

Studying Physics, getting to know Python. RC circuit, simple experiments, coding and data analysis with Raspberry Pi

Submitted to **Computing in Science and Engineering, 2020**

Contents

1	Part list	S-2
2	Experiment zero: turn a LED on and off	S-3
3	Read analog signals: measure a constant voltage	S-4
3.1	Connecting the MCP 3008	S-4
3.2	Reading analog signals	S-5
4	Transient response of the RC circuit	S-6
4.1	Wiring diagram	S-6
4.2	Code	S-6
5	Analysis. Modelling the experimental data	S-8
5.1	Curve fitting: charging process	S-8
5.2	Curve fitting: discharging process	S-11
5.3	Semilog plots	S-15
6	Insights	S-16
6.1	Measurement rate	S-16
6.2	Acquisition based on the <code>spidev</code> module	S-18
6.3	Optimized acquisition rate	S-21

Listings

1	<code>gpio_switching.py</code>	S-3
2	<code>measure_voltage.py</code>	S-5
3	<code>measure_RCtransients.py</code>	S-6
4	<code>plot_voltage.py</code>	S-7
5	<code>curveFit_charging.py</code>	S-8
6	<code>curveFit_discharging.py</code>	S-11
7	<code>plot_voltage_and_fit.py</code>	S-14
8	<code>rate_analysis.py</code>	S-16
9	<code>measure_voltage_SPI.py</code>	S-18
10	<code>histoTimeLapsesCharging.py</code>	S-20
11	<code>log_spacedTimes.py</code>	S-21

1 Part list

- Raspberry Pi3 model B (www.raspberrypi.org)
 - Power supply
 - Micro SD card with Raspbian Operating System (some sellers provide the Operating System already loaded on the micro SD card).
 - (suggested) Case with opening for access to the GPIO contacts
 - Keyboard USB
 - Mouse USB
 - Display with HDMI port (we use a Samsung S24D330 Monitor 24" Full HD, 1920 x 1080)
 - HDMI cable
- Extension cable for the GPIO (Kuman)
- Solderless breadboard (Kuman)
- Jumpers (Kuman)
- LED
- 330 Ω resistor
- MCP 3008 Analog to Digital Converter (on Amazon, DigiKey, Mouser)
- 1 k Ω resistor (3 pcs)
- 10 k Ω resistor
- 100 μ F (electrolytic) capacitor

2 Experiment zero: turn a LED on and off

Lighting a Light Emitting Diode. A $330\,\Omega$ resistor is connected in series to the diode in order to limit the current that flows from the RPi through the LED [1]. One terminal of the series is connected to a ground (GND) pin of the GPIO, the other terminal of the series is connected to the GPIO pin 17, a pin selectable for digital output. The LED has to be correctly polarized, connecting the shorter leg (the cathode) to the lower potential.

An individual GPIO pin can only safely provide 16 mA [2], [3]. If we use a maximum voltage of +3.3 V, according to Ohm's law the minimum resistance connected between the high voltage pin and the ground pin should be $R_{min} = +3.3\text{ V}/16\text{ mA} \simeq 206\,\Omega$. Moreover, the GPIO pins can provide all together a maximum current of 50 mA, distributed on all the pins [2], [4].

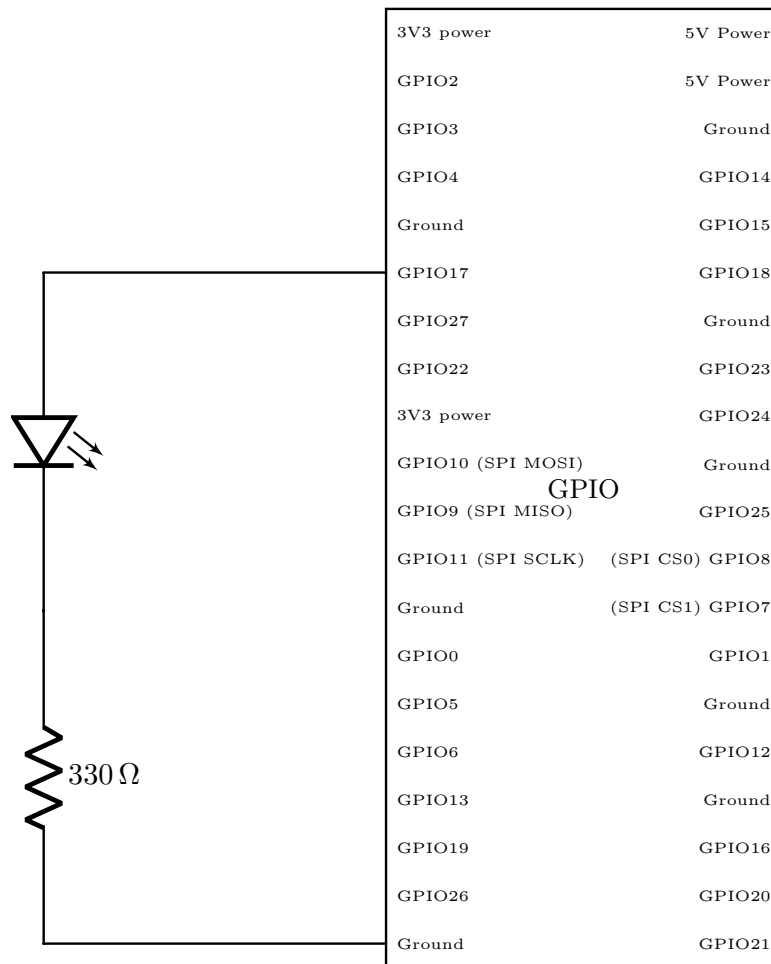


Figure S.1: Experiment Zero.

Listing 1: `gpio_switching.py`

```

1 from gpiozero import LED
2 from time import sleep
3
4 led=LED(17)
5
6 while True:
7     led.on()
8     sleep(0.5)
9     led.off()
10    sleep(0.5)

```

Code from the examples in [5], [6].

3 Read analog signals: measure a constant voltage

3.1 Connecting the MCP 3008

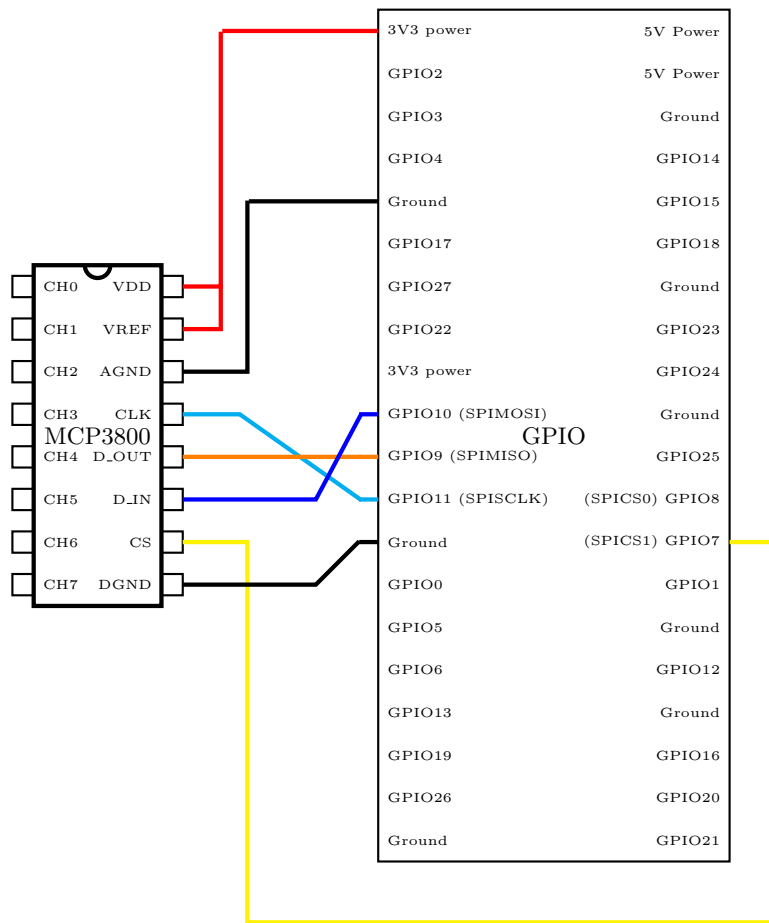


Figure S.2: Reading analog signals. Connections between the Raspberry Pi and the Analog to Digital Converter, the integrated circuit MCP3008.

3.2 Reading analog signals

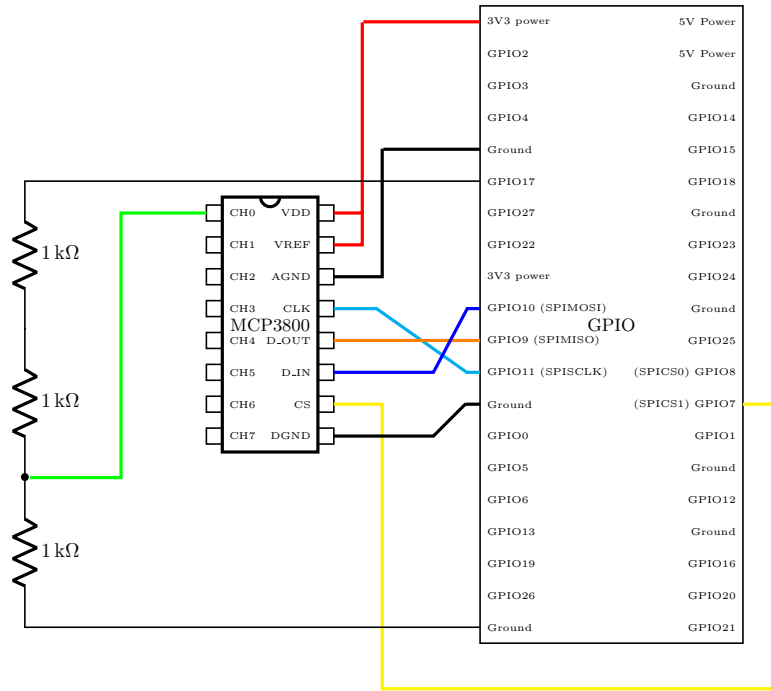


Figure S.3: Measuring on a voltage divider.

Measuring the voltage on a resistor of a voltage divider. We use the pin GPIO17 for applying a 3.3 V tension to the series of three equal resistors, so that a 1.1 V voltage is expected between the two terminals of each resistor. We choose the resistor connected to GND and we connect the other terminal of the resistor to the channel CH0 of the ADC converter.

The following script `measure_voltage.py` allows to read and print the voltage on one of the resistors of the series when the circuit is powered on and off, respectively. Since the values are transmitted from the MCP3008 to the RPi using the SPI protocol (Serial Peripheral Interface), the RPi configuration has to be previously modified in order to activate the SPI support [7], [8]¹.

Listing 2: `measure_voltage.py`

```

1 from gpiozero import LED, MCP3008
2 from time import sleep
3
4 power = LED(17)
5 pot = MCP3008(0)
6 Vdc = 3.3
7
8 while True:
9     power.on()
10    print(pot.value * Vdc)
11    sleep(1)
12    power.off()
13    print(pot.value * Vdc)
14    sleep(1)

```

Code based on the example in [9].

¹For activating the SPI support: Raspberry Menu, Preferences, Raspberry Pi Configuration, Interfaces, SPI: Enable.

4 Transient response of the RC circuit

4.1 Wiring diagram

Measuring the voltage while charging and discharging the capacitor. One end of the RC series is connected to a GND pin of the GPIO, the other end is connected to the pin GPIO17. The maximum current drawn from the RPi to the circuit is $I = V_{DC}/R = 33\mu\text{A}$, well below the recommended limit of 16 mA.

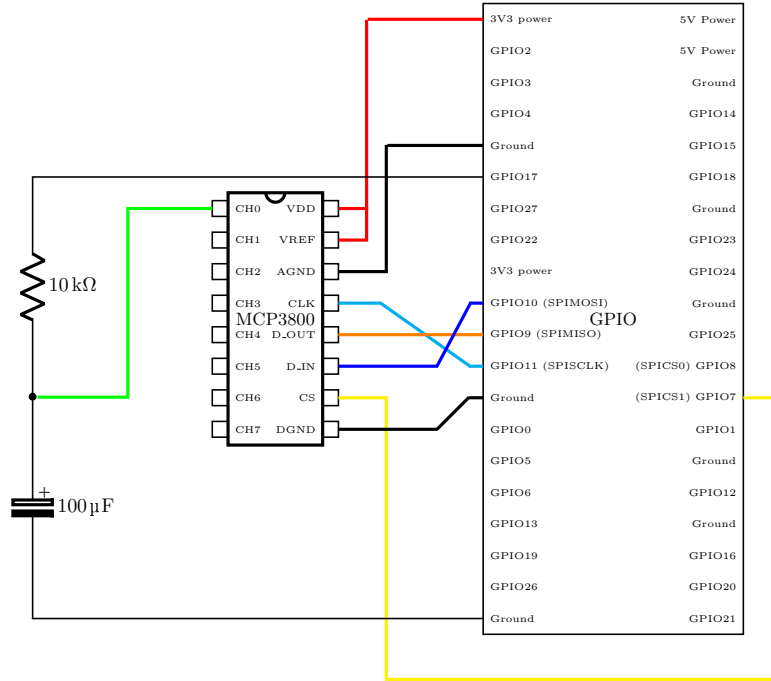


Figure S.4: RC circuit. Wiring diagram.

4.2 Code

Listing 3: measure_RCtransients.py

```

1 from gpiozero import LED, MCP3008
2 from time import sleep, perf_counter
3 import numpy as np
4
5 power = LED(17)
6 pot = MCP3008(0)
7
8 sleep_time = 0.1
9 exec_time = 40
10 v_high_time = 20
11 Vdc = 3.3
12 outputFile='voltageLog.txt'
13
14 def measurement(time_limit):
15     while True:
16         time = perf_counter() - start_time
17         times.append(time)
18         voltage = pot.value * Vdc
19         voltages.append(voltage)
20         if time > time_limit:
21             break
22         sleep(sleep_time)
23
24 times = [ ]

```

```

25 voltages = [ ]
26
27 start_time = perf_counter()
28
29 # charging
30 t_limit = v_high_time
31 power.on()
32 measurement(t_limit)
33
34 # discharging
35 t_limit = exec_time
36 power.off()
37 measurement(t_limit)
38
39 t = np.array(times)
40 v = np.array(voltages)
41
42 np.set_printoptions(precision = 20)
43 np.savetxt(outputFile, np.column_stack((t, v)))

```

Listing 4: plot_voltage.py

```

1 # plot_voltage.py
2
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 # read data from file
7 t, v = np.loadtxt('voltageLog.txt', delimiter=' ', unpack = True)
8
9 # plot of the experimental data
10 plt.plot(t, v, 'o', color='blue', markersize = 3)
11 plt.axhline(color = 'gray', zorder = -1)
12 plt.axvline(color = 'gray', zorder = -1)
13 plt.xlabel('time $t$ (s)')
14 plt.ylabel('voltage $V_C$ (V)')
15 plt.text(27, 1.8, 'R = 10 k$\Omega$'+'\n'+ 'C = 100 $\mu$F')
16
17 plt.savefig('plot_voltage.pdf')
18 plt.show()

```

5 Analysis. Modelling the experimental data

From the original file containing the experimental data we extract two subsets of data: one corresponding to charging, one corresponding to the discharging.

The fitting function cannot reproduce discontinuous changes such those occurring at the time t_0 at which the charging process sets in, and at the time t_1 at which the discharging process starts. For this reason from the dataset concerning the charging process we excluded the early points at about 0.0 V, and from the dataset corresponding to the discharging process we excluded the voltage values equal to the V_{dc} plateau value.

5.1 Curve fitting: charging process

Listing 5: curveFit_charging.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.gridspec as gridspec
4 import scipy.optimize
5
6 res = 3.3 / 1023
7 inputFile = 'chargingProcessData.txt'
8
9 # define fitting function
10 def ChargeProcess(t, V0, t0, tau):
11     return V0 * (1-np.exp(-(t-t0)/tau))
12
13 # read the experimental data from file
14 t, v = np.loadtxt(inputFile, delimiter = ' ', unpack = True)
15
16 # assign the experimental uncertainty on Voltage values
17 dv = np.ones(v.size) * res
18
19 # initial guesses for fitting parameters
20 V0_guess, t0_guess, tau_guess = 3., 0., 1.
21
22 # fitting procedure
23 nlfitt, nlpcov = \
24     scipy.optimize.curve_fit(ChargeProcess, t, v, \
25                             p0 = [V0_guess, t0_guess, tau_guess],
26                             sigma = dv, \
27                             bounds = (0, 100))
28
29 # we apply bounds to the free parameters so that only positive
30 # values are allowed.
31
32 # obtaining parameters from the best fit procedure
33 V0, t0, tau = nlfitt
34
35 # obtaining uncertainties associated with fitting parameters
36 dV0, dt0, dtau = \
37     [np.sqrt(nlpcov[j, j]) for j in range(nlfitt.size)]
38
39 # create fitting function from fitted parameters
40 t_fit = np.linspace(t.min(), t.max(), 128)
41 v_fit = ChargeProcess(t_fit, V0, t0, tau)
42
43 # residuals and reduced chi squared
44 resids = v - ChargeProcess(t, V0, t0, tau)
45 redchisqr = ((resids/dv)**2).sum()/float(t.size-3)
46 ndf = t.size-3
47
48 # where 3 is the number of free parameters
```



```

47 # create figure window to plot data
48 fig = plt.figure(1, figsize = (8,8))
49 gs = gridspec.GridSpec(2, 1, height_ratios = [6, 2])
50
51 # plotting data and fit
52 ax1 = fig.add_subplot(gs[0])
53 ax1.plot(t_fit, v_fit)
54 ax1.errorbar(t, v, yerr = dv, fmt = 'or', ecolor = 'black', markersize =
    2)
55 ax1.set_xlabel(' time (s)')
56 ax1.set_ylabel('voltage $V_{C}$ (V)')
57 ax1.text(0.5, 0.50, r'$V_0$ = {0:6.4f}$\pm${1:0.4f}'.format(V0, dV0),
    transform = ax1.transAxes, fontsize = 14)
58 ax1.text(0.5, 0.40, r'$\tau$ = {0:6.4f}$\pm${1:0.4f}'.format(tau, dtau),
    transform = ax1.transAxes, fontsize=14)
59 ax1.text(0.5, 0.30, r'$t_0$ = {0:5.4f}$\pm${1:0.4f}'.format(t0, dt0),
    transform = ax1.transAxes, fontsize=14)
60 ax1.text(0.5, 0.20, r'$\chi_r^2$ = {0:0.1f}, ndf = {1}'.format(redchisqr,
    ndf), transform = ax1.transAxes, fontsize=14)
61
62 # plotting residuals
63 ax2 = fig.add_subplot(gs[1])
64 ax2.errorbar(t, resids, yerr = dv, ecolor = 'black', fmt = 'ro', markersize
    = 2)
65 ax2.axhline(color = 'gray', zorder = -1)
66 ax2.set_xlabel('time (s)')
67 ax2.set_ylabel('residuals (V)')
68 plt.savefig('ChargingProcessDataAndFit.pdf')
69 plt.show()

```

Code based on the examples in [10].

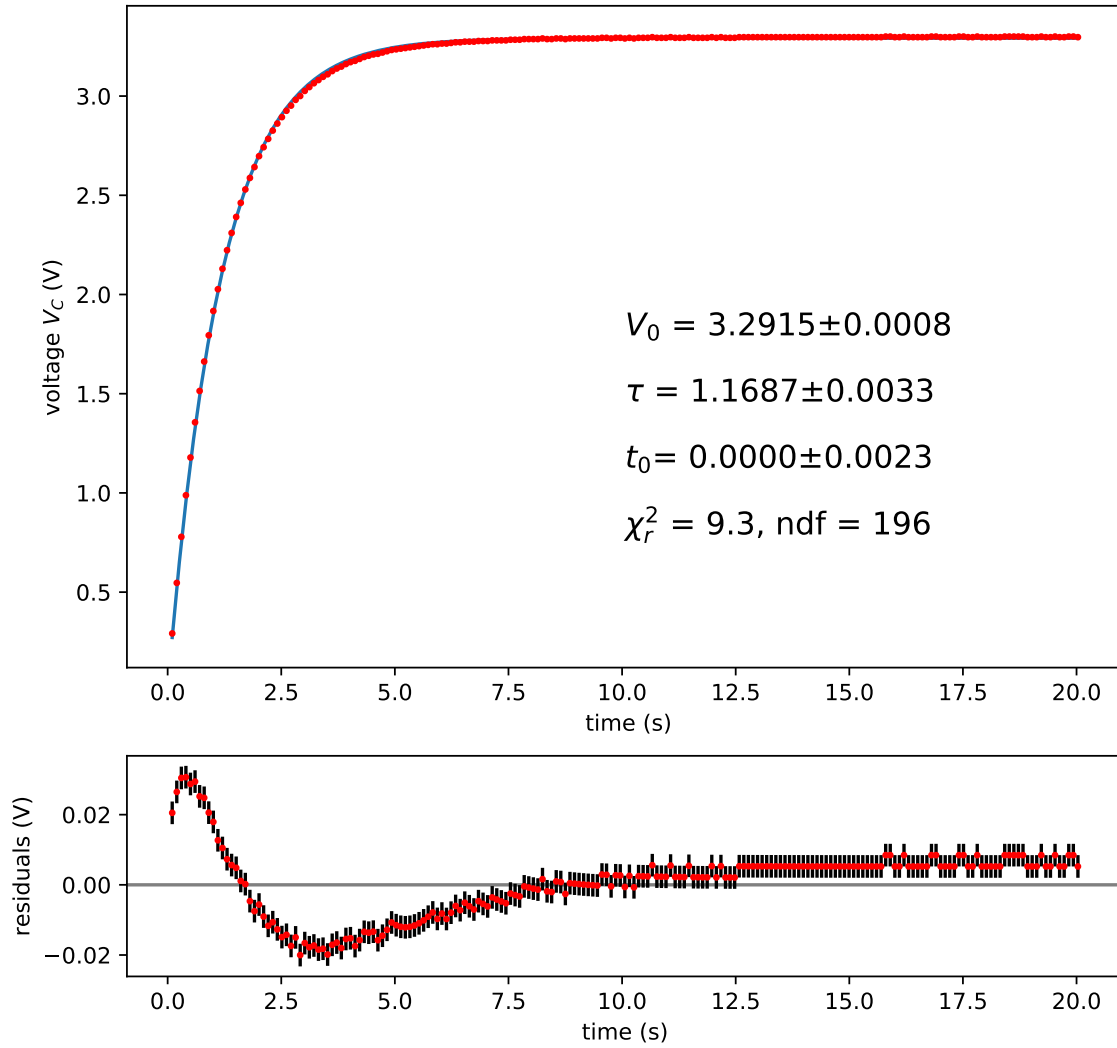


Figure S.5: *Upper panel.* Experimental values (dots) of the voltage on the capacitor in an RC circuit during the charging process measured as a function of time and best fit (solid line). *Lower panel.* Plot of the residuals (experimental value *minus* computed value) versus time. Error bars are plotted assuming that the uncertainty on the voltage measurements is equal to the voltage resolution.

5.2 Curve fitting: discharging process

Listing 6: curveFit_discharging.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.gridspec as gridspec
4 import scipy.optimize
5
6 Vdc = 3.293551538837322212
7 res = 3.3 / 1023
8 inputFile = 'DischargeProcessData.txt'
9
10 # define fitting function
11 def DischargeProcess(t, t0, tau):
12     return Vdc * (np.exp(-(t-t0)/tau))
13
14 # read the experimental data from file
15 t, v = np.loadtxt(inputFile, delimiter = ' ', unpack = True)
16
17 # assign the experimental uncertainty on Voltage values
18 dv = np.ones(v.size) * res
19
20 # initial guesses for fitting parameters
21 t0_guess, tau_guess = 20, 1
22
23 # fitting procedure
24 nlfitt, nlpconv = \
25     scipy.optimize.curve_fit(DischargeProcess, t, v, \
26                             p0 = [t0_guess, tau_guess], sigma = dv, \
27                             bounds = (0, 100))
28
29 # obtaining parameters from the best fit procedure
30 t0, tau = nlfitt
31
32 # obtaining uncertainties associated with fitting parameters
33 dt0, dtau = \
34     [np.sqrt(nlpconv[j, j]) for j in range(nlfitt.size)]
35
36 # create fitting function from fitted parameters
37 t_fit = np.linspace(t.min(), t.max(), 128)
38 v_fit = DischargeProcess(t_fit, t0, tau)
39
40 # residuals and reduced chi squared
41 resids = v - DischargeProcess(t, t0, tau)
42 redchisqr = ((resids/dv)**2).sum()/float(t.size-2)
43 ndf = t.size - 2
44 # where 2 is the number of free parameters
45
46 # create figure window to plot data
47 fig = plt.figure(1, figsize = (8,8))
48 gs = gridspec.GridSpec(2, 1, height_ratios = [6, 2])
49
50 # plotting data and fit
51 ax1 = fig.add_subplot(gs[0])
52 ax1.plot(t_fit, v_fit)
53 ax1.errorbar(t, v, yerr = dv, fmt = 'or', ecolor = 'black', markersize =
    2)
54 ax1.set_xlabel(' time (s)')
55 ax1.set_ylabel('voltage $V_{C}$ (V)')
```

```

56 ax1.text(0.6, 0.80, r'$\tau$ = {0:6.4f}$\pm${1:0.4f}'.format(tau, dtau),
    transform = ax1.transAxes, fontsize = 14)
57 ax1.text(0.6, 0.70, r'$t_0$ = {0:5.4f}$\pm${1:0.4f}'.format(t0, dt0),
    transform = ax1.transAxes, fontsize = 14)
58 ax1.text(0.6, 0.60, r'$\chi_r^2$ = {0:0.1f}, ndf = {1}'.format(redchisqr,
    ndf), transform = ax1.transAxes, fontsize = 14)
59
60 # plotting residuals
61 ax2 = fig.add_subplot(gs[1])
62 ax2.errorbar(t, resids, yerr = dv, ecolor = 'black', fmt = 'ro', markersize
    = 2)
63 ax2.axhline(color = 'gray', zorder = -1)
64 ax2.set_xlabel('time (s)')
65 ax2.set_ylabel('residuals (V)')
66 plt.savefig('DischargingDataAndFit.pdf')
67 plt.show()

```

Code based on the examples in [10].

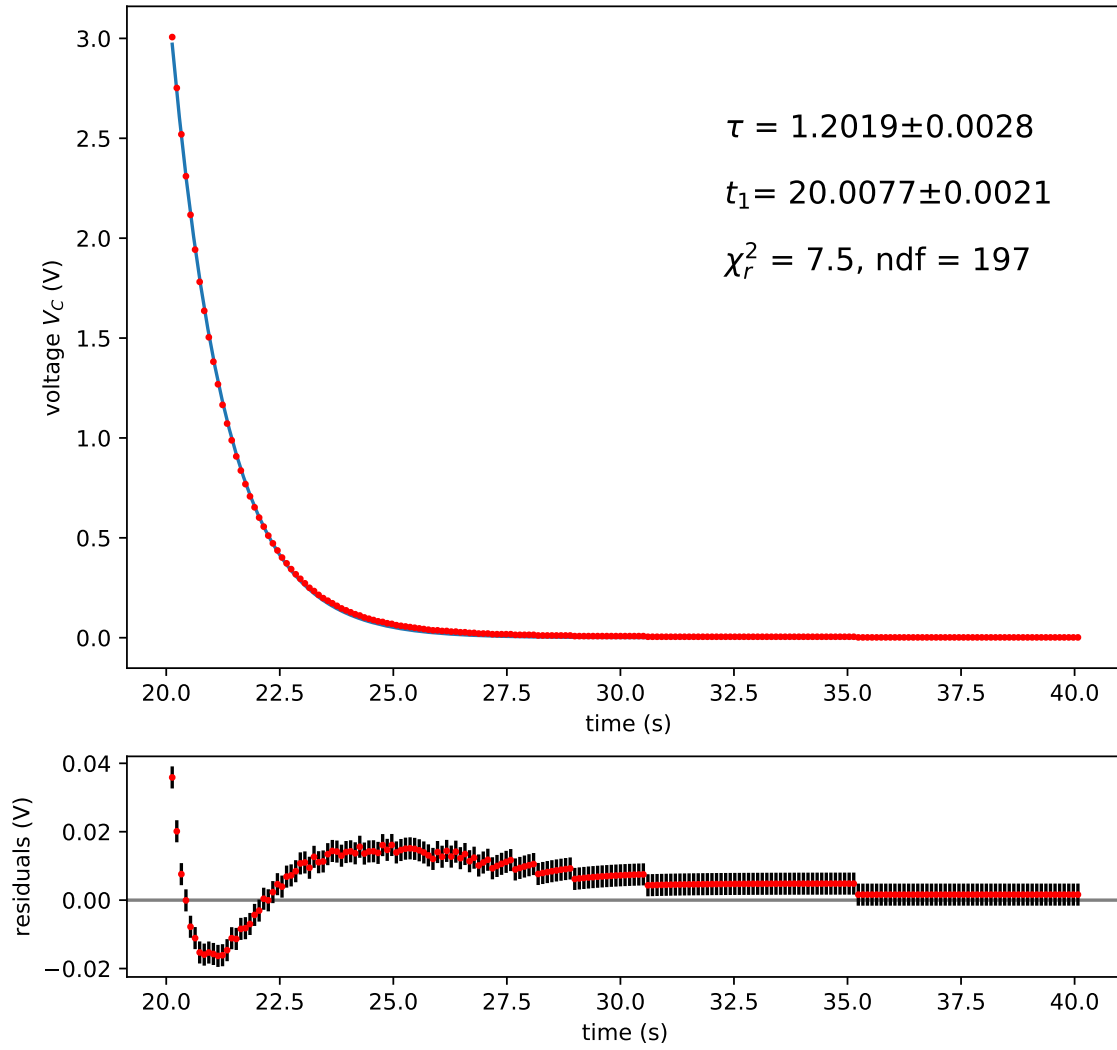


Figure S.6: *Upper panel.* Voltage on the capacitor in an RC circuit during the discharging. The solid line represents the best fit of the model equation $V_C(t) = V_0 \exp[-(t - t_1)/\tau]$ to the experimental data. *Lower panel.* Plot of the residuals.

Listing 7: plot_voltage_and_fit.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # define fitting function
5 def ChargeProcess(t, V0, t0, tau):
6     return V0 * (1-np.exp(-(t-t0)/tau))
7
8 # define fitting function
9 def DischargeProcess(t, V0, t0, tau):
10     return V0 * (np.exp(-(t-t0)/tau))
11
12
13 # read data from file
14 t, v = np.loadtxt('voltageLog.txt', delimiter=' ', unpack = True)
15
16 # plot of the experimental data
17 plt.plot(t, v, 'o', color='blue', markersize = 3)
18 plt.axhline(color = 'gray', zorder = -1)
19 plt.axvline(color = 'gray', zorder = -1)
20 plt.xlabel('time $t$ (s)')
21 plt.ylabel('voltage $V_C$ (V)')
22 plt.text(27, 1.8, 'R = 10 k$\Omega$'+'\n'+ 'C = 100 $\mu$F')
23
24 # model curve for the charge process
25 t_fit_charge = np.linspace(0, 20, 128)
26 V0, t0, tau = 3.2915, 0, 1.1687
27 v_fit_charge = ChargeProcess(t_fit_charge, V0, t0, tau)
28 plt.plot(t_fit_charge, v_fit_charge, color='green')
29 plt.draw()
30
31
32 # model curve for the discharge process
33 t_fit_discharge = np.linspace(20.01, 40, 128)
34 V0, t0, tau = 3.3, 20.0053, 1.2019
35 v_fit_discharge = DischargeProcess(t_fit_discharge, V0, t0, tau)
36 plt.plot(t_fit_discharge, v_fit_discharge, color='green')
37 plt.draw()
38
39
40 plt.savefig('plot_voltage_and_fit.pdf')
41 plt.draw()
42 plt.show()

```

5.3 Semilog plots

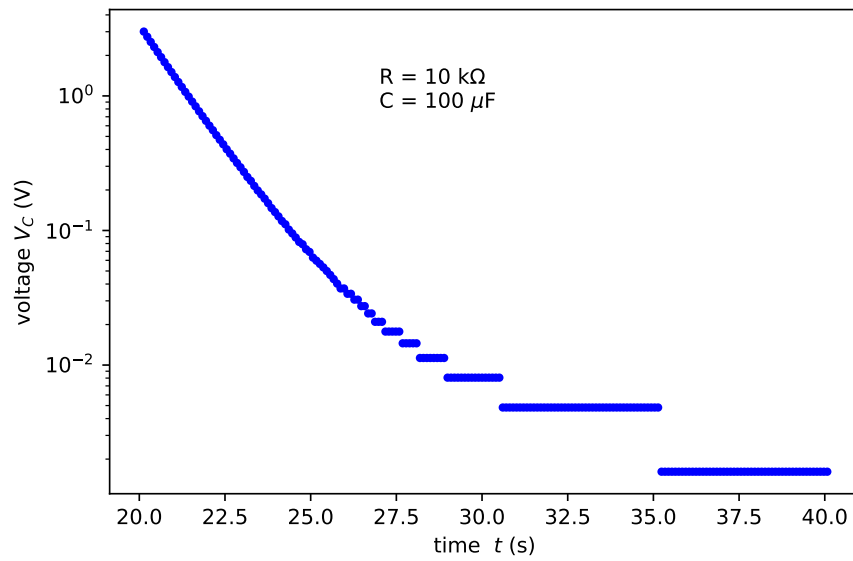


Figure S.7: Semilogarithmic plot of the voltage on the capacitor in the RC circuit studied while discharging the capacitor.

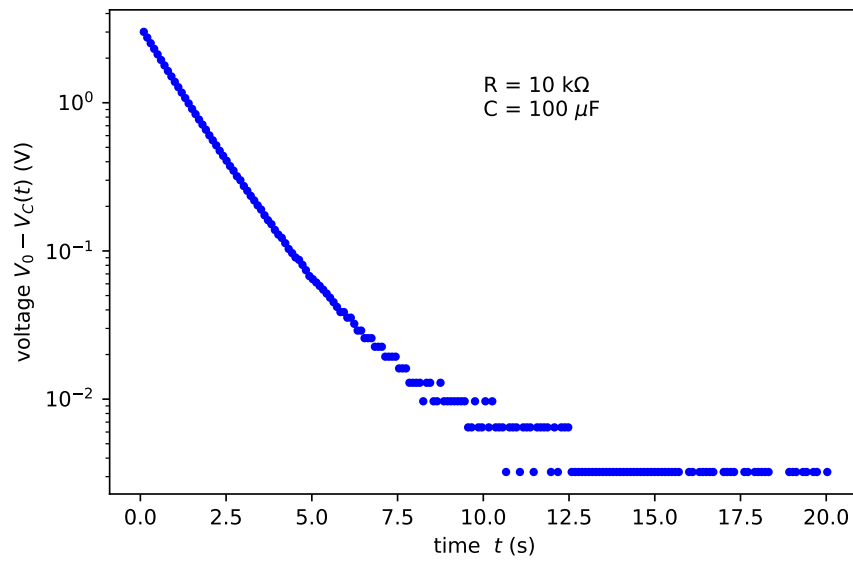


Figure S.8: Semilogarithmic plot of the difference $[V_0 - V_C(t)]$ versus time during the charging of the capacitor in the RC circuit studied.

6 Insights

6.1 Measurement rate

Listing 8: rate_analysis.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 inputDataFile = 'ChargeProcessDataZeroSleepOneSecond.txt'
5 outputDataFile = 'timeLapsesChargeProcessOneSecond.txt'
6 outParFile = 'ChargeDeltatParamsOneSecond.txt'
7
8 # read data from the file
9 t = np.loadtxt(inputDataFile, delimiter = ' ', usecols=(0), unpack = True)
10
11 # compute time lapses
12 delta_t = np.diff(t)
13 t_prime = t[:-1] + (delta_t/2)
14
15 # save the results as text
16 np.set_printoptions(precision = 20)
17 np.savetxt(outputDataFile, np.column_stack((t_prime, delta_t)))
18
19 # statistical analysis on time lapses
20 dt_mean = np.mean(delta_t, dtype = np.float64)
21 dt_min = np.min(delta_t)
22 dt_max = np.max(delta_t)
23 dt_std = np.std(delta_t, dtype = np.float64)
24
25 print('Delta t mean value = ', dt_mean)
26 print('Delta t minimum = ', dt_min)
27 print('Delta t mean value = ', dt_mean)
28 print('Delta t maximum = ', dt_max)
29
30 # write the results of the statistical analysis to a text file
31 fout = open(outParFile, 'w')
32 fout.write('Statistical analysis on times from the file: \n')
33 fout.write(outputDataFile + '\n' + '\n')
34 fout.write('Delta_t mean value = ' + str(dt_mean) + '\n')
35 fout.write('Delta_t minimum = ' + str(dt_min) + '\n')
36 fout.write('Delta_t maximum = ' + str(dt_max) + '\n')
37 fout.write('Delta_t standard deviation = ' + str(dt_std))
38 fout.close()
39
40
41 # plot of Delta_t vs- t
42 plt.plot(t_prime, delta_t*1e6, 'o', color = 'blue', markersize = 1)
43 plt.xlabel('time $t$ (s)')
44 plt.ylabel('time lapses $\Delta t$ ($\mu s$)')
45 plt.tight_layout()
46 plt.ylim(140, 360)
47 # save the plot to a file
48 plt.savefig('rate_analysis_zeroSleepOneSecond.pdf')
49 #plt.draw()
50 plt.show()
```


The statistical analysis of a set of results, chosen as example, suggests that the minimum time lapse is of $161\text{ }\mu\text{s}$, the average value is $167\text{ }\mu\text{s}$ with a standard deviation of $9\text{ }\mu\text{s}$. The values of the time lapses as a function of time are shown as a scatter plot in the following figure.

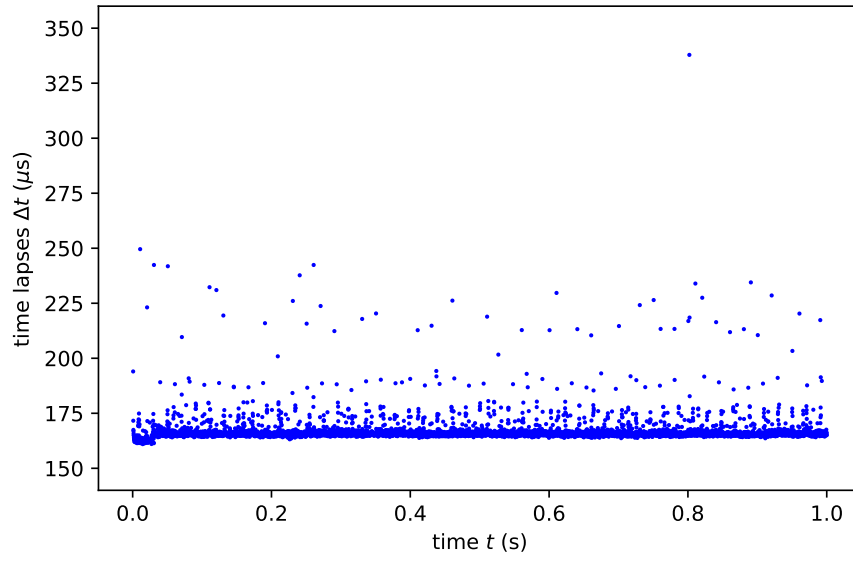


Figure S.9: Time lapses between two subsequent voltage measurements obtained if the code `measure_RCtransients.py` is modified commenting the `sleep()` command. In this condition the mean value of time lapses is approximately $170\text{ }\mu\text{s}$.

6.2 Acquisition based on the spidev module

Listing 9: `measure_voltage_SPI.py`

```
1 from gpiozero import LED
2 from time import sleep, perf_counter
3 import spidev
4 import numpy as np
5
6 power = LED(17)
7
8 # open SPI bus
9 spi = spidev.SpiDev()
10 spi.open(0,0)
11 spi.max_speed_hz = 1000000
12
13 # read the data corresponding to a selected channel
14 # of the ADC MCP3008
15 def ReadChannel(channel):
16     adc = spi.xfer2([1, (8 + channel)<<4, 0])
17     data = ((adc[1]&3) << 8) + adc[2]
18     return data
19
20 # convert data to voltage
21 def ConvertVolts(data):
22     volts = (data * 3.3)/float(1023)
23     return volts
24
25 # define channel
26 voltage_channel = 0
27
28 # function to perform the measurements
29 def measurement(time_limit):
30     while True:
31         # read the time
32         time = perf_counter() - start_time
33         times.append(time)
34         # read the voltage
35         voltage_level = ReadChannel(voltage_channel)
36         voltage_value = ConvertVolts(voltage_level)
37         voltages.append(voltage_value)
38         if time > time_limit:
39             break
40         sleep(sleep_time)
41
42
43 sleep_time = 0.1
44 exec_time = 40
45 v_high_time = 20
46 outputFile = 'VoltageSPI-Log.txt'
47
48 times = [ ]
49 voltages = [ ]
50
51 start_time = perf_counter()
52
53 # charging the capacitor
54 t_limit = v_high_time
55 power.on()
56 measurement(t_limit)
57
```

```

58 # discharging the capacitor
59 t_limit = exec_time
60 power.off()
61 measurement(t_limit)
62
63
64 t = np.array(times)
65 v = np.array(voltages)
66
67 np.set_printoptions(precision = 20)
68 np.savetxt(outputFile, np.column_stack((t, v)))

```

Code based on the example in [7].

The statistic analysis on the time lapses from a sample data set indicates that the minimum time lapse is of about $50\ \mu\text{s}$, the average value is $54\ \mu\text{s}$, and the standard deviation is $12\ \mu\text{s}$. See the scatter plot of the data in Fig.S.10 as well as an histogram of the time lapses concerning a larger data set in Fig.S.11. For comparison, considering that the MCP3008 could work at an acquisition rate of 75 ksps (when $V_{DC} = 2.7\ \text{V}$) [11], this would correspond to time lapses of $13.3\ \mu\text{s}$.

Summarizing, the acquisition rate depends not only on the sampling rate of the ADC, but also on the software used for controlling the MCP3008 chip through the GPIO. Aiming to achieve better performances is convenient to exploit the **spidev** library rather than the **gpiozero** module. Comparative tests on all the experiments carried out using the two different modules can be instructive. In particular, it can be verified whether the minimum detectable voltage corresponds to zero. Moreover, it should be considered that Python on RPi runs under a multitasking operating system so that other processes may gain priority over the CPU while the acquisition managed through the Python code would be slowed down. Faster execution and better timing could be achieved using C codes [12], [13]. However, Python is simpler if compared to C, it is readily available on the RPi, and the acquisition rate achieved with the settings described in this work can be suitable to carry out educational projects.

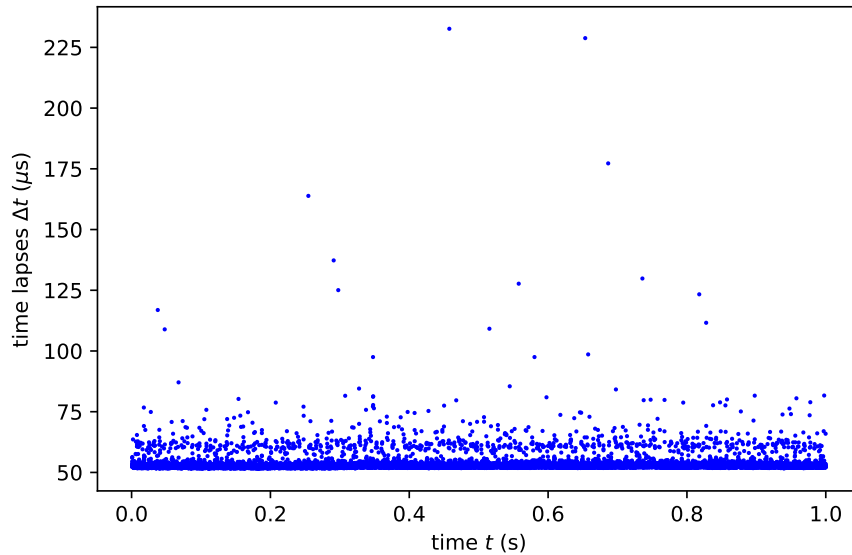


Figure S.10: Time lapses between two subsequent voltage measurements obtained when the code `measure_voltage_SPI.py` is used for the experiment, commenting the `sleep()` instruction. The mean value of the time lapses is of about $55\ \mu\text{s}$. The distribution of the time lapses acquired during a longer time interval is shown in Fig.S.11 as a histogram.

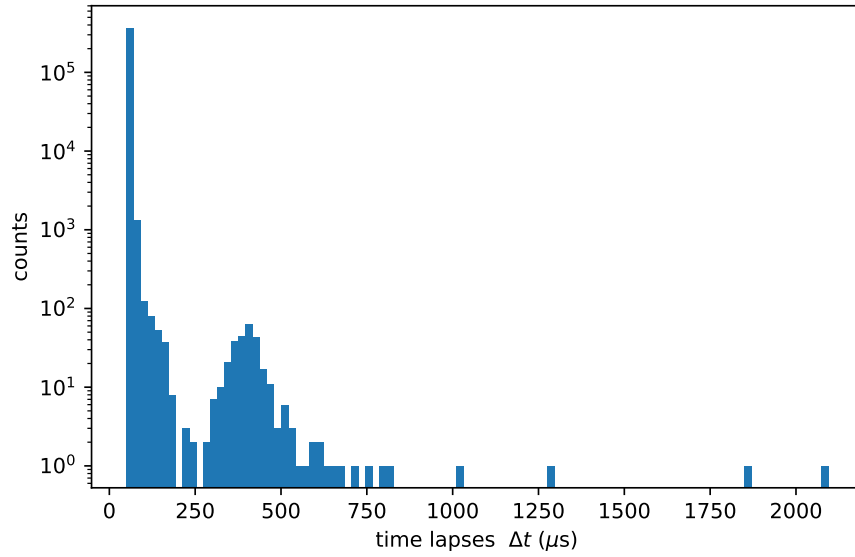


Figure S.11: Distribution of the time lapses Δt during the voltage measurements performed with the code `measure_voltage_SPI.py` where the `sleep()` instruction is commented (371737 data points, 100 bins). The data are acquired in a time interval of 20s.

Listing 10: `histoTimeLapsesCharging.py`

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 DataFile='timeLapsesChargeProcess.txt'
5 OutputFile='histoLapsesCharging.pdf'
6 bins=100
7
8 # read data from file
9 t, Delta_t = np.loadtxt(DataFile, delimiter=' ', unpack = True)
10 # plot
11 plt.hist(Delta_t*1e6, bins, log = True)
12 plt.xlabel('time lapses  $\Delta t$  ( $\mu s$ )')
13 plt.ylabel('counts')
14 plt.tight_layout()
15 plt.savefig(OutputFile)
16 plt.show()

```

6.3 Optimized acquisition rate

Since the voltage varies rapidly when charging and discharging begin, it can be useful to impose a faster acquisition rate during the early steps of the process. To this aim the following code can be used for the measurements.

Listing 11: `log_spacedTimes.py`

```
1 from gpiozero import LED
2 from time import sleep, perf_counter
3 import spidev
4 import numpy as np
5
6 power = LED(17)
7
8 # open SPI bus
9 spi = spidev.SpiDev()
10 spi.open(0,0)
11 spi.max_speed_hz = 1000000
12
13 # read the data corresponding to a selected channel
14 # of the ADC MCP3008
15 def ReadChannel(channel):
16     adc = spi.xfer2([1, (8 + channel)<<4, 0])
17     data = ((adc[1]&3) << 8) + adc[2]
18     return data
19
20 # convert data to voltage
21 def ConvertVolts(data):
22     volts = (data * 3.3)/float(1023)
23     return volts
24
25 # define channel
26 voltage_channel = 0
27
28 # function to perform the measurements
29 def measurement():
30     for time_lapse in lapses:
31         # read the time
32         time = perf_counter() - start_time
33         times.append(time)
34         # read the voltage
35         voltage_level = ReadChannel(voltage_channel)
36         voltage_value = ConvertVolts(voltage_level)
37         voltages.append(voltage_value)
38         #
39         sleep(time_lapse)
40
41
42 times = [ ]
43 voltages = [ ]
44
45 outputFile = 'VoltageSPI-LogSpaced.txt'
46 Delta_t_min = 50e-6
47 v_high_time = 20
48 points = 1000
49
50 start = np.log10(Delta_t_min)
51 stop = np.log10(v_high_time)
52 spots = np.logspace(start, stop, points)
53 lapses = np.diff(spots)
54
```

```
55 start_time = perf_counter()
56
57 # charging the capacitor
58 power.on()
59 measurement()
60
61 # discharging the capacitor
62 power.off()
63 measurement()
64
65
66 t = np.array(times)
67 v = np.array(voltages)
68
69 np.set_printoptions(precision = 20)
70 np.savetxt(outputFile, np.column_stack((t, v)))
```

References and Notes

- [1] (), [Online]. Available: <https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins>.
- [2] (), [Online]. Available: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/>.
- [3] (), [Online]. Available: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/>.
- [4] (), [Online]. Available: www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/gpio_pads_control.md.
- [5] B. Nuttall. (), [Online]. Available: <https://gpiozero.readthedocs.io/en/stable/index.html>. See also <https://www.raspberrypi.org/blog/gpio-zero-v1-5/>.
- [6] (), [Online]. Available: <https://projects.raspberrypi.org/en/projects/physical-computing/3>.
- [7] (), [Online]. Available: <https://www.raspberrypi-spy.co.uk/2013/10/analogue-sensors-on-the-raspberry-pi-using-an-mcp3008/>.
- [8] (), [Online]. Available: <https://gpiozero.readthedocs.io/en/stable/api%5Cspi.html>.
- [9] (), [Online]. Available: <https://projects.raspberrypi.org/en/projects/physical-computing/13>.
- [10] D. J. Pine, *Introduction to Python for Science and Engineering*. CRC Press, 2018, ISBN: 9781138583894.
- [11] MCP3004/3008 datasheets, 2008, Microchip Technology Inc.
- [12] (), [Online]. Available: <https://pypi.org/project/RPi.GPIO/>.
- [13] (), [Online]. Available: <https://www.circuits.dk/everything-about-raspberry-gpio/>.