Modular Integration of Python Programming in Undergraduate Physical Chemistry Experiments
Supporting Information

Derri J. Hughes[1] and Samuel C. Perry[1]

[1]School of Chemistry and Chemical Engineering, University of Southampton, University Road, Highfield, Southampton, SO17 1BJ, United Kingdom

1 Chemical Hazards

Colorimetry of Dyes

- **Allura Red food dye**
  - *Hazards:* None
  - *Risk Mitigation:* Wear standard PPE
  - *Disposal:* Pour down sink with copious amounts of water
- **Deionised Water**
  - *Hazards:* None
  - *Risk Mitigation:* Wear standard PPE
  - *Disposal:* Pour down sink with copious amounts of water

Iodine Clock Reaction

- **H_2SO_4 (2M, 40 mL)**
  - *Hazards:* Corrosive to metals. Skin and serious eye irritant.
  - *Risk Mitigation:* Use in a fume cupboard or ventilated area. Avoid inhalation. Wear standard PPE.
  - *Disposal:* Pour down sink with copious amounts of water.
- **KI (solid, 1 g)**
  - *Hazards:* Harmful if swallowed. Skin and serious eye irritant.
  - *Risk Mitigation:* Wear standard PPE. Ensure adequate ventilation.
  - *Disposal:* Small quantities can be poured down sink with water.
- **H_2O_2 (aq, 6% vol., 8 mL)**
  - *Hazards:* Harmful if swallowed. Causes serious eye damage.
  - *Risk Mitigation:* Avoid inhalation. Wear standard PPE.
  - *Disposal:* Pour down sink with copious amounts of water.

- **Sodium Thiosulfate (0.5 M, 40 mL) and solid (2 g)**
  - *Hazards:* Eye and skin irritant. May evolve sulphurous fumes with other chemicals.
  - *Risk Mitigation:* Use in ventilated area. Wear standard PPE.
  - *Disposal:* Pour down sink with copious amounts of water.
- **Reaction Solution (250 mL)**
  - *Hazards:* Toxic and irritant to skin, eyes, and internally if ingested.
  - *Risk Mitigation:* Wear gloves and standard PPE. Use fume hood if coloured. Add **solid** sodium thiosulfate until clear.
  - *Disposal:*
    - Clear solutions: Down sink with copious amounts of water.
    - If cloudy white/yellow after thiosulfate: Dispose via fume hood sink with copious water.

Particle in a 1D Box

- **Methanol (10 mL)**
  - *Hazards:* Highly flammable. Toxic via ingestion, inhalation, or skin contact. May damage eyes/CNS.
  - *Risk Mitigation:* Use in ventilated area. Keep away from heat. Wear PPE. Avoid inhalation.
  - *Disposal:* Pour down sink with copious amounts of water.

Flash Photolysis of Benzophenone

- **Benzophenone (0.5 g)**
  - *Hazards:* Possible carcinogen. May damage liver/kidneys with prolonged ingestion. Harmful to aquatic life.
  - *Risk Mitigation:* Avoid breathing dust. Use in ventilated area. Wear PPE. Prevent environmental release.
  - *Disposal:* Store in hazardous waste container.
- **NaOH (0.5 M, 100 mL)**
  - *Hazards:* Corrosive. Causes severe burns and eye damage.
  - *Risk Mitigation:* Avoid dust. Wear PPE.
  - *Disposal:* Neutralise with acetic acid. Dispose down sink with water.
- **Isopropyl Alcohol (300 mL)**
  - *Hazards:* Highly flammable. Eye irritant. May cause dizziness.
  - *Risk Mitigation:* Avoid vapour. Wear PPE. Keep away from ignition sources.
  - *Disposal:* Store in hazardous waste container.

2 Programming Skills

Table S1: Outline of programming skills that are developed over the course of each practical. Y1 refers to a first year lab practical and Y2 a second year lab practical. *Skills are optional as discussed in the main text.

Experiment	Previously Learned Skills Used	New Python Skills Learned
Introduction to Python (Y1)	None	Basic input/output (1), data types (variables, lists, floats, integers, strings) (2), conditional statements (if, elif, else) (3), arithmetic operations (4), while loops (5), plotting with matplotlib (6), linear regression (7), data extraction (8)
Kinetics of Nuclear Decay (Y1)	(1), (2), (4), (6), (8)	For loops (9), mathematical functions (absolute values) (10), mathematical functions (exponentials) (11), list indexing (12), generating model data (13), appending data to a list (14)
Basic Simulations of Nuclear Fusion (Y1)	(1), (2), (4), (6), (8), (9), (12), (13), (14)	Writing simulations (15), simulating ordinary differential equations (16)
Colorimetry of Dyes (Y1)	(1), (2), (3), (4), (5), (6), (7), (8), (9), (12), (14)	Arduino interfacing with Python (17), real-time data acquisition (18), mathematical functions (`log10`) (19), error handling (try-catch blocks) (20), Boolean operators (21), iterative data analysis (22)
Iodine Clock Reaction (Y1)	(1), (2), (3), (4), (5), (6), (7), (8), (9), (12), (14), (17), (18), (20), (21), (22)	Data thresholding (23), averaging over windowed data (24), sound generation (`winsound`) (25)
Particle in a 1D Box (Y2)	(1), (2), (4), (6), (8), (9), (12), (13), (14), (22)	File I/O (26), list slicing (27), mathematical functions (trigonometric functions) (28)
Flash Photolysis of Benzophenone (Y2)	(1), (2), (4), (6), (8), (12), (14), (22), (26), (27)	Curve (non-linear) fitting (29), statistical analysis (30), defining functions (31), GUIs (32)*, object-oriented programming (33)*

3 Introduction to Python

Algorithm 1 Calculator to convert between °C and K

$a \leftarrow$ input "Enter 'C' to convert to Celsius or 'K' to convert to Kelvin"
if $a =$ "K" **then**
  $C \leftarrow$ input "Enter the value to be converted to Kelvin"
  $K \leftarrow C + 273.15$
  Output: "C °C = K K"
else if $a =$ "C" **then**
  $K \leftarrow$ input "Enter the value to be converted to Celsius"
  $C \leftarrow K - 273.15$
  Output: "K K = C °C"
end if

Algorithm 2 Find activation energy

Require: `oscillation_data`: list of rate constants, `temperature`: corresponding temperature values list
Plot `oscillation_data` versus `temperature` as a scatter plot
$(m, c) \leftarrow$ `numpy.polyfit(temperature, oscillation_data, 1)` {Linear regression: gradient and intercept}
Plot linear fit as line
$E_a \leftarrow -8.314 \times m$
Output: "Activation energy = E_a kJ mol^{-1}" and linear plot.

4 Creating Kinetic Models and Simulations

4.1 Kinetics of Nuclear Decay

Algorithm 3 Find Half Life

Require: `decay_data`: array of concentrations, `time`: array of time values
1: `conc_half` ← max(`decay_data`)/2 {Calculate half the intial concentration}
2: Initialise `conc_diff` as an empty list
3: **for** each c in `decay_data` **do**
4: `diff` ← $|c -$ `conc_half`$|$
5: Append `diff` to `conc_diff`
6: **end for**
7: `idx` ← index of minimum value in `conc_diff` {Closest value to 0}
8: `half_life` ← `time`[`idx`]
9: **Output:** "Half-life of isotope = `half_life` seconds"

Algorithm 4 Generate and Plot ^{15}O Model Data

1: $d \leftarrow 0.00568$ {Decay constant in s^{-1}}
2: `O15_initial` ← 8372146872633.971
3: Convert `time` to numpy array
4: `decay_data` ← `O15_initial` $\cdot \exp(-d \cdot$ `time`$)$
5: Plot `decay_data` against `time` with ^{18}F data
6: Find the half life using the same method as Algorithm 3

4.2 Basic Simulations of Nuclear Fusion

4.2.1 Derivation of Model - Student Instructions

The relevant expressions (needed to program a simulation of the system) will be derived for the following reaction mechanism (that comprises three elementary steps):

Reaction 1: $\mathrm{B} \rightarrow 2\mathrm{O} \quad (k_1)$
Reaction 2: $\mathrm{R} + \mathrm{O} \rightarrow 2\mathrm{G} \quad (k_2)$

1. Write down the reaction rate for each of the elementary reaction steps.

$$\begin{aligned} \text{Rate of 1st reaction:} \quad & R_1 = k_1[\mathrm{B}] \\ \text{Rate of 2nd reaction:} \quad & R_2 = k_2[\mathrm{R}][\mathrm{O}] \end{aligned}$$

2. Write the overall rate expressions for the species of interest, in terms of the expressions derived in Step 1:

$$\begin{aligned} \frac{d[\mathrm{B}]}{dt} &= -R_1 \quad \text{(1 mole of B is lost in the 1st reaction)} \\ \frac{d[\mathrm{O}]}{dt} &= 2R_1 - R_2 \quad \text{(2 moles of O produced in 1st, 1 lost in 2nd)} \\ \frac{d[\mathrm{R}]}{dt} &= -R_2 \quad \text{(1 mole of R is lost in the 2nd reaction)} \\ \frac{d[\mathrm{G}]}{dt} &= 2R_2 \quad \text{(2 moles of G are produced in the 2nd reaction)} \end{aligned}$$

3. Rewrite the overall rate expressions as numerical integrals and re-arrange to isolate Δ[Species]:

$$\frac{\Delta[\mathrm{B}]}{\Delta t} = -R_1 \Rightarrow \Delta[\mathrm{B}] = -R_1 \Delta t$$
$$\frac{\Delta[\mathrm{O}]}{\Delta t} = 2R_1 - R_2 \Rightarrow \Delta[\mathrm{O}] = 2R_1 \Delta t - R_2 \Delta t$$
$$\frac{\Delta[\mathrm{R}]}{\Delta t} = -R_2 \Rightarrow \Delta[\mathrm{R}] = -R_2 \Delta t$$
$$\frac{\Delta[\mathrm{G}]}{\Delta t} = 2R_2 \Rightarrow \Delta[\mathrm{G}] = 2R_2 \Delta t$$

Note: Providing Δt is small compared to the total reaction time, $\Delta[Species]/\Delta t$ approximates $d[Species]/dt$.

4. Rewrite the overall rate expressions as (indexed) numerical integrals and re-arrange to isolate $[\mathrm{Species}]_i$:

Note: $\Delta[Species] = [Species]_i - [Species]_{i-1}$ and $\Delta t = t_i - t_{i-1}$

$$[\mathrm{B}]_i = [\mathrm{B}]_{i-1} - R_{1,i-1}(t_i - t_{i-1})$$
$$[\mathrm{O}]_i = [\mathrm{O}]_{i-1} + 2R_{1,i-1}(t_i - t_{i-1}) - R_{2,i-1}(t_i - t_{i-1})$$
$$[\mathrm{R}]_i = [\mathrm{R}]_{i-1} - R_{2,i-1}(t_i - t_{i-1})$$
$$[\mathrm{G}]_i = [\mathrm{G}]_{i-1} + 2R_{2,i-1}(t_i - t_{i-1})$$

5. Rewrite the Step 1 expressions, incorporating the indexing:

$$R_{1,i-1} = k_1[\mathrm{B}]_{i-1}$$
$$R_{2,i-1} = k_2[\mathrm{R}]_{i-1}[\mathrm{O}]_{i-1}$$

If the initial conditions were known, the equations from Steps 4 and 5 could then be used to write a program to simulate the reaction.

Algorithm 5 Simulation of Stellar Nuclear Fusion

1: $k_1 \leftarrow 0.0125$, $k_2 \leftarrow 0.025$
2: Initialise $D \leftarrow [10]$, $T \leftarrow [0]$, $He \leftarrow [0]$, $t \leftarrow [0]$
3: **for** $i = 1$ to 50 **do**
4: $\quad t_i \leftarrow t_{i-1} + 2$
5: $\quad$ Calculate R_1, R_2
6: $\quad$ Update and append D_i, T_i, He_i
7: $\quad$ Append t_i to t
8: **end for**
9: Plot D, T, and He vs. t

5 Data Acquisition with an Arduino

5.1 Arduino Components

All Arduino components can be sourced from any reputable electronics supplier. Manufacturer product codes are given where possible for ease of searching. Product codes are intended to help find a product similar to those in use at Southampton, and are not intended to be a recommendation for a specific product or supplier.

- Arduino Uno Board (Part No: A000066)
- RGB light emitting diode (LED) (Part No: L-154A4SUREQBFZGEC)
- Light sensor (Part No: OPL551)
- Breadboard (Part No: 2444)
- 40 Ω, 50 Ω and 100 Ω resistors (Part No: YR1B40R2CC, 45F50RE, 45F100E)
- 4-pin push button switch (Part No: 1301.9302)
- Jumper wires (Part No: MIKROE-513)

Our Arduino-based equipment is often supported by a LEGO architecture, supported by an assembly of LEGO bricks on a 32x32 baseboard.

5.2 Wiring Diagram and Code Repository

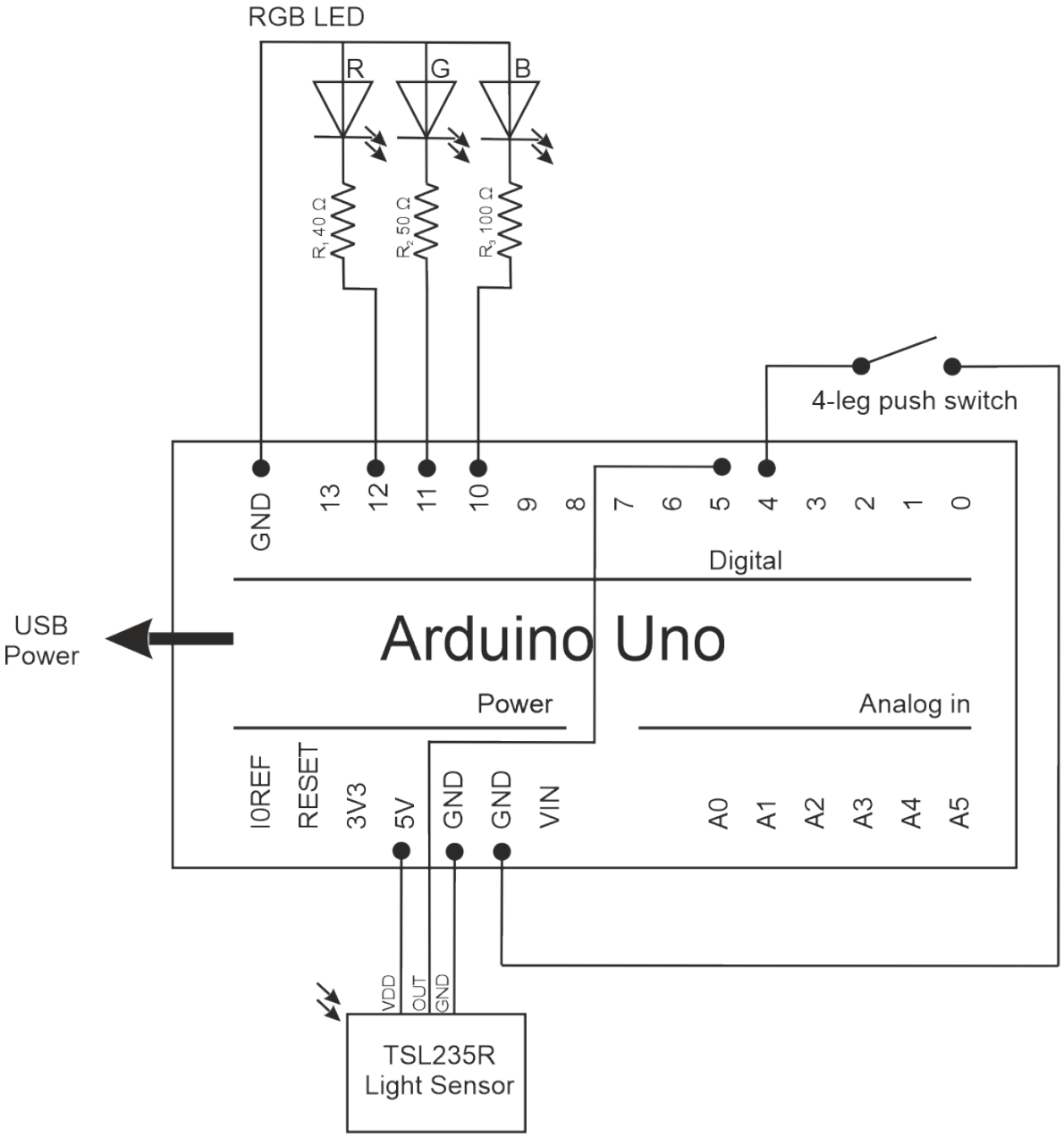

Figure S1: Arduino wiring diagram for the light sensor setups described in section 4 of the main text. Note: the setup shown is for the Colorimetry of Dyes practical. To adjust the setup for the Iodine Clock practical, simply do not include the RGB LED.

The C++ microcontroller codes and example Python data acquisition codes for the Arduino UNO board is available freely on GitHub. [1] Here, we present a summary of the repository's content.

Table S2: Table of contents for the Arduino codes GitHub repository. [1]

Contents	**Description**
ArduinoConfig.cpp	Arduino sketch that measures the frequency of an incoming 5V TTL signal on pin 5 using the `FreqCounter` library. It supports mode selection via a button (pin 4) or serial input, and sets output pins 10–12 HIGH according to the active mode (0–3). The device prints the measured frequency to the serial port every 120 ms.
README.md	Full documentation of the project, including hardware setup, pin configuration, serial interface commands, upload instructions via Arduino IDE, and operational notes. Includes detailed description of the application's purpose and limitations.
python_serial_example.py	Example Python script to communicate with the Arduino over a USB serial connection. Reads 100 frequency measurements from the device, handles conversion and error checking, and outputs the average frequency. Designed for use with the Arduino sketch.
`LICENSE`	GNU GENERAL PUBLIC LICENSE.

5.3 Colorimetry of Dyes

5.3.1 Experimental Procedure:

The dye present in the supplied cough sweets was first identified by its E number. The mass of the cough sweets provided (two lozenges) was recorded using a top-pan balance. The lozenges were transferred to a 100 mL beaker, and approximately 60 mL of deionised water was added. The mixture was left to stand until the lozenges had fully dissolved, with occasional gentle stirring using a wooden stirrer to aid dissolution. Once dissolved, the solution was quantitatively transferred to a 100 mL volumetric flask using a small funnel. To ensure complete transfer, the beaker was rinsed twice with approximately 15 mL of deionised water per rinse, and the rinsings were added to the volumetric flask. The solution was then diluted to the mark with deionised water, stoppered, and inverted several times to ensure homogeneity. This solution constituted the cough sweet sample solution.

Calibration standards were prepared from a 50 μM stock solution of the Allura red dye (Sigma Aldrich). Five calibration solutions were prepared by pipetting appropriate volumes of the stock solution into 10 mL volumetric flasks and diluting each to the mark with deionised water to achieve final concentrations of 2.5×10^{-5} M, 2.0×10^{-5} M, 1.5×10^{-5} M, 1.0×10^{-5} M, and 0.5×10^{-5} M. All dilutions were performed using a 5 mL graduated pipette and disposable droppers for deionised water.

To characterise the colorimeter, two cuvettes were filled: one with deionised water (reference) and one with the most concentrated calibration solution (2.5×10^{-5} M). Each LED (red, green, and blue) was tested in turn. For each LED, the reference signal and the absorbance of the calibration solution were measured using the Python-acquisition program. The LED producing the highest absorbance was selected for subsequent measurements.

Cuvettes were filled with each of the five calibration solutions, the reference solution (deionised water), and the cough sweet sample solution. Using the selected LED (as determined in the characterisation step), the absorbance of each solution was measured. Prior to each measurement, the cuvette was placed into the holder and the lid secured to minimise stray light. Absorbance values were calculated using Beer-Lambert's Law.

Cough sweets were purchased at a local supermarket.

Algorithm 6 Colorimetry of Allura Red dye in a cough sweet

1: Prepare dye solutions of known concentrations
2: Construct LEGO housing with Arduino, LED, cuvette, light sensor
3: Connect components to appropriate Arduino pins
4: Open connection between Python and Arduino board
5: $I_0 \leftarrow$ intensity through solvent
6: $I \leftarrow$ intensity through dye solution
7: $A \leftarrow \log_{10}(I_0/I)$
8: Select optimal LED wavelength
9: Record A for each concentration c
10: Fit linear regression of A vs. c
11: $\varepsilon \leftarrow$ `gradient/path length`
12: Measure A of cough sweet solution
13: $c \leftarrow A/(\varepsilon \cdot l)$
14: **Output:** "Concentration of dye = c mol L^{-1}"

5.4 Iodine Clock

5.4.1 Experimental Procedure:

A reaction mixture was prepared by adding the following to a 250 mL stoppered conical flask placed on a magnetic stirrer: deionised water (160 mL), 2 M sulfuric acid (40 mL), potassium iodide (4.0 g), and hydrogen peroxide solution (8 mL, 6 vol). A magnetic stirrer bar was added, the flask was stoppered, and the solution was stirred at a moderate rate. The mixture was allowed to react to completion over the course of at least one hour. Following this period, the total amount of sodium thiosulfate solution (0.5 M) required to fully decolourise the reaction mixture was determined through incremental titration.

Aliquots of thiosulfate were added using an autopipette in decreasing volumes to improve precision near the endpoint. Initially, 4 mL aliquots were added with gentle swirling after each addition until the brown colouration faded to approximately half its original intensity. Subsequently, 2 mL aliquots were used under the same conditions until further fading occurred. Finally, 1 mL aliquots were added until the solution became fully colourless. The number of aliquots at each volume was recorded. The total volume of sodium thiosulfate (V_{total}) required to clear the solution was calculated as the sum of the individual volumes added.

Prior to starting the kinetic run, the previously determined V_{total} was entered into the Python analysis-acquisition program. The autopipette was set to deliver 1 mL aliquots of 0.5 M sodium thiosulfate. A measuring cylinder (250 mL) was placed on a magnetic stirrer and positioned centrally over a light sensor secured beneath it. The measuring cylinder was enclosed in a cardboard sheath to reduce ambient light interference. A flexible LED light was positioned above the setup to ensure uniform illumination. The cylinder was charged with deionised water (160 mL), 2 M sulfuric acid (40 mL), and potassium iodide (1 g), and the mixture stirred at 500 rpm.

In a separate beaker, hydrogen peroxide solution (8 mL, 6% vol.) was mixed with 0.5 M sodium thiosulfate (1 mL). Prior to initiating the reaction, the light sensor response was tested using the basic acquisition program. The signal was recorded five times with the sensor uncovered, and the average of these readings was calculated. A threshold value was defined as 75% of this average (rounded to the nearest integer) and input into the full acquisition program.

The reaction was initiated by simultaneously adding the pre-mixed H_2O_2/thiosulfate solution to the prepared reaction mixture in the measuring cylinder. The Python program monitored the light sensor in real time. When the signal dropped below the threshold (indicating iodine accumulation), a beep signalled the need for additional thiosulfate. A 1 mL aliquot of 0.5 M sodium thiosulfate was then added to restore the colourless state, and the number of aliquots used at each beep was entered into the program. This process was repeated until six to eight data points were acquired.

All chemicals were supplied from Sigma Aldrich or Thermo Fisher Scientific.

Algorithm 7 Iodine Clock Reaction

Connect light sensor to Arduino
Open connection between Python and Arduino board
Load cell with 200 mL H_2SO_4 and 0.03 M KI
$threshold \leftarrow 0.75 \cdot$ intensity of colourless solution
$t_0 \leftarrow$ current time
Add 8 mL H_2O_2 and 1 mL $Na_2S_2O_3$
while True **do**
  $signal \leftarrow$ sensor reading
  $avg \leftarrow$ average of last 10 values
  if $avg < threshold$ **then**
    Beep using `winsound`
    $t \leftarrow$ current time
    Store $t - t_0$
    User adds thiosulfate until colourless
    Record added volume v
    if $v = 0$ **then**
      `break`
    end if
  end if
end while
$y \leftarrow \ln(\texttt{total} - v)$
Plot y vs. *time*
$k' \leftarrow -$ gradient from regression
Output: "Pseudo rate constant $k' = k'$ s^{-1}"

6 Incorporation of Chemical Data Analysis

6.1 Useful Plotting Functions

Throughout the following description, terms such as 'plt' and 'np' are used as part of functions. These describe the library and their common call names: `plt` refers to the matplotlib library [2], and `np` to the NumPy library [3].

The spectral data shown in Figure 5a of the main text is saved in a `.csv` file format, which can be conveniently imported using the `np.loadtxt()` function, with Raman shift frequencies and intensities stored in separate lists or NumPy arrays. Plotting these data is straightforward using the `plt.plot(X_data, Y_data)` command, followed by `plt.show()` to render the figure. Plot customisation—such as adjusting line colour and width—can be achieved by adding parameters within the `plt.plot()` function. Axis labels are added using `plt.xlabel()` and `plt.ylabel()`, and to focus on the spectral region of interest (in this case, the 1000–1600 cm^{-1} range, encompassing the Fermi resonance and symmetric stretching modes), the x-axis limits are set using `plt.xlim(min, max)`. Adding annotations is similarly accessible, though slightly more involved. This is accomplished using the `plt.text(X_position, Y_position, text)` function, allowing students to highlight key features within their spectra. Additional options for text alignment, font style, size, and colour can also be specified, enabling clear and informative data presentation.

Although this has described the basic functions used to plot Figure 5a in the main text, we encourage readers to consult with the NumPy and Matplotlib online resources [4, 5] for more information regarding optional arguments that can be used with these functions for full customisation.

6.2 Particle in a 1D Box

Algorithm 8 Extract $\lambda_{\max}$ from absorbance data

Load CSV file, skip first two rows {Using `numpy.loadtxt()`}
Extract columns: `wavelength`, `absorbance`
`absorbance_max` $\leftarrow \max($`absorbance`$)$
`idx` $\leftarrow$ index of `absorbance_max`
$\lambda_{\max} \leftarrow$ `wavelength[idx]`
Output: "$\lambda_{\max} = \lambda_{\max}$ nm"

Algorithm 9 Particle in a Box: Wavefunctions and Probability Amplitudes

Define box length L, number of points $N \leftarrow 1000$
Create $x \leftarrow$ list of N points from 0 to L
$n\_values \leftarrow [n, m]$ {Quantum numbers}
for each i in n_values **do**
    $\psi_n \leftarrow \sqrt{\frac{2}{L}} \sin\left(\frac{n_i \pi x}{L}\right)$
    $\psi_n^2 \leftarrow (\psi_n)^2$
    Normalize: $\psi_n \leftarrow (\psi_n / \max(\psi_n)) + i$
    Plot x vs. ψ_n with label $n = n_i$
end for
Label axes and display plot

6.3 Flash Photolysis of Benzophenone

The transient processing app is freely available online on GitHub. [6] Here, we present a summative list of contents for the repository.

Table S3: Table of contents for Transient Processing App GitHub repository. [6]

Contents	Description
Transient Processing App.py	Main commented Python script implementing the GUI. Written using the `Tkinter` library, this tool processes and averages transient absorption traces and converts voltage to absorbance using Beer-Lambert's law.
README.md	Instructions for setup, use, and troubleshooting. Includes system requirements, guidance on configuring Spyder for interactive plotting, and a step-by-step walkthrough for students using the application in a lab setting.
LICENSE	GNU GENERAL PUBLIC LICENSE
1.csv – 30.csv	Example transient datasets in .csv format. Each file contains two columns (time and voltage). These datasets should be used to test the app and is real experimental data collected during the design of the flash photolysis experiment.

Algorithm 10 Fit experimental data to a second-order decay model

1: $data \leftarrow$ Load CSV, transpose to get t, A {Using `numpy.loadtxt()`}
2: Define model as function: $A(t) = \frac{A_0}{1+kA_0t}$
3: Set initial guesses: $A_0 = 0.3$, $k = 0.1$
4: Fit using a least squares approach {Using `scipy.optimize.curve_fit()`}
5: Extract optimal parameters: A_0^*, k^* {`popt`}
6: Compute uncertainties ΔA_0, Δk {Using `pcov`}
7: $\tau \leftarrow 1/k^*$
8: **Output:** $A_0^* \pm \Delta A_0$, $k^* \pm \Delta k$, and τ
9: Plot experimental data and fitted curve

References

(1) Perry, S. ArduinoConfig v1.0.0, https://github.com/Perry-SC/ArduinoConfig/tree/v1.0.0 (Accessed 08/08/2025).

(2) Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* **2007**, *9*, 90–95.

(3) Harris, C. R.; Millman, K. J.; van der Walt, S. J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N. J.; Kern, R.; Picus, M.; Hoyer, S.; van Kerkwijk, M. H.; Brett, M.; Haldane, A.; del Río, J. F.; Wiebe, M.; Peterson, P.; Gérard-Marchant, P.; Sheppard, K.; Reddy, T.; Weckesser, W.; Abbasi, H.; Gohlke, C.; Oliphant, T. E. Array programming with NumPy. *Nature* **2020**, *585*, 357–362.

(4) NumPy v2.3 Manual — numpy.org, https://numpy.org/doc/stable/reference/, (Accessed 17-07-2025).

(5) Matplotlib 3.10.3 documentation — matplotlib.org, https://matplotlib.org/stable/users/explain/figure/api_interfaces.html, (Accessed 17-07-2025).

(6) Hughes, D.; Perry, S. Transient Processing App v1.0.1, https://github.com/Perry-SC/Transient_Processing_App/tree/v1.0.1 (Accessed 08/08/2025).