Computer Science Investigatory Project

Integration using Python

A Basic Implementation of Mathematical Techniques to Achieve Approximation of Definite Integrals in Python

Presented by
Sudhir Krisna K. N.
as the *Computer Science Investigatory Project* for the year *2024 - 2025*

Sri Vignesh Vidyalaya Senior Secondary School

CHAPTER

1

ACKNOWLEDGMENTS

CHAPTER

2

PROJECT DETAILS

The following specifications are required to complete this project.

1. 1 GHz single-core processor
2. 2 GB of RAM
3. 10 GB of free disk space
4. and a basic operating system like Windows 7/8/10, macOS, or Linux.

CONTENTS

CHAPTER

3

APPLICATION IN COMPUTER SCIENCE

3.1 Explanation of the Formula in terms of Computer Science

The basic formula for the approximation of an integral can be written as such.

$$AV(f(x),\, \Delta x)\Big|_a^b = \sum_{i=1}^{n} \left(f(x_i) \cdot \Delta x + \frac{1}{2} \cdot \Delta x \cdot [f(x_i + \Delta x) - f(x_i)] \right)$$

Now, we shall look at this using a computer-scientific lens with an algorithm and a flowchart to explain this process.

Before we take a look at the algorithm, let us first see how the summation can be represented in terms of Computer Science. For instance, the summation,

$$\sum_{i=1}^{10} i^2$$

can be represented in terms of a `for` loop as (in pseudo code),

Pseudocode for Summation

```
sum = 0;
for (int i = 1; i <= 10; i++){
    sum = sum + square(i);
}
```

Here, a variable `sum` is first created to store the sum of the values obtained during the iterations. In the declaration of the `for` loop, we first declare the iterator `i`, which stores the initial value of the summation ($i = 1$); the condition for the `for` loop's termination, which is the endpoint of the summation ($i = 10$); and finally, `i++` means that the value of `i` increases by 1

every iteration.

Inside the `for` loop, we increase the value of sum by a function (here, `square()`) performed on the value of the iterator. After the `for` loop ends, the value of the summation is stored in the variable `sum`, which can then be accessed elsewhere or displayed.

In our case, however, we must not perform the function on the iterator's current value (say `i`), but rather the `i`th element of a set containing values. Now, we shall represent this in a more general fashion.

Let `list` be the set of values to be operated on, and `n` be the number of items in the list. Since list indexing begins with 0 in most languages (the first element of a list is referenced as `list[0]` and the nth element is referenced as `list[n - 1]`), we shall reduce the values of `i` and `n` by 1 in the declaration of the for loop.

Now, the summation of

$$\sum_{i=1}^{n} f(x_i)$$

where x_i is the ith element in the list, is given by

Pseudocode for Summation

```
list = [.....];
sum = 0;
for (int i = 0; i <= n - 1; i++) {
    sum = sum + func(list[i]);
}
```

where `func()` is the function to be performed on the values inside the list.

Now that we understand how to represent a summation in programming languages, let us take a look at the algorithm for approximate integration.

3.2 Algorithms

First, let us see the algorithm for the Approximate Value function, and then take a look at one for the whole program.

3.2.1 Approximate Value Function Algorithm

1. Declare the function `AV()` using the following parameters:
   - the function to be integrated (`f(x)`),
   - the start point (`a`) and end point (`b`),
   - and the value of the size of the intervals (`dx`).
2. Declare a list `l` to hold the values of the range of the integration.
3. Populate the list using the following steps:

(a) Find the number of elements in the list using the formula

$$n = \frac{b-a}{dx}$$

(b) Iterate through a `for` loop from `i = 0` to `i = n` (This excludes the final value `n`).

(c) For each iteration, append the `i`th element of the list to `l`. To find the `i`th element, use the formula

```
l[i] = a + (i * dx)
```

This will end in the final value being $b - dx$, as mentioned in the mathematical explanation.

4. Declare a variable `area` to store the sum of the areas computed in the following loop and another variable `temp` to store the temporary values of the areas.
5. Now, iterate through a new `for` loop from `i = 0` to the aforementioned `i = n`.
6. In every iteration, find the area of the interval using *Formula* 1, i.e.,

```
rectangle = f(l[i]) * dx
triangle = 0.5 * dx * (f(l[i] + dx) - f(l[i])
```

7. Store the values of the area of the rectangle and the triangle in `temp`.
8. Add the value of `temp` to `sum` and move on to the next iteration.
9. After the final iteration is over, return the value of `sum`.

3.2.2 Approximate Value Function Pseudo Code

The following pseudo-code provides a better understanding of the above algorithm in code form.

Pseudocode for Approximate Value Function

```
define AV(f, a, b, dx) {
    l = new list;
    n = (b - a)/dx;
    for (int i = 0; i < n; i++) {
        l[i] = a + i*dx;
    }
    area, temp = 0;
    for (int i = 0; i < n; i++) {
        temp = f(l[i])*dx + 0.5*dx*(f(l[i] + dx) - f(l[i]));
        sum = sum + temp;
    }
    return sum;
}
```

Using this definition, let us move on to the algorithm for the complete program.

3.2.3 Whole Program Algorithm

1. Start
2. Receive the function `f` to be integrated.
3. Receive the start point `a` and end point `b` of the integral.
4. Receive the accuracy of the program `dx`.
5. Evaluate `AV(f, a, b, dx)`.
6. Display the output.

3.3 General Applications of Integrals

Integrals are employed in a wide array of fields, most prominently in pure mathematics, statistics and physics.

One important application of integrals in computer science is the real time calculation of computer graphics, for instance, ray-tracing. Although integrals aren't used widely in the field of computer science specifically, the utilization of computers for calculating integrals has contributed to huge progress in other fields. Many problems which might have taken large amounts of man and/or brain power have been solved easily with the ability to calculate integrals on a computer.

For example, computers are used to calculate integrals pertaining to:

- finding the probability of an event in a complex field, finding the average of a continuous distribution, finding the centroids of certain shapes and so on in mathematics.
- finding the center of mass of a given object, finding quantities such as work, charge distribution, etc. where two quantities depend on each other in a graphical sense, and so on in physics.

In fact, as integration is considered an inverse of differentiation, it is also used to find the actual function from a given rate of change (In physics, this feature is crucial in solving certain problems.)

After implementing integration in Python, we shall use the program to solve a few problems from the fields listed above.

CHAPTER

4

IMPLEMENTATION IN PYTHON

4.1 Creating the Approximate Value Function in Python

Now, let us try to do this in Python. Using the above algorithm, the following function can be written.

```
def integral(begin, end, dx, f):
    area = 0
    for i in [begin + dx*n for n in range(int((end - begin)/dx))]:
        area += (f(i)*(dx)) + (0.5 * dx * (f(i + dx) - f(i)))
    return area
```

The above function may seen confusing due to the emphasis on efficiency. However, it can also be written in the following way to enhance its legibility.

```
def integral(begin, end, dx, f):
area = 0
list_of_points = []
number_of_points = int((end - begin)/dx)

for n in range(number_of_points):
    point = begin + dx*n
    list_of_points.append(point)

for i in list_of_points:
    rectangle = f(i)*dx
    triangle = (0.5 * dx * (f(i + dx) - f(i)))
    area += rectangle + triangle
```

```
    return area
```

Although the program varies from the algorithm slightly, its essence is the same. The variations are features offered by Python to help with efficiency, and are a net benefit.
One can also use the module `numpy` to improve the efficiency of the program.

A Brief Introduction to NumPy:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The predecessor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors. NumPy is a NumFOCUS fiscally sponsored project.

Below is a sample code that shows a rather crude implementation using `numpy`.

```
import numpy as np

def integral(begin, end, h, f):
    elms = int((end-begin)/h)
    x = np.linspace(begin, end, elms + 1)
    area = 0
    for i in range(elms):
        area += (f(x[i])*(h)) + (0.5 * h * (f(x[i + 1]) - f(x[i])))
    return area
```

This can be further improved on efficiency. Still, this code is more viable as certain bugs arise from using pure Python which can easily be avoided if we use the `numpy` module. The algorithm behind the code is the same, with the only difference being that a linear space is created to store the values of the points where the function is calculated. (Here, `h` is used in place of `dx`.)

4.2 Adding Graphs to improve understanding

This is an auxiliary section which is just a quality of life addition to our program. Using the `matplotlib` library, we can add a graph alongside the computation of the integral to easily understand how the area is calculated.

A Brief Introduction to Matplotlib:

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

Below is a block of code that defines the function used to plot graphs with `matplotlib`.

```
import numpy as np
import matplotlib.pyplot as plt

def pltgrph(begin, end, h, f):
    elms = int(end/h)
    spc = (end - begin)/3
    x = np.linspace(begin - spc, end + spc, elms + 1)
    y = np.linspace(min(f(x)) - spc, max(f(x) + spc), elms + 1)
    plt.plot(x, f(x))
    plt.plot(x, np.linspace(0, 0, elms+ 1), color='black',
    ↪  linestyle='dotted')
    plt.plot(np.linspace(end, end, elms+1), np.linspace(0, f(end),
    ↪  elms+1), linestyle="--", color='red')
    plt.plot(np.linspace(begin, begin, elms+1), np.linspace(0,
    ↪  f(begin), elms+1), linestyle='--', color='red')
    plt.show()
```

This code produces a graph upon inputting the same parameters as our Approximate Value Function.

Let us take a look at an example, with an explanation of the resulting graph.

Example 4.1: Graphing using MatPlotLib

Graph the integral of $f(x) = x^2 + 2x + 3$ from $x = 0$ to $x = 2$.
The arguments given to the `pltgrph()` function are as follows:

```
def f(x):
    return x**2 + 2*x + 3

begin = 0
end = 2
h = 0.01
```

The resulting output is shown in Figure 4.1.

As we can see, the area enclosed within the graph line and the two dashed vertical lines is the area to be calculated. Henceforth, all examples shall be accompanied by such graphs.

Example 4.2: Using the Integration tool for Complex Functions

Find the area under the curve $y = e^{cos(x)}$ from $x = 0$ to $x = \frac{\pi}{2}$
Executing the program with the following parameters,

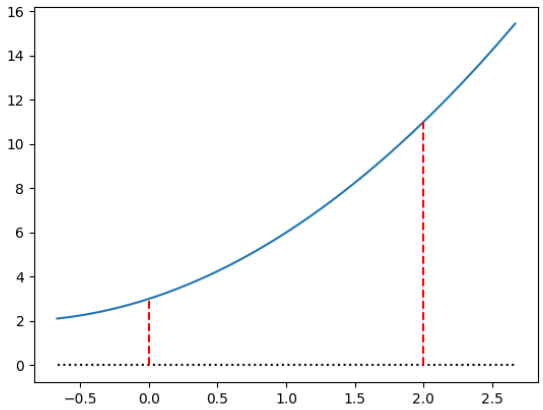

Figure 4.1: Output for *Example* 5.1

```
def f(x):
    return pow(np.e, np.cos(x))

begin = 0
end = np.pi * 0.5
dx = 0.001
```

we get,

```
>>> main.py
3.103582290508624
```

Comparing this result with one derived from the F.T.C, we get,

$$\int_0^{\frac{\pi}{2}} e^{\cos(x)}\, dx = \frac{1}{2}\pi\left(\mathbf{L}_0(1) + I_0(1)\right) \approx 3.10438$$

Figure 4.2: Actual Result (WolframAlpha)

which has a 0.002% variation from our output.

CHAPTER

5

GUI IMPLEMENTATION

5.1 Introduction to Graphical User Interfaces

All this while, we have been using the terminal to access our program. While this is acceptable in development, a graphical user interface has become the norm in deployment. In this section, we shall use the `tkinter` module to develop a graphical user interface for our program, which will also involve a few changes to the functions we defined previously.

A Brief Introduction to Tkinter

Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and macOS installs of Python. The name Tkinter comes from Tk interface. Tkinter was written by Steen Lumholt and Guido van Rossum, then later revised by Fredrik Lundh. Tkinter is free software released under a Python license.

5.2 Implementing the Basic Interface

Before we implement the GUI, let us import the necessary modules for tkinter.

```
from tkinter import *
from tkinter import ttk
```

We shall first define the necessary widgets to be packed in our Tkinter Grid (An explanation of this, similar to Matplotlib, is outside the scope of the project). Since this code is rather large and repetitive, we shall focus on the functionality. Refer the source code for the definitions of the various blocks and widgets.

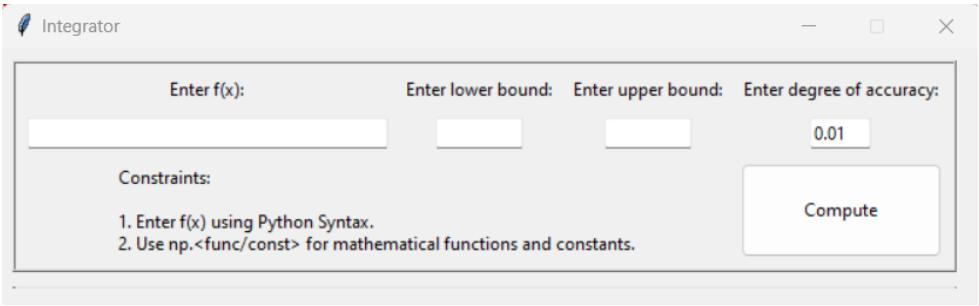

Figure 5.1: Basic Graphical Interface without Input

Now that we have defined the input widgets, it is time to add functionality to the interactive part of the UI - the button. The following function is executed when the button is pressed.

```
def compute(*args):
    try:
        f = functionInput.get()
        a = eval(aval.get())
        b = eval(bval.get())
        dx = eval(dxval.get())
        pltgrph(eval("lambda x: " + f), a, b, dx)
    except ValueError:
        pass
    except Exception as e:
        functionInput.set("Invalid / Non Integrable Function")
```

The `try-except` block is used to ensure that the function does not result in an error when the button is pressed without any input, as well as warn the user when the function they define is non-integrable.

The `eval` keyword is used to convert the expressions that the user enters into code that Python can understand, and consequently, evaluate. An anonymous function (`lambda`) is used to convert the input function into an actual python function.

5.3 Revamping the Graph

The `pltgrph()` function which we defined earlier is called. However, it has undergone a few changes to make it suitable for a graphical environment.

```
def pltgrph(f, begin, end, h):
    fig = plt.figure(1)
    plt.ion()
    plt.clf()
    elms = int(end/h)
    spc = (end - begin)/3
    x = np.linspace(begin - spc, end + spc, elms + 1)
    y = np.linspace(min(f(x)) - spc, max(f(x) + spc), elms + 1)
```

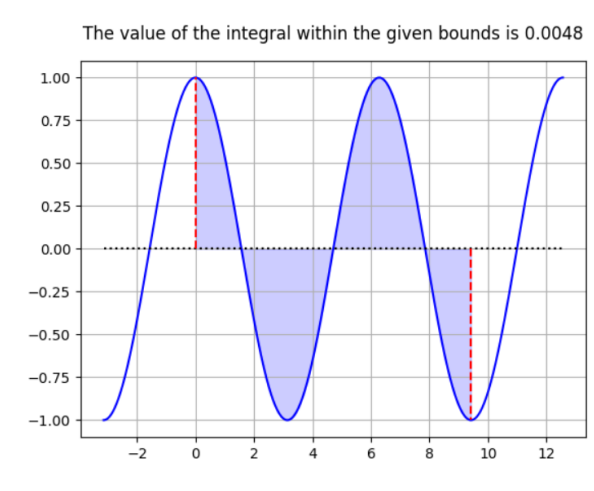

Figure 5.2: Improved Graph

```
plt.plot(x, f(x), color='blue')
plt.plot(x, np.linspace(0, 0, elms+ 1), color='black',
↪  linestyle='dotted')
plt.plot(np.linspace(end, end, elms+1), np.linspace(0, f(end),
↪  elms+1), linestyle="--", color='red')
plt.plot(np.linspace(begin, begin, elms+1), np.linspace(0, f(begin),
↪  elms+1), linestyle='--', color='red')
plt.fill_between(x= x, y1= f(x), where= (begin < x)&(x < end),color=
↪  "b",alpha= 0.2)
plt.title("The value of the integral within the given bounds is
↪  {}".format(round(integral(f, begin, end, h), 4)))
plt.grid(True)
```

Essentially, it now displays the result of the calculation within the graph itself. The area under the graph is also shaded to improve understandability. A graph computed using this version is given.

The final version of the GUI, with an example, is also given.

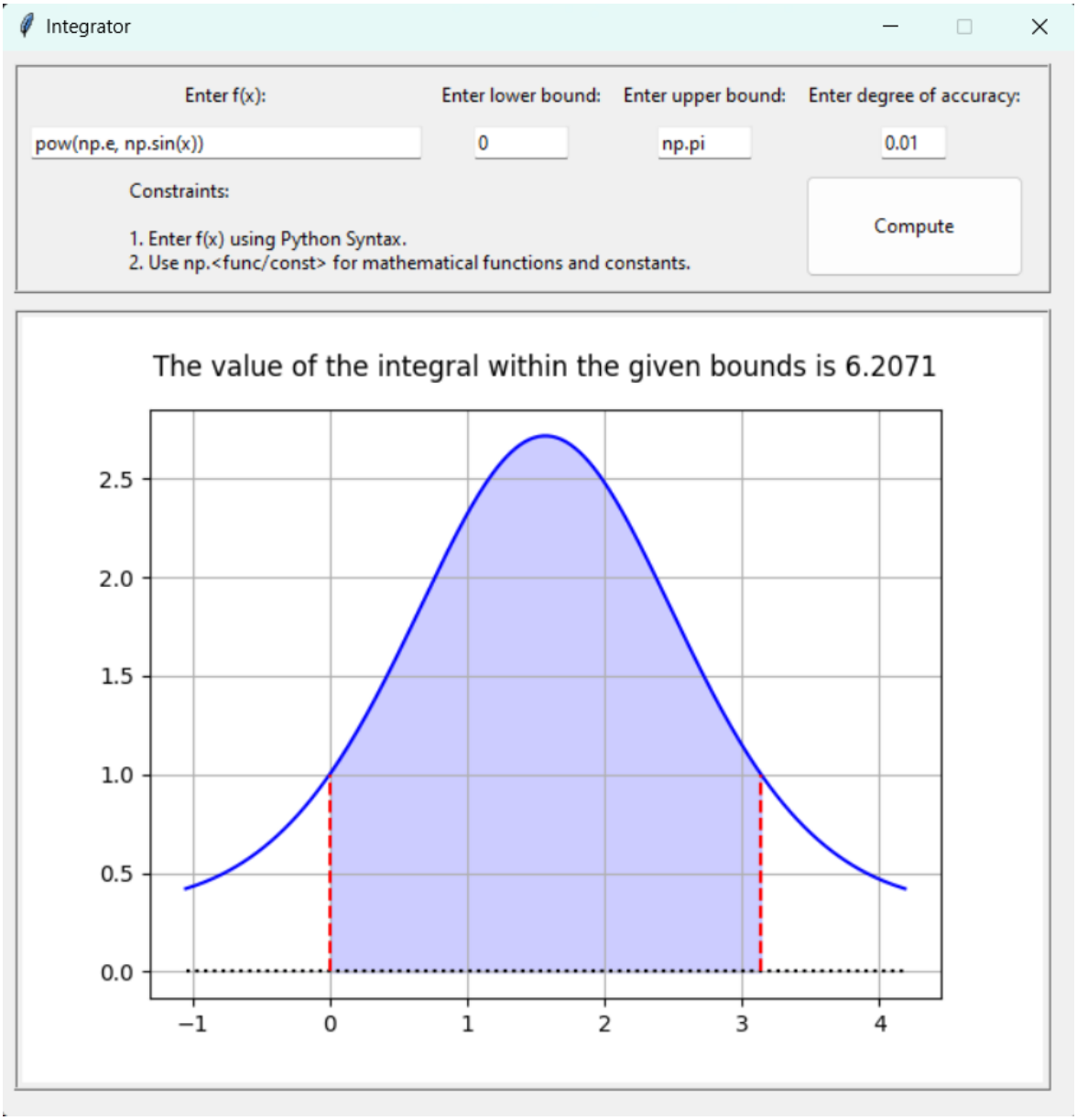

Figure 5.3: Final GUI with Example

CHAPTER

6

EXAMPLES

6.1 Basic Examples

Let us start with a simple practical example. Some methods of calculus not mentioned in the text shall also be used to solve these problems, and they will be explained alongside, albeit not with much detail.

Note: Most of the examples will involve the usage of mathematics (without including integration) to quite a large extent. This is similar to solving actual problems, which cannot be worked out on its own by a tool. Rather, many adjustments have to be made so that we can feed the problem into our program as a set of parameters.

Example 6.1: Physics Example

Problem: If the acceleration of a moving object as a function of time is given by $a(t) = 2t$, and its velocity at $t = 1$ is 2 m/s, find the distance travelled by the object in the first 10 seconds.

We know that velocity is the rate of change of distance and acceleration is the rate of change of velocity. Hence, we may use these to derive a formula.

$$a = \frac{v}{t}$$
$$v = \frac{s}{t}$$

For small changes in the value of area, velocity and distance, we may modify the formulae as such.

$$da = \frac{dv}{dt}$$
$$dv = \frac{ds}{dt}$$

In order to solve the question, we may rearrange the terms as follows.

$$\begin{aligned} da \cdot dt &= dv \\ \int da \cdot dt &= \int dv \\ \int dv &= \int a(t) \cdot dt \\ \int dv &= \int 2t \, dt \end{aligned}$$

Since dt and dv means small divisions of t and v respectively, we can use the integral sign to denote the sum of all such divisions, i.e., the whole. Therefore, we integrate dv to get v.

Moreover, we add an arbitrary constant. The reasoning for this is that when we differentiate (inverse of integrate) a function, we get rid of all constant terms. In order to show that a constant has to be added before solving, we add $+C$ to one side of the equation.

Now, since an antiderivative is the inverse of a derivative, we need to find a function such that its derivative is $2t$. Here, the required function is t^2.

$$v = t^2 + C$$

We can now find a function for the velocity of the object as a function of time. We know that the velocity at $t = 1$ is 2. We can substitute that into our equation to find the value of C.

$$\begin{aligned} 2 &= 1^2 + C \\ 2 - 1 &= C \\ C &= 1 \end{aligned}$$

Note: In case we had not added a constant, we would have gotten $2 = 1$ which is evidently not possible.

Therefore, the velocity as a function of time can be written as $v(t) = t^2 + 1$. Following the same method, we can now find the distance travelled by the object. However, here we have the exact of t within which we need to find the distance travelled. Thus, we shall use our program here.

$$\begin{aligned} ds &= dv \cdot dt \\ \int ds &= \int_0^{10} v(t) \, dt \\ s &= \int_0^{10} t^2 + 1 \, dt \end{aligned}$$

In order to input these values as parameters in our program, let us change t to x. Integrating $x^2 + 1$ from 0 to 10 with these parameters,

```
def f(x):
    return pow(x, 2) + 1
```

```
begin = 0
end = 10
dx = 0.001
```

we get 343.333 (refer output), which is the distance travelled by the object in the first 10 seconds.

Many such problems in physics can be solved using integration, as long as we know the dependence of certain variables on others.

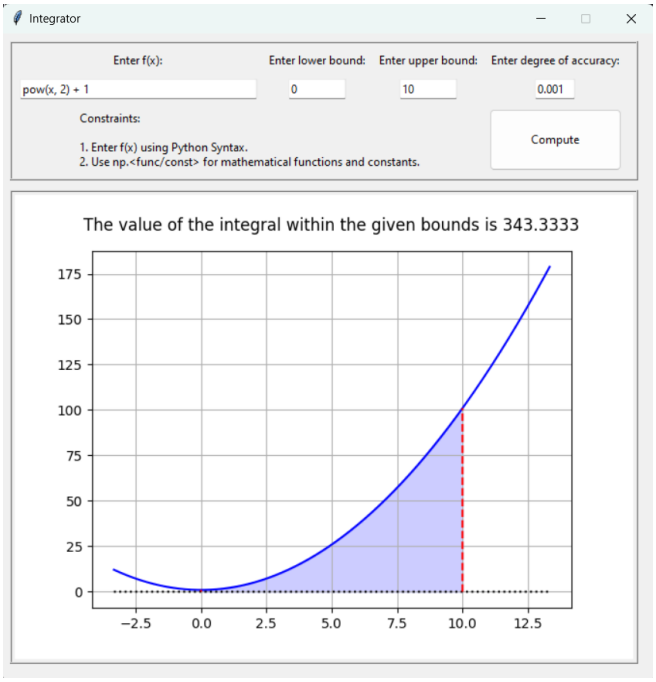

Figure 6.1: Output for *Example* 6.1

Now, let us look at a mathematical example. This will be much simpler than the previous example, as we will not have to do many physics calculations.

Example 6.2: Mathematical Example

Problem: Find the area of an ellipse (Figure 4.2) with 10 *cm* and 6 *cm* as the lengths of its major and minor axes respectively.

Let us graph the ellipse (given below), with the major axis being the x-axis and the minor axis being the y-axis.

Using the formula for an ellipse, we can convert the given data into an equation.

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

Here, a and b represent the half the lengths of the major and minor axes respectively. In order for our program to process the function, we have to feed it in the form of $y = f(x)$,

so we shall convert the resulting equation.

$$\begin{aligned}\frac{x^2}{25} + \frac{y^2}{9} &= 1\\ \frac{y^2}{9} &= 1 - \frac{x^2}{25}\\ y^2 &= 9\left(1 - \frac{x^2}{25}\right)\\ y &= \pm\sqrt{9\left(1 - \frac{x^2}{25}\right)}\end{aligned}$$

The plus-or-minus sign is because the square root function only outputs the positive value. However, we cannot input both outputs in our parameters. Since the results are the same with only the sign being different, we can consider only the positive results as being a semi-ellipse, with area equal to half of the total area. So, in order to find the area of the original ellipse, we can double the area of the semi-ellipse.

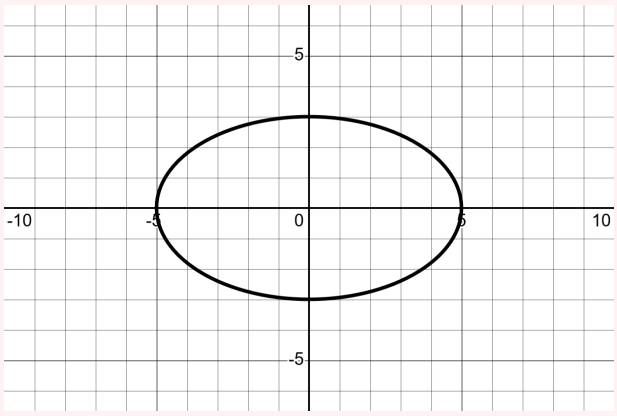

Figure 6.2: Graph of Ellipse

We can now run the program with the following parameters.

```
def f(x):
    return pow(9 * (1 - (pow(x, 2)/25)), 0.5)

begin = -5
end = 5
dx = 0.01
```

we get 23.5612.

Since our output is only half of the total area, we double it to get the area of the original ellipse.

$$23.5612 \times 2 = 47.1224$$

which is the area that we need.

Now, let's take a look at a situation where integration can be used in economics.

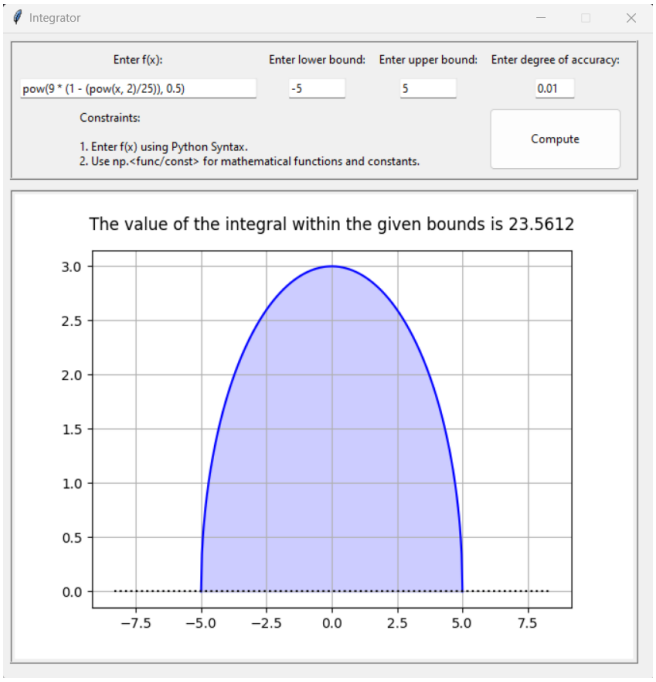

Figure 6.3: Output for *Example* 6.2

Example 6.3: Economics Example

Problem: The net profit when manufacturing x number of toys is given by $P(x) = 0.1x^2 - 4x + 2$ dollars. Find the average profit when manufacturing 0 to 80 toys. (Treat the function as continuous at all non-integral points).

This is quite a short and straightforward example. We can simply plug the values into the formula for the average value of a function.

$$AVG = \frac{1}{b-a}\int_a^b f(x)\,dx$$
$$AVG = \frac{1}{80-0}\int_0^{80} (0.1x^2 - 4x + 2)\,dx$$

Evaluating the definite integral using the following parameters,

```
def f(x):
    return 0.1 * pow(x, 2) - 4*x + 2

begin = 0
end = 80
dx = 0.001
```

we get 4426.6668 (refer output).

Substituting this value in our equation we get,

$$AVG = \frac{4426.66}{80}$$
$$AVG = 55.33325$$

$\therefore$ The average profit is 55.33325 dollars.

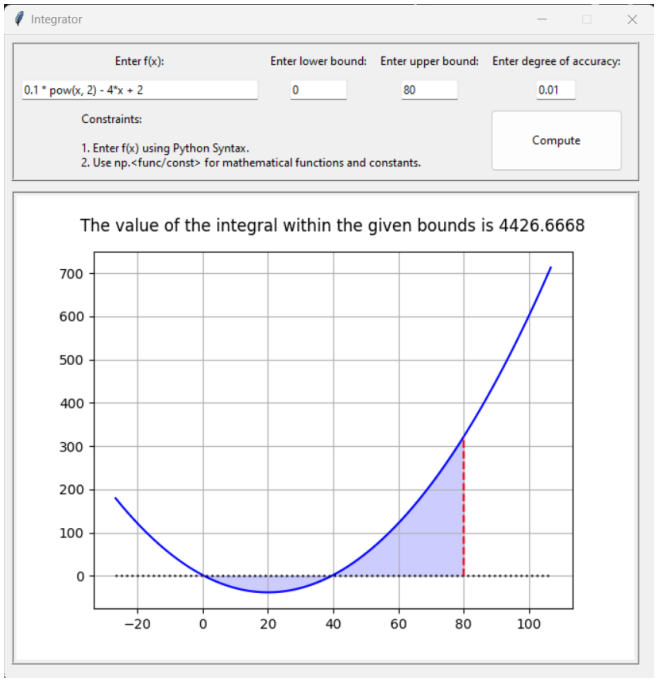

Figure 6.4: Output for Example 6.3

As we can see, the profit is negative when the number of toys manufactured is less than 40, and then increases. While this may not be the exact equation for the profit of a company, this emulates how integration in Python can be used for application in business as well.

CHAPTER

7

SOURCE CODE

The source code can be found on Github at *https://github.com/Sudhboi/integrator* as well.

```
from tkinter import *
from tkinter import ttk
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg)
from matplotlib import rcParams
rcParams['axes.titlepad'] = 15

def integral(f, begin, end, dx):
    area = 0
    for i in [begin + dx*n for n in range(int((end - begin)/dx))]:
        area += (f(i)*(dx)) + (0.5 * dx * (f(i + dx) - f(i)))
    return area

def pltgrph(f, begin, end, h):
    fig = plt.figure(1)
    plt.ion()
    plt.clf()
    elms = int(end/h)
    spc = (end - begin)/3
    x = np.linspace(begin - spc, end + spc, elms + 1)
    y = np.linspace(min(f(x)) - spc, max(f(x) + spc), elms + 1)
    plt.plot(x, f(x), color='blue')
    plt.plot(x, np.linspace(0, 0, elms+ 1), color='black',
    ↪  linestyle='dotted')
```

```
    plt.plot(np.linspace(end, end, elms+1), np.linspace(0, f(end),
    ↪  elms+1), linestyle="--", color='red')
    plt.plot(np.linspace(begin, begin, elms+1), np.linspace(0, f(begin),
    ↪  elms+1), linestyle='--', color='red')
    plt.fill_between(x= x, y1= f(x), where= (begin < x)&(x < end),color=
    ↪  "b",alpha= 0.2)
    plt.title("The value of the integral within the given bounds is
    ↪  {}".format(round(integral(f, begin, end, h), 4)))
    plt.grid(True)

    canvas = FigureCanvasTkAgg(fig, master = graphFrame)
    plot_widget = canvas.get_tk_widget()
    plot_widget.grid(row=0, column=0)

def compute(*args):
    try:
        f = functionInput.get()
        a = eval(aval.get())
        b = eval(bval.get())
        dx = eval(dxval.get())
        pltgrph(eval("lambda x: " + f), a, b, dx)
    except ValueError:
        pass
    except Exception as e:
        functionInput.set("Invalid / Non Integrable Function")

root = Tk()
root.title("Integrator")
root.resizable(False, False)

mainframe = ttk.Frame(root, padding="3 3 12 12")
mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)

inputFrame = ttk.Frame(mainframe, borderwidth=5, relief="ridge")
inputFrame.grid(column=0, row=0, sticky=(W, E), columnspan=6, rowspan=3)

functionInput = StringVar()
functionEntry = ttk.Entry(inputFrame, width=40,
↪  textvariable=functionInput)
ttk.Label(inputFrame, text="Enter f(x):").grid(column=0, row=0,
↪  columnspan=3)
functionEntry.grid(column=0, row=1, columnspan=3, sticky=(W, E))

aval = StringVar()
aEntry = ttk.Entry(inputFrame, width=9, textvariable=aval)
ttk.Label(inputFrame, text="Enter lower bound:").grid(column=3, row=0)
```

```
aEntry.grid(column=3, row=1)

bval = StringVar()
bEntry = ttk.Entry(inputFrame, width=9, textvariable=bval)
ttk.Label(inputFrame, text="Enter upper bound:").grid(column=4, row=0)
bEntry.grid(column=4, row=1)

dxval = StringVar()
dxval.set("0.01")
dxEntry = ttk.Entry(inputFrame, width=6, textvariable=dxval)
ttk.Label(inputFrame, text="Enter degree of accuracy:").grid(column=5,
↪  row=0)
dxEntry.grid(column=5, row=1)

detailText = "Constraints:\n\n1. Enter f(x) using Python Syntax.\n2. Use
↪  np.<func/const> for mathematical functions and constants."
details = ttk.Label(inputFrame, text=detailText)
details.grid(columnspan=5, column=0, row=2)

comButton = ttk.Button(inputFrame, text="Compute", command=compute)
comButton.grid(column=5, row=2, rowspan=2, sticky=(N, W, E, S))

graphFrame = ttk.Frame(mainframe, borderwidth=5, relief="ridge")
graphFrame.grid(column= 0, row= 3, columnspan=6, sticky=(N, W, E, S))

for child in inputFrame.winfo_children():
    child.grid_configure(padx=5, pady=5)

for child in mainframe.winfo_children():
    child.grid_configure(padx=5, pady=5)

functionEntry.focus()
root.mainloop()
```

CHAPTER

8

CONCLUSION

8.1 References

1. **James Stewart - Calculus: Early Transcendentals 9th Edition** - Used for certain examples, as well as the general idea for the approximation of a function.
2. **NCERT Mathematics Textbook for Grade 12** - *https://ncert.nic.in/ncerts /l/lemh201.pdf* - Used for the definitions, and procedure to calculate integrals using the Fundamental Theorem of Calculus.
3. **Paul's Online Notes** - *https://tutorial.math.lamar.edu/Classes/CalcII/ CenterOfMass.aspx* - Used for the procedure to find the centre of mass, as well as an example.
4. **Summation - Wikipedia** - *https://en.wikipedia.org/wiki/Summation* - Used to express the procedure for approximation in the form of a formula.
5. **Sumita Arora Computer Science Textbook for Class 11 and 12** - Used for the basics of Python programming and formulation of algorithms.
6. **Numpy - Wikipedia** - *https://en.wikipedia.org/wiki/NumPy* - Used for a Brief Introduction to Numpy.
7. **Matplotlib - Wikipedia** - *https://en.wikipedia.org/wiki/Matplotlib* - Used for a Brief Introduction to Matplotlib.
8. **Tkinter - Wikipedia** - *https://en.wikipedia.org/wiki/Tkinter* - Used for a Brief Introduction to Tkinter.
9. **Udemy - Computational Physics: Scientific Programming with Python by Dr. Börge Göbel** - *https://www.udemy.com/course/computational-physics/* - Used for basic programming in NumPy and Matplotlib.
10. **Several Documentations** - Used for programming in Python, Numpy, Matplotlib and Tkinter.

11. **UTRGV** - *https://www.utrgv.edu/cstem/utrgv-calculus/integration/average-value/ index.htm* - Used for the average value of a function procedure.

8.2 Tools

1. **Desmos Graphing Calculator** - *https://www.desmos.com/calculator* - Used to check most of the graphs, as well as draw the ones in Example 12 and 15.
2. **Wolfram Alpha** - *https://www.wolframalpha.com/* - Used to check all of the calculations done using the Fundamental Theorem of Calculus.
3. **Visual Studio Code** - *https://code.visualstudio.com/* - Used as a code environment for all the programs.
4. **Python** - *https://www.python.org/* - Duh.
5. **Numpy, Matplotlib and Tkinter** - *https://numpy.org/*, *https://matplotlib.org/* and *https://tkdocs.com/index.html* - Used for alternate program, drawing graphs and GUI.

8.3 Further Reading and Additional Materials

1. **Multivariable Calculus by James Stewart**
2. **Computational Physics and Computational Mathematics**
3. **Calculus I to III - University Level Courses**
4. **Programming in Python**

A multitude of additional materials is available online.