585
Traceback (most recent call last):
  File "integrate3.py", line 75, in <module>
    print( 4*integrate(g, 0, 1, 0.00000001, True))
  File "integrate3.py", line 66, in integrate
    next_sum = trapezoid(f, b, t, n)
  File "integrate3.py", line 42, in trapezoid
    return sum([(f(x) + f(x+delta))*delta/2 for x in divisions])
  File "integrate3.py", line 42, in <listcomp>
    return sum([(f(x) + f(x+delta))*delta/2 for x in divisions])
  File "integrate3.py", line 9, in g
    return math.sqrt(1 - x*x)
ValueError: math domain error
```

```
Compilation exited abnormally with code 1 at Mon Mar 02 19:53:20
```

4 Monte Carlo integration

Using a random number generator to find areas. Maybe I can make this work for improper integrals too.

5 numpy and scipy

Here is the LaTeX source for this document. You can cut it from the pdf and use it to start your answers. I used the `\jobname` *macro for the source file name, so you can call your file by any name you like.*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Integration
% Math 480 Spring 2015
%

\documentclass[10pt]{article}
\usepackage[textheight=10in]{geometry}

\usepackage{verbatim}
\usepackage{amsmath}
\usepackage{amsfonts} % to get \mathbb letters

\usepackage[utf8]{inputenc}
\DeclareFixedFont{\ttb}{T1}{txtt}{bx}{n}{9} % for bold
\DeclareFixedFont{\ttm}{T1}{txtt}{m}{n}{9}  % for normal
% Defining colors
\usepackage{color}
\definecolor{deepblue}{rgb}{0,0,0.5}
\definecolor{deepred}{rgb}{0.6,0,0}
\definecolor{deepgreen}{rgb}{0,0.5,0}

\usepackage{listings}

%Python style from
%http://tex.stackexchange.com/questions/199375/problem-with-listings-package-for-python-syntax-colo
\newcommand\pythonstyle{\lstset{
  language=Python,
  backgroundcolor=\color{white}, %%%%%%%
  basicstyle=\ttm,
  keywordstyle=\ttb\color{deepblue},
  emph={MyClass,__init__},
  emphstyle=\ttb\color{deepred},
  stringstyle=\color{deepgreen},
  commentstyle=\color{red},  %%%%%%%%
  frame=tb,
  showstringspaces=false,
  numbers=left,numberstyle=\tiny,numbersep =5pt
}}

%On my computer on just this one file I get a weird error when
%using the hyperref package.
%
% Comment out the next two lines when the problem is solved
%\usepackage{hyperref}
\newcommand{\url}[1]{\texttt{#1}}

\begin{document}

\pythonstyle{}

%%%%%%%%%%%%%%%% start here %%%%%%%%%%%%%%%%
\begin{center}
\Large{
Integrating with Python \\
Ethan Bolker \\
\today
```

```
}
\end{center}

\section{Riemann sums}

These are approximate lecture notes from the fifth class of the
semester (February 26, 2015). I took advantage of the fact that I had
not prepared the
material to model the process of building a simple program from
scratch -- something I'd talked about but never done in front of the
students.

I began by asking how they would compute
%
\begin{equation*}
\int_0^1 \sqrt{1 - x^2} dx \quad .
\end{equation*}

I got several blank looks, one person suggesting a trig
substitution. When I drew the picture of a quarter of a circle they
all agreed that the answer should be $\pi/4$.

Then I asked about
%
\begin{equation}\label{eq:weird}
\int_0^1 \exp(-x^2) \log(1+x) \sin(x^3) dx \quad .
\end{equation}

Some said look at a table of integrals. When I told them there was a
theorem showing that function could not be integrated (using the kind
of functions they studied in calculus) they were stuck.

At that point I hinted at the definition of the definite integral and
they all remembered something about summing the areas of
rectangles. The standard blackboard picture of Riemann sums did the
trick.

After a brief but important digression on the Fundamental Theorem of
Calculus I said we'd do the numerical work in Python. I began by
writing and testing a program that simply defined the function we
wanted to integrate and a function that accepted the limits of
integration as parameters, using a Python list just to prove that I'd
passed the arguments correctly.

\lstinputlisting{integrate0.py}

\begin{verbatim}
myshell> python integrate0.py
0.8660254037844386
[0, 1]
\end{verbatim}

Since the function is computing $\sqrt{3/4}$, 0.87 is believable as
the right answer. The second line of output shows I read the
parameters correctly.

Next we decided to pass as a parameter the number of subdivisions of
the interval, and to calculate the points of division. List
comprehension (which we talked about at the end of the last class) was
just the way to translate this mathematics
```

```
%
\begin{equation*}
\left\{ b + j\frac{t-b}{n} \ | \  j = 0, 1, \dots, n-1\right\}
\end{equation*}
%
into Python.

I took the
opportunity to sneak in the ability to provide default parameter
values, and tested that too.

\lstinputlisting{integrate1.py}

\begin{verbatim}
myshell> python integrate1.py
[[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]]
[[0.0, 0.2, 0.4, 0.6, 0.8]]
\end{verbatim}

Now we were ready for the final iteration. One of the students beat me
to the idea that the function to be integrated should also be passed
to the \lstinline!integrate()! function. I knew that was possible in
Python, but had never actually tried at, and didn't know whether any
special syntax would be required. Given the choices \emph{just try it}
or \emph{look it up} I always go for the try, so we did. I renamed the
function defining the quarter circle, wrote the weird one in
Equation~\ref{eq:weird}.

I also noted the number of times I was calculating
\lstinline$j*(t-b)/n$, both in creating
\lstinline$divisions$ and in the summation, so created the variable
\lstinline$delta$ to save the value computed just once.

To test, I integrated three times.
The first call found the area of the quarter circle with 10
subdivisions, the second with 100. I multiplied the answers by 4 so I
could easily check the approximation to $\pi$. The third call computes
the area under the weird function. It's pretty small, since each of
its three factors is less than 1 on the unit interval.

\begin{verbatim}
myshell> python integrate2.py
3.304518326248318
3.160417031779047
0.06827253318842395
\end{verbatim}

Here's the code:

\lstinputlisting{integrate2.py}

\section{Doing it better}

This code isn't robust. When I accidentally coded the weird function
with \lstinline!log(x)! instead of \lstinline!log(1+x)! Python raised
a \lstinline!ValueError! because $\log(0)$ is undefined.

I don't know what would happen if I tried a value of $n$ that wasn't a
positive integer, or a value of $f$ that wasn't a Python
function. Would the code work if $b > t$? I'm sure it would be fine
```

```
even if the function took on negative values.

The approximation would also be better if we used the trapezoid rule

The website
\url{http://rosettacode.org/wiki/Numerical\_integration\#Python}
does a better
job in both precision and testing.

\newpage
\section{Trapezoid rule; converging to a better answer}

\lstinputlisting{integrate3.py}

\begin{verbatim}
python integrate3.py
3.143555466911028
3.141555466911024
3.141555466911024
1e-08
0.785388866727756  0.7853901051688568
1210
0.7853901051688568  0.7853911786347608
1331
0.7853911786347608  0.7853921091017629
1465
0.7853921091017629  0.7853929204538184
1612
0.7853929204538184  0.7853936210024427
1774
0.7853936210024427  0.7853942287775434
1952
0.7853942287775434  0.785394754498148
2148
0.785394754498148  0.7853952102638213
2363
0.7853952102638213  0.7853956039896094
2600
0.7853956039896094  0.7853959458364763
2861
0.7853959458364763  0.7853962422563087
3148
0.7853962422563087  0.7853964988951585
Traceback (most recent call last):
  File "integrate3.py", line 75, in <module>
    print( 4*integrate(g, 0, 1, 0.00000001, True))
  File "integrate3.py", line 66, in integrate
    next_sum = trapezoid(f, b, t, n)
  File "integrate3.py", line 42, in trapezoid
    return sum([(f(x) + f(x+delta))*delta/2 for x in divisions])
  File "integrate3.py", line 42, in <listcomp>
    return sum([(f(x) + f(x+delta))*delta/2 for x in divisions])
  File "integrate3.py", line 9, in g
    return math.sqrt(1 - x*x)
ValueError: math domain error

Compilation exited abnormally with code 1 at Mon Mar 02 19:53:20
\end{verbatim}

\section{Monte Carlo integration}
```

```
Using a random number generator to find areas. Maybe I can make this
work for improper integrals too.

\section{numpy and scipy}

\newpage
\emph{
Here is the \LaTeX{} source for this document. You can cut it from the
pdf and use it to start your answers. I used the} \verb!\jobname!
\emph{macro for the source file name, so you can call your file by any
  name you like.}
\verbatiminput{\jobname}

\end{document}
```