University of Massachusetts Lowell
Department of Mathematical Sciences
MATH 4750
Senior Seminar Project Report

Joel Savitz
Fall 2020
SID: 01739537

August 24, 2020

Abstract

//TODO

1 Introduction

Scientists express physical laws of the universe in the language of differential equations. The reason for this is not immediately obvious, though my personal work with differential equations has provided me with an intuition as to why. Fortunately for me, one scientist describes three necessary properties of a physical law that are satisfied by a differential equation as:

- "The mathematical relation must be sufficiently general"
- "It must define connections between neighboring points"
- "It must imply the continuity of change"[Siddiqui, 2014]

Differential equations satisfy all three of these properties since they are general enough that changes to the initial values of a system do not violate the constraints defined by a general solution and the relationship between a function and one or more of its derivatives defines the connection between points in the domain of the function and implies that change in the system is continuous due to the nature of calculus.

Newton's law of cooling states that "the rate of heat loss of a body is directly proportional to the difference in the temperatures between the body and its surroundings" [Wikipedia, 2020].

Let ΔT be the difference between the temperature of the body and the ambient temperature of the environment as a function of a point in time t, we have that equation 1 describes Newton's law of cooling.

$$\frac{d\Delta T}{dt} = k\Delta T(t) \text{ for some } k \in \mathbb{R} \tag{1}$$

Then, we can solve this differential equation by separating the paramaters.

$$\begin{aligned} \frac{dT}{dt}\frac{1}{\Delta T(t)} &= k & (2) \\ \int \frac{dT}{dt}\frac{1}{\Delta T(t)}dt &= \int kdt & (3) \\ \ln|\Delta T(t)| &= kt + C \text{ for some } C \in \mathbb{R} & (4) \\ e^{\ln|\Delta T(t)|} &= e^{kt+C} & (5) \\ \Delta T(t) &= De^{kt} \text{ where } D = e^C & (6) \end{aligned}$$

In this paper, I verify the this law relative to a particular environment and calculate an approximate value of the cooling coefficient by fitting experimental data to equation 6.

2 Materials and Methods

I performed this experiment using using a hardware sensor to a Raspbery Pi computer.

I purchased the DS18D20, a waterproof temperature sensor compatible with the Raspberry Pi. These sensors are relatively innexpensive, and I purchased a 5 pack for $12.98 on Amazon [Gikfun, 2020].

Figure 1: The full experimental setup

Figure 2

I wrote two programs to analyze the data. See sections 5 and 6 for the full code listings.

3 Results

4 Discussion

Newton was right lol

5 Appendix A: The `gather` program

5.1 Main program

```
#!/usr/bin/python3
```

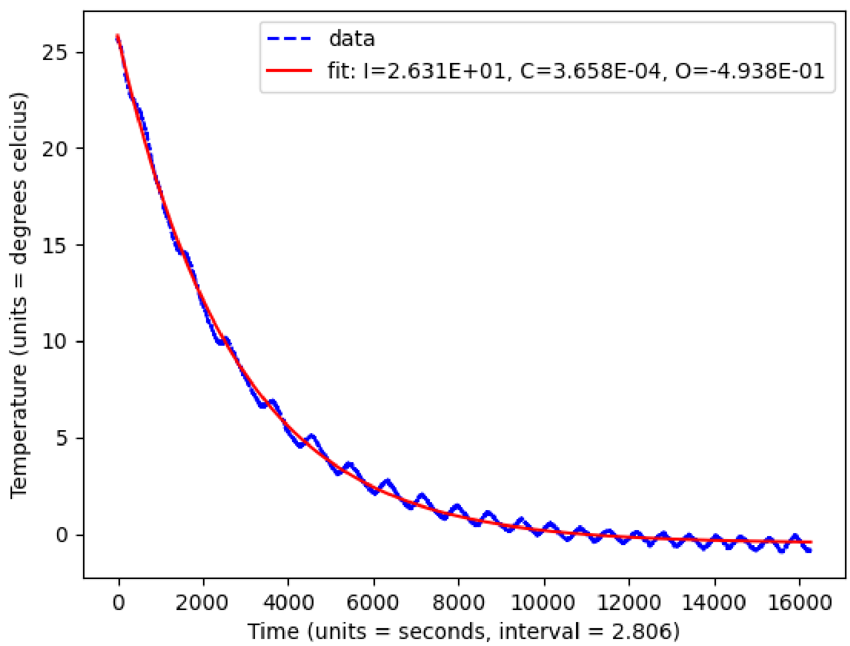

Figure 2: A plot of the data and the fitted curve

```python
# Based on code from https://learn.adafruit.com/
    adafruits-raspberry-pi-lesson-11-ds18b20-temperature
    -sensing/software

import glob
import time
import datetime
import sys
import signal

BASE_DIR = '/sys/bus/w1/devices/'
SENSOR_DIRS = glob.glob(BASE_DIR + '28*')
NUM_SRCS = len(SENSOR_DIRS)
DATA_SRCS = [SENSOR_DIRS[i] + '/w1_slave' for i in
    range(NUM_SRCS)]

# Print the date on recipt of SIGINT
def print_date_on_sigint(sig, frame):
    print("End sensor reading at {}".format(str(
        datetime.datetime.now())))
    sys.exit(0)

# Get the pure, raw, uncensored sensor data
def read_temp_raw(index):
    srcfile = open(DATA_SRCS[i], 'r')
    lines = srcfile.readlines()
    srcfile.close()
    return lines

# Extract the actual temperature value from the raw
    data
def read_temp(index):
    lines = read_temp_raw(index)
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
```

```
        temp_string = lines[1][equals_pos+2:]
        temp_c = format(float(temp_string) / 1000.0, '
            .2f')
        return temp_c


# Register the signal handler and read on loop until
    forcibly stopped
signal.signal(signal.SIGINT, print_date_on_sigint)
print("Begin sensor reading at {}".format(str(datetime.
    datetime.now())))
while True:
    for i in range(NUM_SRCS):
        print(read_temp(i), end=' ' if i < NUM_SRCS - 1
            else '\n')
    time.sleep(1)
```

5.2 Wrapper shell script

```
#!/bin/bash

usage() {
echo << DELIMITER
Usage: gather.sh [-o] [-f <output_filename>

-f: specify output filename (defaults to 'data')
-o: overwrite file instead of append

DELIMITER
}

FILENAME="data"
TEE_ARGS="-a"

while getopts "f:o" OPTION; do
        case ${OPTION} in
                f)
                        FILENAME="${OPTARG}"
```

```
                    ;;
            o)
                    TEE_ARGS=""
                    ;;
            *)
                    echo "Unknown option ${OPTION},
                        ignoring"
                    shift
                    ;; esac
done
shift $((OPTIND -1))


# Gather buffered input (instead of using python -u) to
    allow for Ctl+C
script -qc 'python3 gather.py' | tee ${TEE_ARGS} ${
   FILENAME}

# We don't actually need this lol
rm -rf "typescript"

# Remove the ugly ^C from the logfile
sed -i  's/\^C//g' ${FILENAME}
```

6 Appendix B: The `analysis` program

```
#!/usr/bin/python3
import sys
import datetime
import re
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import numpy

def getlines(filename):
    f = open(filename)
    lines = f.readlines()
    f.close()
```

```python
    return lines

def stamp_stripper(unstripped_stamp):
    res = re.search(r'(?<=reading at ).*',
        unstripped_stamp)
    return res.group(0)

# Get total reading duration
def calculate_duration(start_datetime_str,
    end_datetime_str):
    start_datetime = datetime.datetime.strptime(
        start_datetime_str, "%Y-%m-%d %H:%M:%S.%f")
    end_datetime = datetime.datetime.strptime(
        end_datetime_str, "%Y-%m-%d %H:%M:%S.%f")
    return end_datetime - start_datetime

# paramaters: t(ime), I(nitial), C(ooling coefficient),
    O(ffset)
def exponential_decay(t, I, C, O):
    return I * numpy.exp(-C * t) + O

# If we don't have datafile, we can't do anything
if len(sys.argv) < 2:
    print("usage: analysis.py <datafile>")
    sys.exit(0)


# Get the raw data from the datafile
datafile = sys.argv[1]
lines = getlines(datafile)

# Calculate the duration of the sensor reading
duration = calculate_duration(stamp_stripper(lines[0]),
    stamp_stripper(lines[-1]))

# Turn temperature readings into differences of type
    float
y_data = [nums[1] - nums[0] for nums in [list(map(numpy
```

```
    .float64, s[0:-1].split(' '))) for s in lines
    [1:-1]]]

# Calculate the time interval by dividing the total
    duration by the number of data points
x_range = len(y_data)
x_units = (duration / x_range).total_seconds()
x_data = numpy.array([numpy.float64(i * x_units) for i
    in range(x_range)])

# Perform non-linear least squares curve fitting using
    the scipy library
popt, pcov = curve_fit(exponential_decay, x_data,
    y_data)

# Plot the raw data in blue dashed lines
plt.plot(x_data, y_data, 'b--', label='data')

# Plot the curve we fitted to the data
plt.plot(x_data, exponential_decay(x_data, *popt), 'r-'
    ,
         label='fit: I=%.3E, C=%.3E, O=%.3E' % tuple(
            popt))

plt.xlabel('Time (units = seconds, interval = %.3f)' %
    x_units)
plt.ylabel('Temperature (units = degrees celcius)')
plt.legend()

# The result :)
plt.show()
```

References

Shabnam Siddiqui. Why are differential equations used for expressing the laws of physics?, 2014.

Wikipedia. Newton's law of cooling - wikipedia.

https://en.wikipedia.org/wiki/Newton%27s_law_of_cooling, 2020. (Accessed on 06/25/2020).

Gikfun. Amazon.com: Gikfun ds18b20 temperature sensor waterproof digital thermal probe sensor for arduino (pack of 5pcs) ek1083: Computers & accessories, 2020. (Accessed on 06/25/2020).