IOP Publishing European Journal of Physics
Eur. J. Phys. 39 (2018) 065803 (18pp) https://doi.org/10.1088/1361-6404/aadaee

MATLAB time-based simulations of projectile motion, pendulum oscillation, and water discharge

Kaan Yetilmezsoy[1,3] **and Carl E Mungan**[2]

[1] Department of Environmental Engineering, Faculty of Civil Engineering, Yildiz Technical University, 34220, Davutpasa, Esenler, Istanbul, Turkey
[2] Physics Department, US Naval Academy, Annapolis, MD 21402-1363, United States of America

E-mail: yetilmez@yildiz.edu.tr and mungan@usna.edu

Received 27 March 2018, revised 8 July 2018
Accepted for publication 17 August 2018
Published 24 September 2018

Abstract

In addition to teaching undergraduate technical majors about theory and experiments, an important component of their instruction should be to develop proficiency in using computers for simulations and data presentation. A particularly appropriate course for this purpose is classical mechanics because one of the primary goals of that course is to simulate the motion of physical systems, which is greatly aided by visualizations that help to build students' intuition about expected behaviors. As a demonstration of the ease with which this objective can be accomplished using the MATLAB software package, fully executable scripts are presented here for three commonly studied systems: point projectiles, simple pendula, and water flowing out of a hole in a tank. It is much easier for students to edit a functioning script than to try to write their own from scratch. However, the present scripts are sufficiently short and well-documented that instructors can readily ask their students to modify and adapt them according to their local instructional environments and purposes.

Keywords: computational simulation, projectile motion, simple pendulum, water discharge, MATLAB

(Some figures may appear in colour only in the online journal)

[3] Author to whom any correspondence should be addressed.

1. Introduction

In today's world, educators and students are aware of the needs of the science education community regarding the use of computation in physics. There is no doubt that conventional hand calculations are often laborious and time-consuming when the number of inputs is increased, and/or different initial parameters are introduced in every new scenario of a complicated problem. More importantly, computer-aided techniques are inevitable in the analysis of many physical systems (e.g. structural mechanics, and fluid mechanics) whose behavior must be studied both visually and practically under different dynamic conditions. Although theoretical knowledge obtained during lectures is important and can play a significant role in learning fundamental principles, scientific interpretation of a concept is limited for lectures full of theory. Implementation of modern computer technologies (e.g. animation, dynamic objects, and visualization of the studied physics phenomena) improves students' perception of the material and help them comprehend it [1]. It is essential to teach students about computational methods, modeling knowledge, and simulation experience to encourage them to create robust and practical solutions to the real-world problems that they will face in their future careers [2, 3].

The American Association of Physics Teachers recommends that computer-based applications be incorporated into the undergraduate physics curriculum [4]. Topics in classical mechanics are particularly appropriate for this purpose because they are well-suited to computational approaches [5]. A synthesis of curriculum development, computational physics, computer science, and physics education will be useful for scientists and students wishing to write their own simulations and develop their own curricular material [6]. Moreover, to support the computer-based applications, inclusion of course-related experiments in the curriculum will offer a remarkable opportunity for both students and discipline-specific practitioners to comprehend the theoretical issues in a visual manner [3].

In a study performed at the International IT University in Kazakhstan, Daineko *et al* [1] investigated the effect of using physics virtual laboratories (PVL) on the performance of Physics-1 and Physics-2 courses. PVL was developed in C#(.NET) based on the XNA4.0 framework and realized with the use of 3D modeling by employing Blender and Maya 3D software. The authors implemented a PVL software package consisting of nine physical experiments (i.e. the Atwood machine, Maxwell pendulum, Clement–Desormes method, Stefan–Boltzmann constant, photoconductivity properties of semiconductors, magnetic induction, direct current laws, Ohm's law for alternating current, and Frank–Hertz experiment) from various domains of physics integrated into a single curriculum. The study concluded that the introduction of innovative computer technologies, such as virtual laboratories, could be used during lectures to enhance the teaching experience and improve education quality, without large facilities and material costs.

Investigations on the incorporation of computational science into introductory physics courses revealed positive results. For instance, at the University of Maryland in the USA, Redish and Wilson [7] proposed a M.U.P.P.E.T.-based computational environment at the beginning of an introductory physics course for student programming. The authors saw several benefits of teaching physics in a computerized environment. The study concluded that when the computational science tools were combined with introductory physics courses, students were able to discuss real-world problems (e.g. projectile motion with air resistance) without needing an intensive mathematical knowledge.

A study based on an object-oriented programming language was performed by Lee and Lee [8] at Soongsil University in Korea to discuss various aspects of developing interactive physics education software using Adobe Flash and ActionScript source code for the simulator. They demonstrated three examples of interactive simulations (i.e. one-dimensional kinematics: bus simulator; Newton's laws of motion and law of gravity: motions of two celestial bodies interacting with gravity; and motions of eight celestial bodies in the solar system) based on Flash software that enable students to understand basic concepts in physics. The authors concluded that the students could grasp relevant concepts intuitively with the help of the object-oriented nature and powerful graphics capability of the proposed programming.

Another study reported by Sherin [9] aimed to compare the performance of two distinct groups of students (from University of California, Berkeley, USA), an algebra pool and a programming pool, where each group consisted of five pairs of students. In the study, the algebraic physics group was asked to solve traditional textbook problems (e.g. shoved block, vertical pitch, air resistance, mass on a spring, stranded skater, buoyant cube, and running in the rain). The students in the programming physics group were asked to create a set of simulations of physical motions (e.g. shoved block, dropped ball, tossed ball, air resistance, and mass on a spring) in the Boxer programming environment. The author concluded that the algebraic notation of the physics formulas did not indicate causal relations between the parameters, and thus students were directed to discover the existence of balance between the two sides of an equation rather than causal relations. Conversely, programming physics students were encouraged to seek time-varying phenomena and express certain types of causal intuitions.

In a computer-aided physical multidomain modeling study conducted at the University of Ljubljana in Slovenia, Zupančič and Sodja [10] modeled two mechanical systems (an inverted pendulum and a laboratory helicopter) using MATLAB-Simulink and Dymola-Modelica environments within the framework of an educational application project. For their industrial application project, the authors described the modeling and control of thermal and radiation flows in buildings, and two applications from mineral–wool production: modeling of a mechanical pendulum system, and a recuperator process. The study concluded that MATLAB-Simulink is a very usable environment for the design of control systems based on linearized models. Likewise, Modelica-based environments with inverse models can be usable in the control of mechanical systems.

Another study was undertaken by Bozkurt and Ilik [11] at Selcuk University in Turkey to explore the impact of teaching supported by interactive computer simulations on students' beliefs about physics and physics achievements. In the study, simulations (the moving man and energy skate park) benefited from Java and Flash programming resources. According to the authors' analysis, the study concluded that the groups who used computer simulations were more successful than those who worked with traditional methods, and the courses with interactive simulations had a positive effect on students' beliefs about physics and physics achievements.

In another study that was conducted by Taub *et al* [12], at the Weizmann Institute of Science in Israel, the research objective was to explore the effect of computer science on the students' conceptual understanding of the physics behind formulas. The programming tools used in the study were Easy Java Simulations and Maxima. To assess the learning effects of computer simulations in physics education, the students were subjected to three different disciplines of physics: (i) kinematics (simulation of launching a rocket and intercepting an

enemy rocket); (ii) dynamics (simulation of a car driving on a circular road); and (iii) optics (simulation of an object located in front of a lens positioned in front of a mirror). With the help of object-oriented programming of a physical system, the analysis of the authors indicated that each of the implemented domains contributed to the emergence of knowledge integration processes, and promoted conceptual understanding of the physics concepts.

Finally, Sarabonda *et al* [13] investigated the contribution of a computer simulation to students' learning of physics concepts (weight and mass) by implementing simulations based on Modellus software. In the analysis, students from four schools in northern Portugal were subjected to three different scenarios: (i) using only 'hands-on' activities, (ii) using only a computer simulation, and (iii) using both. The study concluded that the use of the computer simulation improved students' learning of the physics concepts, when either used alone or together with 'hands-on' activities. Additionally, the authors emphasized that the total gains attained were linked to the teachers' pedagogy/mediating role when using the computer simulation to teach the concepts of weight and mass.

The present study aims to expand the positive findings reported in these studies based on the use of computation in physics. The main intention of the authors is not on the theoretical analysis of the problems, but on the computer-aided programming approach. For this purpose, in the present paper, three introductory problems are considered: (i) projectile motion of several simultaneously launched point particles, (ii) the oscillational period of a simple pendulum at large amplitudes, and (iii) the gravity-driven discharge of water through a circular hole at the bottom of a cylindrical tank whose axis of symmetry is vertical. The motions are simulated using scripts written in MATLAB. The concise description of MATLAB functions will make this research of pedagogic interest to a wide range of readers, including those with no experience in MATLAB. Before the concluding remarks, the advantages of using MATLAB over other programming languages are provided for the readers. Detailed explanations of the steps needed to write the algorithms are presented, and fully operational codes are included in the appendix. Furthermore, the computational outputs are compared to experimental results in the cases of the pendulum and water discharge problems, using latitude-corrected values of the acceleration g due to gravity. Excellent agreements are found.

2. Kinematics of projectile motion

Imagine that a group of students are discussing the motion of n balls projected with different initial speeds v_1, v_2,…,v_n and launch angles θ_1, θ_2,…,θ_n above the horizontal. They would like to simulate the resulting trajectories on a single graph and determine the flight time, range, and maximum height attained by each ball. Laborious hand calculations of the sort performed in [14] can be avoided using computer software to animate the motion.

The position of the i th ball (where i is an index running from 1 to n) at time t after being launched from the origin is

$$x_i = v_i t \cos\theta_i \tag{1}$$

for the horizontal position in the direction of motion, and

$$y_i = v_i t \sin\theta_i - \tfrac{1}{2} g t^2 \tag{2}$$

for the upward vertical position, where g is the acceleration due to gravity. Equation (1) uses the fact that the horizontal component of a ball's velocity is constant during the motion with a value $v_i \cos\theta_i$, whereas the vertical component varies in time as

$$v_{yi} = v_i \sin\theta_i - gt. \tag{3}$$

By setting equation (2) to zero, the flight time of the i th ball is found to be

$$T_i = \frac{2v_i \sin\theta_i}{g} \tag{4}$$

and that time can be substituted into equation (1) to find the range R_i. Likewise, half of this flight time can be substituted into equation (2) to find the maximum height H_i attained by the i th ball.

3. Oscillation period of a simple pendulum

A simple pendulum is an idealized model [15] consisting of a point particle of mass m attached to the end of a massless inextensible string of length L. When displaced from its vertical equilibrium position, the bob swings back and forth. Newton's second law gives rise to the differential equation of motion [16]:

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \frac{g}{L}\sin\theta = 0. \tag{5}$$

This expression can also be obtained from conservation of mechanical energy. If it is assumed that the angle is less than 0.1 rad (or about 6°) so that $\theta \ll 1$, then $\sin\theta \approx \theta$ in radians. In that case,

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \omega^2\theta = 0 \tag{6}$$

where $\omega = \sqrt{g/L}$ is the angular frequency of oscillation (to be carefully distinguished from the angular velocity $\mathrm{d}\theta/\mathrm{d}t$ of the bob). The error in this small-angle approximation is of order θ^3 (from the Taylor series for $\sin\theta$). If the initial angular displacement is θ_0 and the bob is released from rest, then the solution to equation (6) is

$$\theta(t) = \theta_0 \cos\left(\sqrt{\tfrac{g}{L}}\, t\right) \text{ provided that } \theta_0 \ll 1. \tag{7}$$

The motion is simple harmonic with a period of

$$T_0 = 2\pi\sqrt{\frac{L}{g}} \text{ provided that } \theta_0 \ll 1. \tag{8}$$

This period is independent of the amplitude θ_0, a property known as isochronism, as discovered by Galileo [17, 18].

However, for larger amplitudes the period T is bigger than T_0. In that case, the period can be expanded in a Legendre series [18] as

$$T = 2\pi\sqrt{\frac{L}{g}} \sum_{n=0}^{\infty} \left\{ \left[\frac{(2n)!}{(2^n n!)^2} \right]^2 \sin^{2n}\left(\frac{\theta_0}{2}\right) \right\}. \tag{9}$$

An excellent approximation is provided by the arithmetic–geometric mean method [19], which predicts

$$T \approx 8\pi\sqrt{\frac{L}{g}} \left(1 + \sqrt{\cos\tfrac{\theta_0}{2}}\right)^{-2} \tag{10}$$

with a relative error of less than 1% for angles up to 163°.

4. Water discharging from a vertical cylindrical tank

Water flowing out of a circular hole in a tank is an example of unsteady flow because the fluid mass inside a control volume (CV) varies with time. That is, $dm_{CV} \neq 0$ where dm_{CV} is the rate of change of the mass within the CV [3, 20]. Using Torricelli's law, the volumetric flow rate is

$$Q = C_d \sqrt{2gz}\ \ \pi d_o^2/4 \tag{11}$$

where C_d is the coefficient of discharge ranging in value from 0 to 1, z is the time-varying distance from the free surface of the water in the tank to the hole, d_o is the diameter of the hole, and g is the gravitational acceleration. In the absence of viscous losses and the *vena contracta* [21], C_d would be equal to 1.

Using the continuity equation, equation (11) can be integrated for a vertical tank of inner diameter D to find the time T required for the surface of the water to drain down to a height h above the hole as

$$\int_0^T \mathrm{d}t = -\int_{h_0}^{h} \frac{\pi D^2/4}{C_d \sqrt{2gz}\ \pi \mathrm{d}_o^2/4}\mathrm{d}z \tag{12}$$

starting with the water at rest at an initial height of h_0. Performing the integrals gives

$$T(h) = \left(\frac{1}{C_d}\right)\left(\frac{D}{d_o}\right)^2\left(\frac{2}{g}\right)^{0.5}(\sqrt{h_0} - \sqrt{h}). \tag{13}$$

5. Computational procedure

Computer-based simulations were conducted by writing scripts in MATLAB R2018a on a 64-bit Windows 10 system. The step-by-step computational procedure is summarized as follows.

Step 1. *Free up system memory and set the output format*. The 'Editor Window' (for a new M-File) is opened in MATLAB. All of the current contents of system memory and all variables in the 'Command Window' are erased, and the visual 'Command Window' and command output are customized using the built-in functions *clear* (which releases all variables from the current 'Workspace' and from system memory), *clc* (which clears all input and output commands from the display and homes the cursor), and *format short* (which sets the output display format to a scaled fixed-decimal format with four digits after the decimal point).

Step 2. *Set the input parameters*. In the 'Editor Window', an assignment is made for each input variable using the built-in function *input*: (**a**) for projectile motion—launch angles (θ_1, θ_2,…,θ_n), initial speeds (υ_1, υ_2,…,υ_n), number n of balls projected, and gravitational acceleration g; (**b**) for pendulum oscillation—length L of the string, angular amplitude θ_0, and gravitational acceleration g; and (**c**) for water discharge—initial height h_0, final height h, diameter D of the water tank, diameter d_0 of the hole, coefficient of discharge C_d, and gravitational acceleration g.

Step 3. *Define the formulas.* The equations are coded using different built-in functions according to the symbolic and numerical methods used: (**a**) for projectile motion—the horizontal and vertical components of the initial velocities υ_{0x} and υ_{0y}, total flight time T, range R, and maximum height H of each ball are computed using the built-in functions *cosd* (cosine of the argument in degrees), *sind* (sine of the argument in degrees), and *max* (returns the largest element in an array); (**b**) for pendulum oscillation—the small-angle approximation, Legendre series, and arithmetic–geometric mean method are implemented using the built-in functions *pi* (value of π), *syms* (shortcut for symbolic variables), *gamma* (gamma function),

sin (sine of the argument in radians), *symsum* (symbolic summation of a series), *double* (convert the value to double precision), and *sqrt* (square root); and (**c**) for water discharge—the time T for the surface of the water to drain down to a height h above the hole is determined using the built-in functions *syms, pi, sqrt, int* (symbolic integration), *double,* @ (function handle), *quad* (numeric integration based on adaptive Simpson quadrature), and *fix* (round a scalar to the integer nearest to zero). In addition, the stopwatch timer codes *tic* and *toc* are used to measure the CPU time or time elapsed for an analysis in seconds.

Step 4. *Output the results*. Output formats are created using the built-in function *fprintf*. This function specifies a floating-point number $\%d_1.d_2$ with a width of d_1 digits having d_2 digits after the decimal point. It can also include the control character /n to start a new line. After running the scripts in the 'Editor Window', the resulting numeric and character arrays are sent to the 'Command Window' and stored in the 'Workspace'.

Step 5. *Construct the animations*. Primitive line objects, in terms of which time-based motions are animated, are created using the built-in function *line* using data stored in vectors **x** and **y**. As options in the *line* command, 'LineStyle' such as '-', 'LineWidth' such as '2', 'Color' such as 'r', 'Marker' such as '.', and 'MarkerSize' such as '40' are defined to control the appearance and behavior of the lines and markers.

Additionally, the following built-in functions are introduced to enhance the simulation environment: *axis* (which sets the axis limits and aspect ratios), *axis equal* (which uses the same length for the data units along each axis), *title* (which adds a specified title to the axes or chart), *xlabel* (which labels the x-axis of the current axes or chart), *ylabel* (which labels the y-axis of the current axes or chart), *box on* (which displays a box around the current axes), *grid on* (which displays major grid lines for the current axes or chart), *set* (which specifies a value for the property name of the identified object), *gca* (which returns the current axes or chart for the current figure), *findall* (which finds a text object for the x-axis label), *gcf* (which returns the current figure handle), *hold on* (which holds the current plot in memory while adding new curves to it), *sprintf* (which formats a data string), and *paus*e (which temporarily stops execution).

Step 6. *Animate the motions*. In the final step, '*fo*r index = values, statements, *end*' loops are constructed to execute a group of statements a specified number of times. For the present problems, *values* are created as time vectors by using the built-in function *linspace* (which returns a row vector of n evenly spaced points in the interval [0, T] for projectile motion, [θ_0, $-\theta_0$] for pendulum oscillation, and [0, T] for water discharge). The *set* function is used to display the property names and values of the line objects created in Step 5. For the projectile motion and pendulum oscillation, the function *plot* creates a two-dimensional (2D) line graph of the y data versus the corresponding x values. To simulate the projectile motion and water discharge, the *sprintf* function is used to format the time data as a string. For the water discharge problem, the animation of the free surface of the water draining down is colored using the function *fill* (which creates 2D polygons specified by x and y with the color of 'ColorSpec') or *patch* (which creates a solid patch having its shape specified by the coordinate vectors x and y). In all scripts, the built-in function $M(r) = getframe$ is implemented to capture the current axes as it appears on the screen as a movie frame. Here M is a structure containing the image data, and r denotes its index. For the pendulum oscillation, the function *movie* is used to play the recorded movie defined by a matrix whose columns are frames produced by *getframe*, using the current axes as the default target. The code *movie*(M, $n < 0$) plays the movie $|n|$ times. The negative value of n means that each cycle is played forward and then backward.

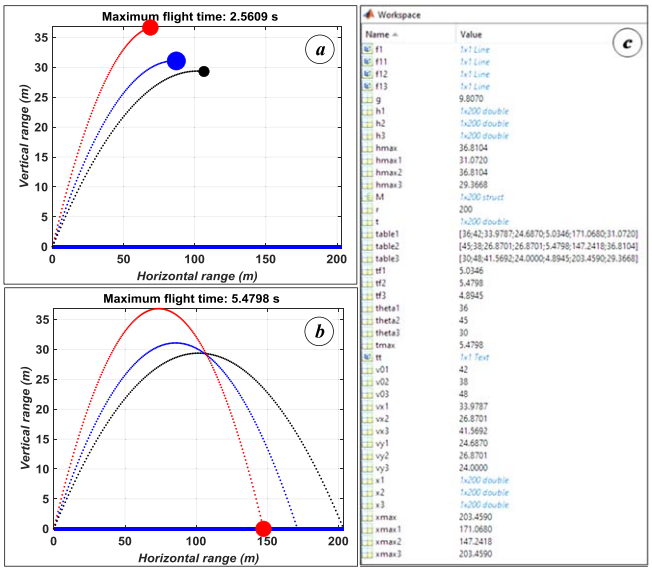

Figure 1. Projectile motion problem: (a) a screenshot taken at an intermediate point during the animation; (b) a screenshot at the end of the animation; and (c) the contents of the 'Workspace' window after completion of the script.

6. MATLAB-based simulations

Calculations are conducted for each problem by running the MATLAB scripts presented in tables A1–A5 of the appendix. The scripts are evaluated in the 'Editor Window' and the outputs for various input variables are shown in figures 1–3.

Consideration of the 'Elapsed time is … seconds' outputs of the Stopwatch Timer code in the 'Command Window' demonstrates that time-consuming calculations can be avoided with the help of computational software. MATLAB provides an excellent educational platform for observing the results both numerically and visually. This savings enables students to spend more time learning and practicing problems for different inputs. Even though the computation time required for the pendulum oscillation problem is higher than that for the other problems owing to the symbolic solution applied to sum the Legendre series, a MATLAB-based approach remains satisfactory in terms of speed of execution.

The maximum CPU usages recorded in the Task Manager are 17%, 4%, and 18% when simulating projectile motion, pendulum oscillation, and water discharge problems, respectively, in MATLAB. These differences in CPU demand result from the size of the time vectors involved and the number of captured movie frames. Additional MATLAB scripts are presented in tables A4 and A5 of the appendix for the pendulum oscillation, and the computational outputs obtained from these codes are presented in figures 4 and 5.

7. Comparison with experimental studies

To validate the computational analysis, the predictions were experimentally tested in the 2017–2018 Spring Term with active participation of 60 undergraduate students at Yildiz Technical University in Istanbul, Turkey, in the undergraduate course CEV2232 'Fluid Mechanics for

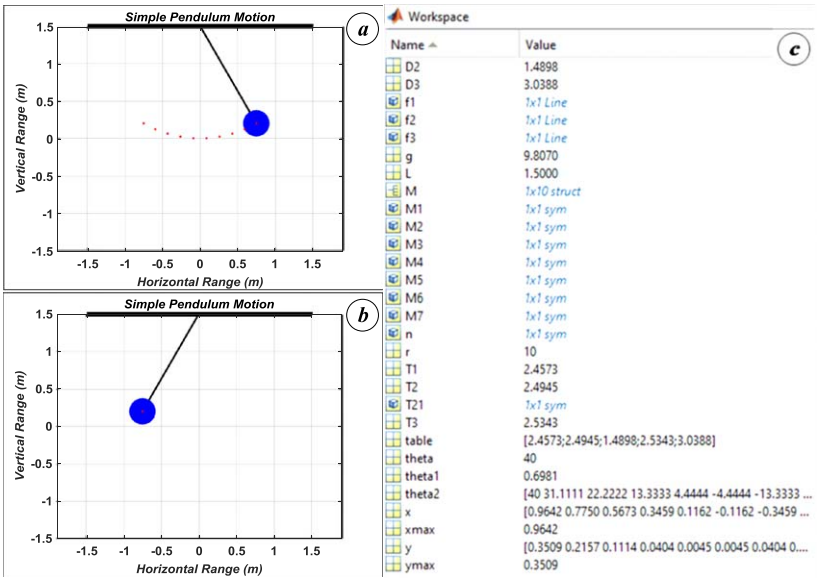

Figure 2. Pendulum oscillation problem: (a) a screenshot during the animation; (b) a screenshot at the end of the animation; and (c) the contents of the 'Workspace' window after completion of the script.

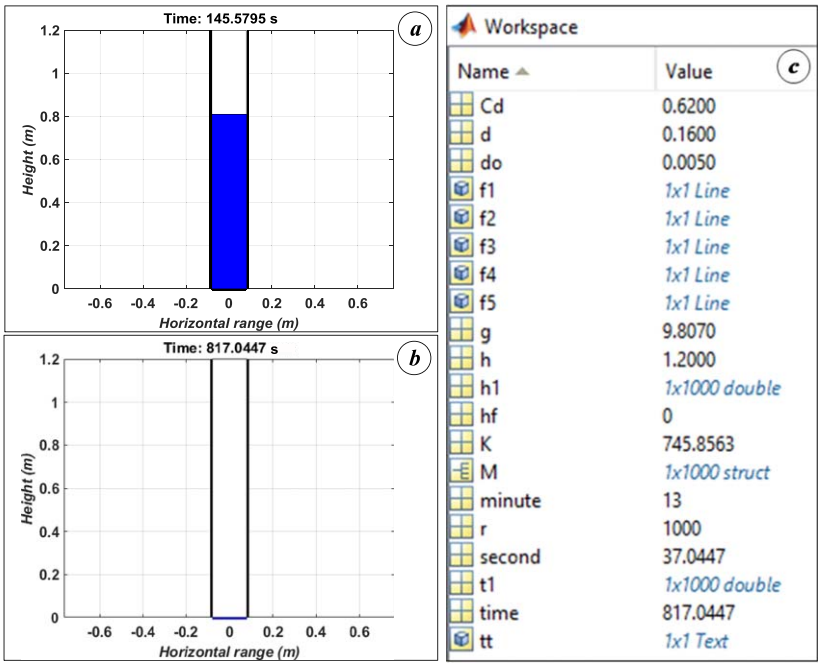

Figure 3. Water discharge problem: (a) a screenshot during the animation; (b) a screenshot at the end of the animation; and (c) the contents of the 'Workspace' window after completion of the script.

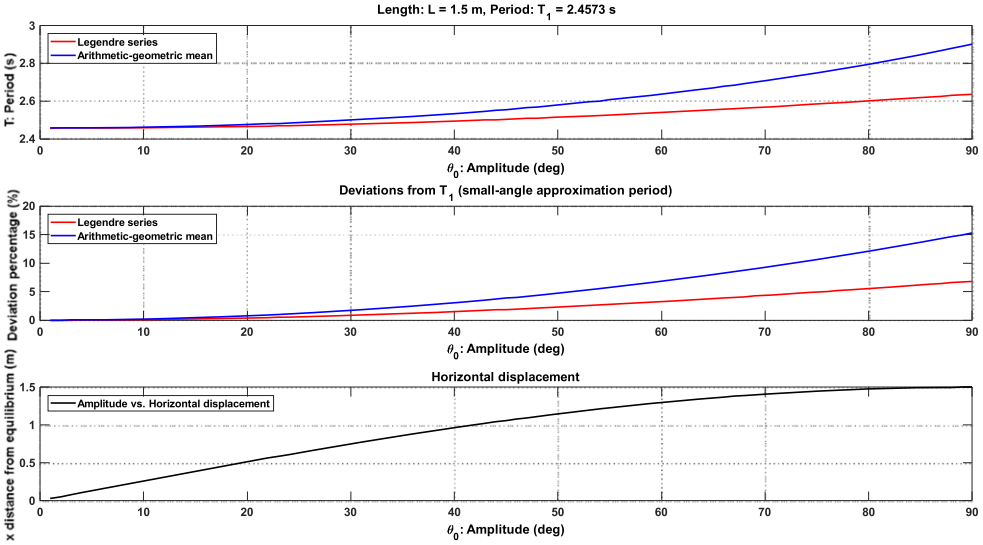

Figure 4. Output obtained by running the script in table A4.

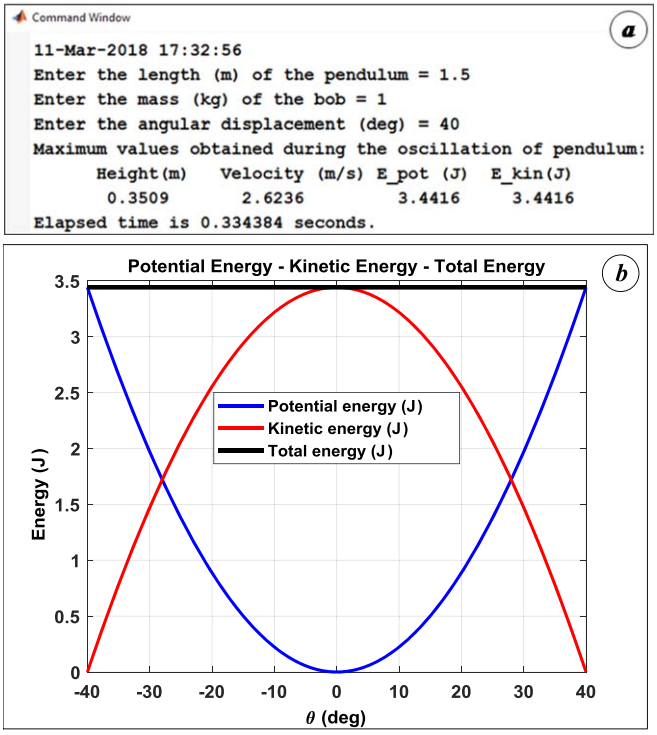

Figure 5. Output of the script in table A5: (a) numerical results (based on input variables, energy values, and elapsed time) in the 'Command Window' and (b) graph of the potential and kinetic energies during a half-cycle of oscillation.

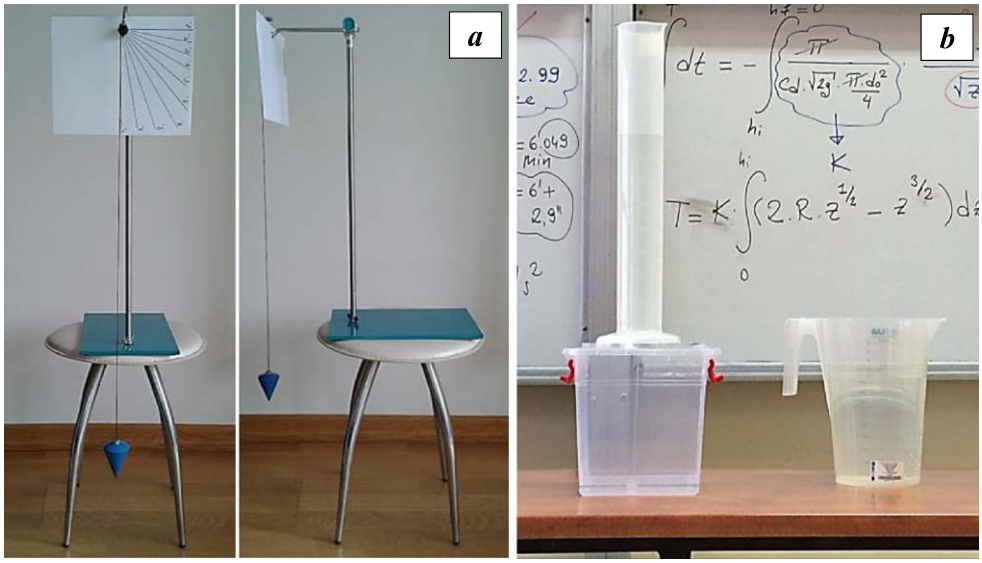

Figure 6. Experimental setups for (a) the pendulum experiment and (b) the tank discharge experiment.

Engineers'. Pendulum oscillation and water discharge measurements were conducted. Each student was asked to measure the oscillation period and discharge time using a watch or mobile phone. Each experiment was repeated three times to determine its reproducibility.

A 1 L polypropylene graduated cylinder (Isolab product #025.01.600) and stainless steel pendulum (Isolab product #S.049.01.200.001) are used in the experimental setup, as photographed in figure 6. First, the value of the gravitational acceleration g at the location of the laboratory is numerically calculated [22], and then verified using a simple pendulum. Its theoretical value as a function of latitude φ is

$$g_{\text{theor}} = g_{45} - \tfrac{1}{2}(g_{\text{poles}} - g_{\text{equator}})\cos 2\varphi \tag{14}$$

while its experimental value is (from equation (10))

$$g_{\text{exp}} = 64\pi^2 \frac{L}{T_{\text{exp}}^2}\left(1 + \sqrt{\cos\tfrac{\theta_0}{2}}\right)^{-4}. \tag{15}$$

Here, g_{45}, g_{poles}, and g_{equator} are, respectively, 9.806 m s^{-2} at 45° latitude, 9.832 m s^{-2} at the poles, and 9.780 m s^{-2} at the equator. The latitude for the lab is 41°02′36.60′ N so that $\varphi = 41.0435°$. The length of the string is $L = 0.99$ m, and the initial angular displacement is $\theta_0 = 10°$.

Equation (14) predicts $g_{\text{theor}} = 9.802\,\text{m s}^{-2}$. Pendulum experiments performed for ten oscillations result in an average oscillation period of $T_{\text{exp}} = 1.999 \pm 0.036\,\text{s}$, so that $g_{\text{exp}} = 9.819\,\text{m s}^{-2}$ from equation (15). Experimentally, the average discharge time is found to be $355 \pm 1\,\text{s}$, which agrees well with the value computed using equation (13).

At the end of the experiment, the students were asked to provide anonymous comments about it. Some of them are reproduced below (where M denotes a male student and F a female student).

- M: 'Experiments were very useful and encouraging for me because of the visualization of a theoretical field in real life, thanks to this educational activity.'
- F: 'I believe that this application was an unforgettable, attractive, and remarkable attempt that has made both theoretical calculation and computer-based analysis more permanent in mind.'
- M: 'I think that experiencing the validation of the computational analysis in a visual way increased the understanding of the principal concepts for the respective problems.'
- F: 'I have experienced a comprehensible and target-based lecture with the help of the experimental studies.'

8. Benefits of MATLAB over other programming languages

MATLAB (derived from MATrix LABoratory) is a sophisticated software package that covers built-in functions to achieve a wide range of tasks, from mathematical operations to three-dimensional image processing. MATLAB provides a full set of programming tools that allows users to customize programs to their own specifications. Beginning students who learn the built-in functions will be well-prepared to use MATLAB. Then, the best approach is to teach them both the programming concepts and advanced function features for their future careers [23]. Because MATLAB is straightforward to use, it is a perfect computational environment for this task in teaching programming and problem-solving techniques [2, 3, 23–28].

Several mathematical variables can be easily entered into MATLAB's computing environment as matrices ($r \times c$, where $r > 1$ and $c > 1$), scalars ($r \times c$, where $r = c = 1$), row arrays or horizontal vectors ($1 \times c$, where $c > 1$), and column arrays or vertical vectors ($r \times 1$, where $r > 1$) [3]. This approach enables complex calculations to be efficiently solved using various built-in functions and/or user-defined functions in M-files [3, 24]. Thus, if a problem can be formulated with a matrix-based solution, MATLAB executes substantially faster than a similar program in a high-level language. While MATLAB provides the user ease of creating a matrix, the matrix must be defined in C programming using looping such as a 'for/end loop'. Additionally, the main advantage of MATLAB over C programming when doing numerical calculations is that MATLAB is interpreted. That is, you can see the result of one command before you continue to the next one, unlike C programming where you must compile the source each time you make a change [29].

A 'number-crunching' program (i.e. large-scale processing of numerical data) is generally easier to write in MATLAB compared to other programs [24]. For instance, Python is a general-purpose programming language requiring add-on libraries for performing even basic mathematics. Matrix and array mathematics in Python requires function calls, not natural operators, and you must keep track of the differences between scalars, one-dimensional (1D) arrays, and 2D arrays. MATLAB, on the other hand, makes no artificial distinction between scalar, 1D, 2D, and multidimensional arrays. Unlike Python, MATLAB toolboxes offer professionally

developed, rigorously tested, field-hardened, and fully documented functionality for scientific and engineering applications [30]. Likewise, in contrast to R programming, MATLAB provides (i) a high-productivity development environment, (ii) extensive toolboxes and apps, (iii) faster performance and easy deployment, and (iv) assurance of quality and reliability [31].

Furthermore, MATLAB has the additional following advantages over other methods or languages [32]: (i) someone can perform operations from the command line as a sophisticated calculator and develop programs and functions that implement repetitive tasks; (ii) adding two arrays together needs only one single-line command instead of a for or while loop; (iii) it is easy to plot data and then specify color, line styles, and markers by using the graphical interactive tools; and (iv) its functionality can be greatly expanded by the addition of toolboxes (sets of specific functions that provide more specialized functionality).

9. Concluding remarks

The present computational analysis meets important instructional and research-based objectives. Fundamental principles of kinematics and mechanics from introductory physics and engineering are simulated for three different problems. The detailed codes are provided (as appendices) and are written using MATLAB. The main points of the codes are concisely explained, the results of the codes are compared with relevant experiments, and the educational impact of the whole experience is evidenced by a selection of student comments from a class of 60. Using the powerful and flexible properties of MATLAB, notably including its numerical and graphical tools, many challenging aspects of these extended scenarios can be investigated by students. Only a few previous papers [33] have presented complete MATLAB code, a comparison of simulated and experimental results at the introductory physics level, and field testing of the computer and lab work in an actual course with collection of anonymous student evaluations of the projects. In light of the positive findings of this work, future work will extend these simulations to other theoretical and experimental topics.

Appendix

Fully executable MATLAB codes of the three time-based problems from the main text are presented in the following five tables.

Table A1. Script for the projectile motion problem of 3 balls having different initial speeds and launch angles.

```
%Ball motion problem (simulation of n balls)

%STEP 1
clear, clc
format short
disp(datestr(now,0))

%STEP 2
theta1 = input('Enter the angle (deg) between velocity vector and ground = ');
v01 = input('Enter initial velocity (m/s) = ');
theta2 = input('Enter the angle (deg) between velocity vector and ground = ');
disp(' theta2 v02 vx2 vy2 tf2 xmax2 hmax2');
fprintf('%10.0f %6.0f %9.4f %9.4f %9.4f %9.4f %9.4f/n',table2);
disp('————————————————————————————————————————————————————————');
table3=[theta3;v03;vx3;vy3;tf3;xmax3;hmax3];
disp(' theta3 v03 vx3 vy3 tf3 xmax3 hmax3');
fprintf('%10.0f %6.0f %9.4f %9.4f %9.4f %9.4f %9.4f \n',table3);
toc

%STEP 5

xmax = max([xmax1,xmax2,xmax3]);
hmax = max([hmax1,hmax2,hmax3]);
```

Table A1. (Continued.)

```
v02 = input('Enter initial velocity (m/s) = ');
theta3 = input('Enter the angle (deg) between velocity vector and ground = ');
v03 = input('Enter initial velocity (m/s) = ');
tic

%STEP 3
%Horizontal and vertical components of the velocity vectors
vx1 = v01*cosd(theta1);
vy1 = v01*sind(theta1);
vx2 = v02*cosd(theta2);
vy2 = v02*sind(theta2);
vx3 = v03*cosd(theta3);
vy3 = v03*sind(theta3);

%Total flight times (sec) for the motion
g = 9.807; %Gravitational acceleration (m/s2)
tf1 = 2*vy1/g;
tf2 = 2*vy2/g;
tf3 = 2*vy3/g;

%Maximum horizontal and vertical ranges (meters) for the motion
xmax1 = vx1*tf1;
hmax1 = vy1*(tf1/2)-0.5*g*(tf1/2)^2;
xmax2 = vx2*tf2;
hmax2 = vy2*(tf2/2)-0.5*g*(tf2/2)^2;
xmax3 = vx3*tf3;
hmax3 = vy3*(tf3/2)-0.5*g*(tf3/2)^2;

%STEP 4
table1=[theta1;v01;vx1;vy1;tf1;xmax1;hmax1];
disp(' theta1 v01 vx1 vy1 tf1 xmax1 hmax1');
fprintf('%10.0f %6.0f %9.4f %9.4f %9.4f %9.4f %9.4f \n',table1);
disp('————————————————————————————————————————————————');
table2=[theta2;v02;vx2;vy2;tf2;xmax2;hmax2];
f1 = line([0 xmax],[0 0],'linestyle','-','color','b','linewidth',4);
f11 = line(0,0,'linestyle','none','marker','.','markersize',70,'color','b');

f12 = line(0,0,'linestyle','none','marker','.','markersize',60,'color','r');
f13 = line(0,0,'linestyle','none','marker','.','markersize',40,'color','k');
axis([0 xmax 0 hmax]);
%title('Projectile Motion')
xlabel('\itHorizontal range (m)','FontSize',14,'FontWeight','Bold')
ylabel('\itVertical range (m)','FontSize',14,'FontWeight','Bold')
box on
grid on
set(gca,'FontSize',14,'FontWeight','Bold')
set(findall(gcf,'type','text'),'FontSize',14,'FontWeight','Bold')
hold on
tt = title(sprintf('Maximum flight time: %0.4f sec', 0),'FontSize',14);
pause(5)

%STEP 6
tmax = max([tf1,tf2,tf3]);
t = linspace(0,tmax,200);
for r = 1:length(t)
x1(r) = vx1*t(r);

h1(r) = vy1*t(r)-0.5*g*t(r)^2;
x2(r) = vx2*t(r);
h2(r) = vy2*t(r)-0.5*g*t(r)^2;
x3(r) = vx3*t(r);
h3(r) = vy3*t(r)-0.5*g*t(r)^2;
plot(x1(r),h1(r),'.b',x2(r),h2(r),'.r',x3(r),h3(r),'.k','LineWidth',1)
set(f1,'xdata',[0 xmax],'ydata',[0 0]);
set(f11,'xdata',x1(r),'ydata',h1(r));
set(f12,'xdata',x2(r),'ydata',h2(r));
set(f13,'xdata',x3(r),'ydata',h3(r));
set(tt, 'String', sprintf('Maximum flight time: %0.4f sec', t(r)));

M(r) = getframe;
end
```

Table A2. Script for the oscillation period of a simple pendulum.

```
%Pendulum oscillation #1 (periods, deviations, and simulation)

%STEP 1

clear, clc
format short
disp(datestr(now,0))

%STEP 2
L = input('Enter the length (m) of the pendulum = ');
tic
g = 9.807; %Gravitational acceleration (m/s2)
theta = input('Enter the angular displacement (deg) = ');
theta1 = theta*pi/180; %Conversion to radian
disp('Computed periods (sec) and deviations from small-angle approximation:');
disp(' T1(sec) T2(sec) T3(sec) ');
disp('Simple harmonic Legendre Deviation(%) Aritmetico Deviation(%)');
fprintf('%10.4f %14.4f %10.4f %13.4f %11.4f \n',table);
toc

%STEP 5
f1 = line([-L L],[L L],'linestyle','-','color','k','linewidth',6);
f2 = line([0 0],[L 0],'linestyle','-','color','k','linewidth',2);
f3 = line(0,0,'linestyle','none','marker','.','markersize',100,'color','b');
axis([-L L -L L])
axis equal

title('\itSimple Pendulum Motion','FontSize',14)
xlabel('\itHorizontal Range (m)','FontSize',14)
```

Table A2. (Continued.)

```
%STEP 3
%Period for the small-angle approximation:
T1 = (2*pi)*((L/g)^0.5);

%Legendre series:
syms n
M1 = gamma(2*n+1);
M2 = 2^(2*n);
M3 = (gamma(n+1))^2;
M4 = (M1/(M2*M3))^2;
M5 = (sin(theta1/2))^(2*n);
M6 = M4*M5;
M7 = symsum(M6,0,inf);
T21 = T1*M7;
T2 = double(T21);
%Deviation from the small-angle approximation:
D2 = ((T2-T1)/T2)*100;

%Arithmetic-geometric mean method:
T3 = (4*T1)*((1+sqrt(cos(theta1/2)))^-2);
%Deviation from the small-angle approximation:
D3 = ((T3-T1)/T3)*100;

%STEP 4
table = [T1;T2;D2;T3;D3];
ylabel('\itVertical Range (m)','FontSize',14)
box on
grid on
set(gca,'FontSize',14,'FontWeight','Bold')
set(findall(gcf,'type','text'),'FontSize',14,'FontWeight','Bold')
hold on
pause(5)

%STEP 6
theta2 = linspace(theta,-theta,10);
for r = 1:length(theta2)
x(r) = L*sind(theta2(r));
y(r) = L-L*cosd(theta2(r));
set(f1,'xdata',[-L L],'ydata',[L L]);
set(f2,'xdata',[0 -x(r)],'ydata',[L y(r)]);
set(f3,'xdata',[-x(r)],'ydata',[y(r)]);
plot(-x(r),y(r),'.r')
M(r) = getframe;
end
xmax = max(x);
ymax = max(y);
disp(['Maximum horizontal displacement = ' num2str(xmax) ' m']);
disp(['Maximum vertical displacement = ' num2str(ymax) ' m']);
movie(M,-10)
```

Table A3. Script for water flowing out of a circular hole in a vertical cylindrical tank.

```
%Discharge of water from a vertical cylindrical tank (simulation)

%STEP 1
clear, clc
format short
disp(datestr(now,0))

%STEP 2
h = input('Enter the height (m) of the water tank, h = ');
hf = input('Enter the final height (m) of the water, hf = ');
d = input('Enter the diameter (m) of the water tank, d = ');
do = input('Enter the orifice diameter (m), do = ');
Cd = input('Enter the discharge factor, Cd = ');
g = 9.807; %Gravitational acceleration (m/s2)
tic

%STEPS 3-4
%Hand calculation
time = (1/Cd)*((d/do)^2)*((2/g)^0.5)*(sqrt(h)-sqrt(hf))
minute = fix(time/60)
second = (time/60 - minute)*60
toc

%Symbolic-based solution
%syms z
%A = pi*(d^2)/4;

%STEP 5
f1 = line([-do/2 -d/2],[0 0],'linestyle','-','color','k','linewidth',4);
f2 = line([-d/2 -d/2],[0 h],'linestyle','-','color','k','linewidth',4);
f3 = line([do/2 d/2],[0 0],'linestyle','-','color','k','linewidth',4);
f4 = line([d/2 d/2],[0 h],'linestyle','-','color','k','linewidth',4);
f5 = line([-d/2 d/2],[h h],'linestyle','-','color','b','linewidth',4);
%axis([-2*d 2*d 0 h]);
xlabel('\itHorizontal range (m)','FontSize',14,'FontWeight','Bold')
ylabel('\itHeight (m)','FontSize',14,'FontWeight','Bold')
axis equal
box on
grid on
set(gca,'FontSize',14,'FontWeight','Bold')
set(findall(gcf,'type','text'),'FontSize',14,'FontWeight','Bold')
hold on
tt = title(sprintf('Time: %0.4f sec', 0),'FontSize',14);
patch([-d/2 d/2 d/2 -d/2],[0 0 h h],'blue')
pause(5)

%STEP 6
t1 = linspace(0,time,1000);
for r = 1:length(t1)
K = (1/Cd)*((d/do)^2)*((2/g)^0.5);
h1(r) = (sqrt(h)-((t1(r)/K)))^2;
set(f1,'xdata',[-do/2 -d/2],'ydata',[0 0]);
set(f2,'xdata',[-d/2 -d/2],'ydata',[0 h]);
```

Table A3. (Continued.)

```
%ao = pi*(do^2)/4;
%Eq = -(A/(ao*Cd))*1/(sqrt(2*g*z));
%I1 = int(Eq,h,hf); %1000 mL filled
%time = double(I1)

%Numeric integration-based solution
%A = pi*(d^2)/4;
%ao = pi*(do^2)/4;
%Eq = @(z) -(A./(ao.*Cd)).*1./(sqrt(2.*g.*z));
%time = quad(Eq,h,hf)
%minute = fix(time/60)
%second = (time/60 - minute)*60
set(f3,'xdata',[do/2 d/2],'ydata',[0 0]);
set(f4,'xdata',[d/2 d/2],'ydata',[0 h]);
set(f5,'xdata',[-d/2 d/2],'ydata',[h1(r) h1(r)]);
fill([-d/2 d/2 d/2 -d/2],[0 0 h h],'w')
%patch([-d/2 d/2 d/2 -d/2],[0 0 h h],'white')
fill([-d/2 d/2 d/2 -d/2],[0 0 h1(r) h1(r)],'b')
%patch([-d/2 d/2 d/2 -d/2],[0 0 h1(r) h1(r)],'blue')
%patch('Faces',[1 2 3 4],'Vertices',[-d/2 0; d/2 0; d/2 h1(r);...
%-d/2 h1(r)],'FaceColor','blue');
set(tt, 'String', sprintf('Time: %0.4f sec', t1(r)));
M(r) = getframe;
end
```

Table A4. Script that accepts the length of a massless inextensible string as an input and returns three figures displaying the oscillation period, percent deviation, and horizontal displacement.

```
%Pendulum oscillation #2 (amplitude-related graphs)
clear, clc
format short
disp(datestr(now,0))
L = input('Enter the length (m) of the pendulum = ');
tic
g = 9.807; %Gravitational acceleration (m/s2))
%Period for the small-angle approximation:
T1 = (2*pi)*((L/g)^0.5);

theta = 1:1:90;
for r = 1:length(theta)
theta1(r) = theta(r)*pi/180; %Conversion to radian
%Horizontal distance from equilibrium position
x(r)=sind(theta(r))*L;

%Legendre series:
syms n
M1 = gamma(2*n+1);
M2 = 2^(2*n);
M3 = (gamma(n+1))^2;
M4 = (M1/(M2*M3))^2;
M5(r) = (sin(theta1(r)/2))^(2*n);
M6(r) = M4*M5(r);
M7(r) = symsum(M6(r),0,inf);
T21(r) = T1*M7(r);
T2(r) = double(T21(r));
%Deviation from the small-angle approximation:
D2(r) = ((T2(r)-T1)/T2(r))*100;

%Arithmetic-geometric mean method:
T3(r) = (4*T1)*((1+sqrt(cos(theta1(r)/2)))^-2);
%Deviation from the small-angle approximation:
D3(r) = ((T3(r)-T1)/T3(r))*100;
end

%Amplitude versus period
subplot(3,1,1)
plot(theta,T2,'-r','Linewidth',2)
hold on
plot(theta,T3,'-b','Linewidth',2)
legend('Legendre series','Arithmetic-geometric mean','Location','NorthWest')
xlabel('\theta_{0}: Amplitude (deg)')
ylabel('T: Period (sec)')
title(['Length: L = ' num2str(L) ' m, Period: T_{1} = ' num2str(T1) ' sec '])
box on
grid on
set(gca,'FontSize',12,'FontWeight','Bold')

%Amplitude versus period deviation from the small-angle approximation
subplot(3,1,2)
plot(theta,D2,'-r','Linewidth',2)
hold on
plot(theta,D3,'-b','Linewidth',2)
legend('Legendre series','Arithmetic-geometric mean','Location','NorthWest')
xlabel('\theta_{0}: Amplitude (deg)')
ylabel('Deviation percentage (%)')
title('Deviations from T_{1} (small-angle approximation period)')
box on
grid on
set(gca,'FontSize',12,'FontWeight','Bold')

%Amplitude versus horizontal displacement
subplot(3,1,3)
plot(theta,x,'-k','Linewidth',2)
legend('Amplitude vs. Horizontal displacement','Location','NorthWest')
xlabel('\theta_{0}: Amplitude (deg)')
ylabel('x distance from equilibrium (m)')
title('Horizontal displacement')
box on
grid on
set(gca,'FontSize',12,'FontWeight','Bold')
set(findall(gcf,'type','text'),'FontSize',12,'FontWeight','Bold')
toc
```

Table A5. Script that accepts as inputs the massless inextensible length L of the string, mass m of the bob, and initial angular displacement θ_0 and returns a plot of the variation of the potential energy and kinetic energy during the first half-cycle.

```
%Pendulum oscillation #3 (energy curves)
clear, clc
format short
disp(datestr(now,0))
L=input('Enter the length (m) of the pendulum = ');
m=input('Enter the mass (kg) of the bob = ');
%theta is the angle between the pendulum and
%the equilibrium line
theta=input('Enter the angular displacement (deg) = ');
tic

%Compute h, v, Ep, Ek and Et for theta
%theta is the angle between the pendulum and the
%equilibrium line on the pivot point (or fixed point)
%h(1) is the inital height from which the bob is released.

%For the position of theta, h(1) = L - L.cosd(theta)
%i.e., If theta = 0 deg, it means that the bob is on the
%equilibrium line and therefore h-initial will equal to zero.
%So that the potential energy will also equal to zero.

%Symmetrical oscillation
%theta1 includes the range of angles during the oscillation
%i.e., for an angular displacement of theta = 40 deg,
%range of the oscillation can be considered between
%-40 and 40 deg
%i.e., -40 -39 -38 ... equilibrium line (0) 1 2 3 ... 40
%Increment rate is considered as 1 degree.

%At the initial height of h(1), Epmax = m.g.h(1) = Et
and Ek = 0
%For any h, Et = Ep + Ek

%m.g.h(1) = m.g.h + (1/2).m.(v^2) and
%v = (2.g.(h(1)-h))^0.5 or v = sqrt(2.g.(h(1)-h))

g = 9.807; %Gravitational acceleration (m/s2))
theta1 = [-theta:1:0 1:1:theta];
for r = 1:length(theta1)
h(r) = L-L*cosd(theta1(r));
v(r) = sqrt(2*g*(h(1)-h(r)));
Ep(r) = m*g*h(r);
Ek(r) = (1/2)*m*(v(r).^2);
Et(r) = Ep(r)+Ek(r);
end

%Position, velocity and energy values
hmax = max(h);
vmax = max(v);
Epmax = max(Ep);
Ekmax = max(Ek);
table = [hmax;vmax;Epmax;Ekmax];
disp('Maximum values obtained during the oscillation of
pendulum:');
disp(' Height(m) Velocity (m/s) E_pot (J) E_kin(J)');
fprintf('%13.4f %12.4f %14.4f %10.4f \n',table);

%Plotting of energy curves on the same graph
plot(theta1,Ep,'-b','Linewidth',2)
hold on
plot(theta1,Ek,'-r','Linewidth',2)
plot(theta1,Et,'-k','Linewidth',3)

%Graphical details (legend, labels, titles, box and grid)
legend('Potential energy (Joule)','Kinetic energy (Joule)'...
 ,'Total energy (Joule)','Location','Best')
xlabel('\theta (deg)','FontSize',12)
ylabel('Energy (Joule)','FontSize',12)

title('Potential Energy - Kinetic Energy - Total
Energy','FontSize',12)
box on
grid on
set(gca,'FontSize',12,'FontWeight','Bold')
set(findall(gcf,'type','text'),'FontSize',12,'FontWeight','Bold')
toc
```

ORCID iDs

Kaan Yetilmezsoy https://orcid.org/0000-0003-1478-9957
Carl E Mungan https://orcid.org/0000-0001-7084-5402

References

[1] Daineko Y, Dmitriyev V and Ipalakova M 2017 Using virtual laboratories in teaching natural sciences: An example of physics courses in university *Comput. Appl. Eng. Educ.* **25** 39–47
[2] Magana A J and Silva Coutinho G 2017 Modeling and simulation practices for a computational thinking-enabled engineering workforce *Comput. Appl. Eng. Educ.* **25** 62–78
[3] Yetilmezsoy K 2017 IMECE—implementation of mathematical, experimental, and computer-based education: a special application of fluid mechanics for civil and environmental engineering students *Comput. Appl. Eng. Educ.* **25** 833–60

[4] Behringer E and Engelhardt L 2017 Guest editorial: AAPT recommendations for computational physics in the undergraduate physics curriculum, and the partnership for integrating computation into undergraduate physics *Am. J. Phys.* **85** 325–6
[5] Timberlake T and Hasbun J E 2008 Computation in classical mechanics *Am. J. Phys.* **76** 334–9
[6] ComPADRE 2018 *OSP Programming: Physics Pedagogy and Computer Science* (https://compadre.org/osp/webdocs/programming.cfm)
[7] Redish E F and Wilson J M 1993 Student programming in the introductory physics course: M.U.P.P.E.T *Am. J. Phys.* **61** 222–32
[8] Lee K C and Lee J 2007 Programming physics softwares in Flash *Comput. Phys. Commun.* **177** 195–8
[9] Sherin B L 2001 A comparison of programming languages and algebraic notation as expressive languages for physics *Int. J. Comput. Math. Learn.* **6** 1–61
[10] Zupančič B and Sodja A 2013 Computer-aided physical multi-domain modelling: Some experiences from education and industrial applications *Simul. Model. Pract. Th.* **33** 45–67
[11] Bozkurt E and Ilik A 2010 The effect of computer simulations over students' beliefs on physics and physics success *Procedia Soc. Behav. Sci.* **2** 4587–91
[12] Taub R, Armoni M, Bagno E and Ben-Ari M M 2015 The effect of computer science on physics learning in a computational science environment *Comput. Educ.* **87** 10–23
[13] Sarabando C, Cravino J P and Soares A A 2014 Contribution of a computer simulation to students' learning of the physics concepts of weight and mass *Procedia Technol.* **13** 112–21
[14] Mungan C E 2017 Optimizing the launch of a projectile to hit a target *Phys. Teach.* **55** 528–9
[15] Sarafian H 2011 A study of super nonlinear motion of a simple pendulum *Mathematica J.* **13** 1–16
[16] Wright J A, Bartuccelli M and Gentile G 2017 Comparisons between the pendulum with varying length and the pendulum with oscillating support *J. Math. Anal. Appl.* **449** 1684–97
[17] Marszal M, Witkowski B, Jankowski K, Perlikowski P and Kapitaniak T 2017 Energy harvesting from pendulum oscillations *Int. J. Non-Lin. Mech.* **94** 251–6
[18] Nelson R A and Olsson M G 1986 The pendulum-rich physics from a simple system *Am. J. Phys.* **54** 112–21
[19] Carvalhaes C G and Suppes P 2008 Approximations for the period of the simple pendulum based on the arithmetic-geometric mean *Am. J. Phys.* **76** 1150–4
[20] Çengel Y A and Cimbala J M 2006 *Fluid Mechanics Fundamentals and Applications* (New York: McGraw-Hill)
[21] Hicks A and Slaton W 2014 Determining the coefficient of discharge for a draining container *Phys. Teach.* **52** 43–7
[22] Noll R B and McElroy M B 1974 *Models of Mars' Atmosphere* NASA Goddard Space Flight Center, Greenbelt, Maryland: NASA Goddard Space Flight Center, SP-8010
[23] Attaway S 2009 *MATLAB: A Practical İntroduction To Programming and Problem Solving* (London: Butterworth-Heinemann)
[24] Nirmalakhandan N 2001 *Modeling Tools for Environmental Engineers and Scientists* (Boca Raton, FL: CRC Press)
[25] Hahn B and Valentine D T 2016 *Essential MATLAB for Engineers and Scientists* 6th edn (Cambridge, MA: Academic)
[26] Gustafsson F and Bergman N 2012 *MATLAB for Engineers Explained* (Berlin: Springer Science)
[27] Kalechman M 2008 *Practical MATLAB Basics for Engineers* (Boca Rato, FL: CRC Press)
[28] Moore H 2014 *MATLAB for Engineers: Global Edition* 4th edn (Harlow: Pearson Education)
[29] Harder D W 2018 *Matlab® Tutorial* https://ece.uwaterloo.ca/~dwharder/Matlab/
[30] The MathWorks Inc. 2018 *MATLAB vs. Python: top reasons to choose MATLAB* https://mathworks.com/products/matlab/matlab-vs-python.html?s_tid=srchtitle
[31] The MathWorks Inc. 2018 *Comparing MATLAB and R for prototyping and implementing analytics* https://mathworks.com/discovery/matlab-vs-r.html#apps
[32] Crawford J D 2010 *Doug Crawford's Visuomotor Neuroscience Lab (York University)* https://yorku.ca/jdc/Matlab/Lesson1.htm
[33] Gimenez A, Chausse V and Meseguer A 2015 Numerical continuation in classical mechanics and thermodynamics *Eur. J. Phys.* **36** 015015