8th International Scientific Conference
Technics and Informatics in Education
Faculty of Technical Sciences, Čačak, Serbia, 18-20th September 2020

Session 3: *Engineering Education and Practice*

Original paper
UDC: 004.42GNU Octave

Implementation of GNU Octave in a University Course of General Physics

Vera P. Pavlović [1*], Jelena T. Ilić [1]
[1] University of Belgrade, Faculty of Mechanical Engineering, Belgrade, Serbia
[*] vpavlovic@mas.bg.ac.rs

Abstract: *This paper presents some examples of the implementation of an open-source software GNU Octave, within a General Physics undergraduate course. The examples refer to writing a program in a Script Editor for cases that include: 1) solving equations* symbolically *for a two-body or a multi-body system, 2) different ways of restriction of a tabular and graphical display of obtained results to the ones that are physically logical for a given problem and 3) creation of animations which describe certain physical phenomena and processes. The coding was directed to prompt the user for specified values of various input parameters, after running the script, in order to interactively reach conclusions concerning the investigated physical dependences. The presented examples can be a good basis for the creation of small applets in the field of General Physics. Such an application of programming in the field of General Physics and other fundamental educational areas at technical faculties are an excellent pedagogical tool, enabling a better understanding of both the programming process and the phenomena in specific physical and technical fields.*

Keywords: *Octave; m-file*; *animation; function file; physics*

1. INTRODUCTION

GNU Octave software is a multifunctional computer tool featuring as a higher-level programming language for data processing in the form of matrices. Its mathematically-oriented syntax is good enough for the development of algorithms relevant for machine learning [1], yet sufficiently intuitive to be easily mastered. The mentioned software also contains convenient tools for data visualization (graphical and tabular display of results), with the possibility of further upgrades and build-in functionalities to adjust the effects for visualization and animation. Octave belongs to the open-source software, within the GNU project and it can be run on different operating systems such as Microsoft Windows, Linux, macOS, or BSD. The possibilities that it offers, especially with a number of additional packages from the entire Octave Forge collection (e.g.: Symbolic Package, I/O package, etc.) which are available on the Internet, make it quite comparable to certain commercially popular computer tools, in view of software content and quality. Therefore, Octave is one of the major free alternatives to software packages such as MATLAB, and in some segments even to software such as Scilab, Origin or FreeMat [2-5]. Octave GUI (Graphical User Interface) is developed to have a working environment analogous to the one in the MATLAB software package [1-7]. Despite the various tutorials related to Octave software [1-5], there is a lack of tutorials in which specific problems in the field of general physics are presented as examples. Therefore, having in mind the importance of correlation of teaching in informatics fields with specific examples from natural and technical subjects, not only in the higher years of undergraduate studies, but also within general basics in the first year of studies, some applications of the software GNU Octave that are related to examples in general physics are considered in this paper.

2. MAIN PRINCIPLES OF GNU OCTAVE

GNU Octave offers two basic modes of work. In the first mode, one can keep entering commands in the Command Window and Octave executes them immediately. In the second mode one can write series of commands and statements in the Editor Window and save them into a script file (a so-called m-file, due to the .m extension), executing the file later as a complete unit, which also includes writing functions and calling them. The basic data type in Octave is a matrix, which general format can be expressed as (mxn), where *m* is the number of rows and *n* is the number of columns of the matrix. All elementary operations are defined to support working with matrices. Vectors and matrices are used to store sets of values, all of which are the same type. Since defining vectors in Octave software involves creating one-dimensional arrays of *n* numbers ((1×n) or (n×1) matrices), an ordered one-dimensional array of possible (given)

values of any physical quantity can be considered in the Octave as a vector (row or a column vector), regardless of whether the given physical quantity is classified as a scalar (e.g.: time, coordinate, angle, energy, etc.) or as a vector physical quantity (e.g.: speed, acceleration, force, etc.). Appart from regular matrix algebra, where matrices should be conformable for a certain operation (e.g. for multiplication, division, or squaring) within the mathematical expression in the code, performing of element-by-element operations on matrices is possible, too. The matrix syntax, lists of basic arithmetic and relational operators, lists of special characters, special variables and constants, as well as the list of most built-in functions, are the same or very similar as in the MATLAB software package [1-7]. The same observation can be made for the typical decision making structures (including *if*, *else* and *elseif* commands) and various repetition control structures such as loop statements (e.g. *for* loop, or *while* loop) for managing the process of executing some part of a script file [1,6]. Regarding solving a system of equations symbolically, the shortest method, which usually gives the most simplified expressions as solutions, involves the use of the *solve* command. It implies the prior application of the *syms* function for the definition of a set of symbolic variables. The commands for differentiation with respect to a particular variable, as well as for performing integration, are of the same form within the Octave software and in MATLAB [1,6]. However, unlike MATLAB where the short command *table* provides an easy tabular display of results (e.g. values of directly and indirectly measured quantities), in Octave such a formatted display of results is possible only by applying several other (more complicated) methods, including the use of a couple of *fprintf* statements [1,6]. The function *fprintf* can also be used to save data from the created table, in the form of a *txt* file. Although most of the commands for plotting 2D/3D graphs and subgraphs, are the same in both mentioned software packages, there are still some differences. One of the advantages of Octave is that in one call of the *plot* command (in one command line) a significantly larger number of variables can be specified than in MATLAB, including a number of parameters for setting colors and other properties of lines and points on the graph. However, in MATLAB the additional so-called *set* commands may be necessary for a more detailed specification of the mentioned graph properties (e.g. for the linewidth, etc.). On the other hand, one of the shortcomings of the Octave software is the lack of an appropriate solution for adjusting an angle of axis labels on graphs, as well as the lack of an adequate solution for updating ("refreshing") a legend on the same graph after several consecutive runs of the given script file with changed input data. Commands for printing text on a graph, either by specifying coordinates for text positioning (via the *text* command), or by using the keyboard (via the *gtext* command and via clicking on the chosen spot in the created Figure), are completely analogous in both Octave and MATLAB and can be modified in order to "refresh" the text record on the graph, for each entering of new values for certain input variables [1,6].

3. METHODS

Octave version 5.1.0 and the Octave Symbolic Package 2.8.0 (Octave-forge-symbolic 2.8.0) were used to create examples in this paper [1]. The symbolic package was necessary for finding the general form of solutions for a system of equations (symbolic solutions). The command *solve* was used for these purposes. Two methods were used to define scalar variables. The first method involved application of an assignment operator (operator =) and an assignment statement, which defines a scalar variable as: a numeric value, a string (a separate sequence of characters placed between single quotes), a mathematical expression, or a function (e.g. by calling a specific function file). The second way refers to setting the input scalar variables via the command: *input*( ), where the appropriate string should be written in parentheses, inviting (prompting) the user to enter the value of a given variable after starting the execution of a written set of commands (after the *RUN* command). This value can generally be not only a numerical value, but also a certain mathematical expression, or a string which indicates the color, linestyle or marker on the graph, etc. Some independent variables were defined as a range of a values, i.e. as row vectors with equidistant elements (equally spaced elements), in cases when it was of interest to consider the output values of a dependent variable for a whole range of values of a given input variable. To define a vector with equally spaced elements the colon operator was mostly used to iterate through these values (e.g. t=first element: increment: last element), but the *linspace* function (t=linspace(first element, last element, number of elements)) was applied as well. Depending on the type of different input variables, operations on individual elements of given vectors (matrices) were used (when it was necessary) for the calculation of dependent variables, which mainly referred to the usage of operators such as:. *, or ./, or. ^. For the purpose of tabular presentation of results, formatting was performed using a specific block of *fprintf* functions. Within this, the visual appearance of the table was adjusted using the command '\t to separate the columns and the command \n' (new-line character) to display the following text in a new row. The formatting also included setting the name of the table and columns, as well as setting the number of potential digits and the number of decimals (using commands of the

form % 7.4f or% 6.3f, etc., where the mark *f* refers to the floats type of placeholder, the number after the dot refers to the number of decimals, and the number in front of the dot refers to the so-called field width i.e. to the number of potential digit places to be used in printing the results). Additional commands for writing tabular data to a txt file were also applied. For graphical representation of certain physical dependencies, the commands *plot* and *plot3* were used, with or without commands for subgraphs, depending on the needs in the considered task. The appearance of lines and points on the graph, the labels and other properties of the axes, as well as appearance of the text on the graph, were adjusted to spruce up the graph. Limiting the graphs in order to display only those values that have a physical meaning, as well as achieving an animation effect, was done in several ways, which is shown and discussed in the third section. The usage of *text* and *gtext* commands instead of the classic *legend* command, especially for the purpose of comparing multiple curves (lines and/or points) on the same graph, where the curves are obtained for different input values, is also discussed in the next section.

4. RESULTS

4.1. Example 1

The system shown in Figure 1a has been considered as a first example. The following assumptions were made: a friction between the body 1 and an inclined plane is present, the masses of the pulley and the thread are negligible, the thread is inextensible and the friction in the pulley is absent. Creation of a script file for this example has involved writing a multi-part script. The first part of the script has covered solving of the equation system, obtained by the application of Newton's second law in the direction of the system acceleration and in the direction perpendicular to it, for each of the two bodies in the system. Solving the problem assumed obtaining the solution algebraically, i.e. in symbolic form, for acceleration and tension in the thread, for the case of the direction of motion indicated in Fig. 1a. The intention was to present the solution in two forms: first, in unicode form, that looks like a usual handwritten equation, and second, in flat form, suitable for further use in calculations of actual numerical values in the Editor. The second part of the script was aimed at: a) calculation of the numeric values of the acceleration and thread tension, for certain values of independent variables, b) presentation of the results in a formatted table in the Command Window, and c) setting a condition that enables the occurrence of a special warning in the case of obtaining negative values of any elements of the row vector assigned to acceleration. The last mentioned fact implies that if any element of the acceleration row vector is negative, the notification that only positive values of accelerations have a physical meaning for the assumed direction of motion of the system, should appear in the Command Window. To achieve that goal, first, certain fixed numerical values were assigned to gravitational acceleration, the mass of the body 1 and the dynamic coefficient of friction, i.e. those variables were defined as scalars in Octave script. The angle of an inclined plane was defined within the certain range in degrees, as a row vector in Octave script, while the so-called *input* function was used to define the value of mass m_2 in kilograms. The third part of the script-file was related to the creation of a 2D graph, with two subgraphs, which presented the dependence of system acceleration and thread tension on the angle of the inclined plane, but only for those elements of the row vectors that corresponded to non-negative (≥0) elements of the row vector assigned to acceleration. Along with that, the intention was to prompt the user for an input of arbitrary color, linestyle and marker on the graphs after running the script, as well as to enable display of information about the new input values of m_2, on each subgraph. Instead of the *legend* function, which is not appropriately editable in Octave software on the same graph, in the case of multiple execution of a script file, an editable *text* command has been chosen, which enabled display of the newly entered value of variable m_2 (via *input* function) on a graph, in the selected (entered) marker color. Additionally, standard font size adjustments for tick labels, axes labels and subgraph titles, as well as adjustments of grid line appearance and *x*-axis range, in accordance with the specified range of angles of an inclined plane, were done. Fig. 2 shows a script that meets the above requirements. In order to enhance the visibility and the comprehensiveness of the commands, the colors in the displayed script are set according to the typical colors of MATLAB script. For the sake of simplicity of writing marks within the script, the sign *k* was used for the dynamic coefficient of friction in the script, while the usual symbol μ was entered only in the graph title. For the purpose of comparing the results obtained for different values of m_2, after repeated running of the script, the commands were adjusted to allow drawing of a new curve on the same graph for each new execution of the script file, i.e. for each change in the value m_2 defined via the *input* function. A rather complex form was chosen for the title of the graph, in order to demonstrate the possibilities to write the title in two lines. Fig. 1b shows graphs derived from the above script, for two different

inputs of the m_2 value.

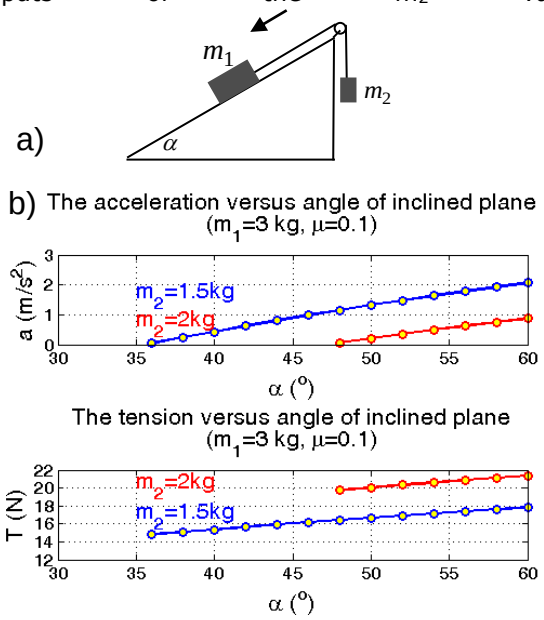

Figure 1. *a) Two-body system on an inclined plane; b) Two subgraphs which present the acceleration and the tensile force versus angle of inclined plane, respectively, for two inputs of the m_2 value.*

Since separation of the obtained results into those that are physically logical and those that are illogical is important from the aspect of application in physics, special attention has been paid to the various forms of conditioning within the script that enable display only of physically logical results for the assumed movement direction of the system. For that reason, several ways of the mentioned conditioning have been considered in this example. For instance, it can be noticed that instead of the conditionality defined by the underlined commands in the script in Fig. 2, usage of the following commands was also possible, in order to achieve the same effect:

alpha(a<0)=[ ]; T(a<0)=[ ]; a(a<0)=[ ]; (1)

If the mentioned conditioning (presented in the first or second form) were stated in front of the block of fprintf commands for tabular formatting, only the values corresponding to the non-negative acceleration elements ($a \geq 0$) would be seen in the table.

In order to obtain a graph only with dots, but in animated presentation, several changes in the script in Fig. 2 should be made. Namely:

- specification of the linestyle and linewidth would not be needed in the *input* command "UserLMC" and in the *plot* command, respectively;

- in front of the commands for the first subgraph, opening of the *for* loop should be performed, i.e.

```
disp 'PART 1 - Solving a system of equations symbolically'
y='a (acceleration (m/s^2))',    z='T (tension (N))'
syms g m1 m2 k alpha y z,      sympref display unicode
[y,z]=solve(m1*y==m1*g*sin(alpha)-k*m1*g*cos(alpha)-z,m2*y==z-m2*g,y,z)
sympref display flat,   y,z
disp ''PART 2 – Obtaining the values of acceleration and tension, for the given independent \
variables; formatted presentation of the results within the table in the Command Window ''
g=9.81,   m1=3
m2=input('Enter a value of m2 in the range from 1 to 2 kg: m2=   ')
k=0.1,    alpha=30:2:60;
disp 'The masses are in (kg), the acceleration is in (m/s^2), the angle is in degrees.'
a=g*(m1*(sind(alpha)-k*cosd(alpha))-m2)/(m1+m2);
T=g*m1*m2*(1-k*cosd(alpha)+sind(alpha))/(m1+m2);
fprintf('\tResults\n')
fprintf('\tAlpha (degree)\t\tAcceleration (m/s^2)\tTension (N)\n')
fprintf('\t%6.3f\t\t\t%6.3f \t\t\t %6.3f\n',[alpha;a;T])
if any(a(:)<0)     % if any(a(1,:)<0)
disp 'Only the obtained positive acceleration values are acceptable for the assumed motion direction!'
end
disp 'PART 3 – Creating a graph'
UserLMC=input('Enter a mark for the color, marker and linestyle on the graph:  ')
UserTC= input('Enter a text color label on the graph, the same as for the marker color: ')
alpha(a<0)=NaN;     a(a<0)=NaN;     T(a<0)=NaN;
subplot(211)
plot(alpha,a,UserLMC, 'linewidth',1.5,'markerfacecolor','y')
set (gca,'fontsize',14),    xlabel('\alpha (^o)','fontsize',18),    ylabel('a (m/s^2)','fontsize',18)
title({'The acceleration versus angle of an incline','(m_1=3 kg, \mu=0.1)'},'fontsize',16)
grid on,     xlim([30 60]),      hold on
text(35,0.8*max(a),['m_2=' num2str(m2)  'kg'], 'color',UserTC,'fontsize',18)
subplot(212)
plot(alpha,T,UserLMC, 'linewidth',1.5,'markerfacecolor','y')
set (gca,'fontsize',14),     xlabel('\alpha (^o)','fontsize',18),       ylabel('T (N)','fontsize',18)
title({'The tension versus angle of inclined plane','(m_1=3 kg, \mu=0.1)'},'fontsize',16)
grid on,    xlim([30 60]),       hold on
text(35,0.9*max(T),['m_2=' num2str(m2)  'kg'], 'color',UserTC,'fontsize',18)
```

Figure 2. *Part of the script file for example* 1

the following command should be entered: for i = 1: length (alpha)

- at the end of the script for the second subgraph, the so-called *pause* command that specifies the time interval between drawing adjacent points should be entered, as well as the command for closing the *for* loop;

- the plot command for the first subgraph should be changed in order to display: alpha (i) and a (i) (instead of alpha and a) and the analogue modification should be entered in the *plot* command for the second subgraph.

To avoid slowing down the animation, both text commands should be placed after closing the *for* loop. It would also be convenient to use the simplest forms of titles of the subgraphs, or to omit titles. It should be noted that instead of conditioning that are presented by the underlined commands in Fig. 2, or by the commands marked as equation (1), additional alternatives are possible. Namely, the analogous restriction could also be achieved through the *for* loop if the commands shown in Figs. 3a and 3b are used in front of the *subplot* commands. This would also imply the application of marks alpha(i), a(i) and T(i) within the further commands for plotting, as well as closing of the entire block for drawing of graphs via a double command *end* (or via *endif*, *endfor* commands).

a)
```
for i=1:length(alpha),  if a(i)>=0
```

b)
```
for i=1:length(alpha),   if a(i)<0
a(i) = NaN;  T(i)=NaN;   else
```

Figure 3. *The alternative parts of the script*

4.2. Example 2

Example 2a

In the second example, a 3D animation of a projectile motion trajectory (with no air resistance) in the Cartesian coordinate system (DCS) has been considered, where the projectile was launched at an angle θ from the origin of the coordinate system and motion was performed in the x-z plane. The intention was: 1) to prompt the user to enter the values for the initial velocity (in m/s) and for the launch angle θ (in degrees) within the limited range ($0<\theta<90$), as well as 2) to enable a tabular display of results in the form of a txt file and 3) to create an animated graph (Fig. 5a). In order to visually compare the results obtained for different values of υ_0 and θ, the commands that allow drawing of a new animation on the same graph for each running of the script, were included as well. The function *input*( ) was also used for choosing a marker color, marker symbol and marker size on the graph. The *rotate3d on* command was applied to enable manual rotation of a graph, in order to achieve different observation perspectives. Instead of the *legend* function, an editable *text* command was used to enable the appearance of labels of the input arguments on the graph in the selected marker color and to show "refreshed" values of the initial velocity and angle θ (entered after each running the script). The upper limit for the uniform row vector that defined time values, was set to the time of flight, according to the relevant equation. Saving a created graph, as an image in tiff or gif format, was also planned. Fig. 4 shows one of the forms of the relevant script.

```
g=9.81;  x0=0;  z0=0;
v0=input('Enter a value for initial velocity in the range 2-5 m/s:  ')
theta=input("Enter a launch angle of a projectile \
in the range from 30 to 60 degrees:   ")
tR=2*v0*sind(theta)/g;   % time of the flight
t=0:0.01:tR;

v0x=v0*cosd(theta);   v0z=v0*sind(theta);
x=x0+v0x*t;      z=z0+v0z*t-(1/2)*g*t.^2;      y=0;
UserMC=input("Enter a mark for the marker color  \
and marker type on the graph:   ")
UserMS=input('Enter a mark for the marker size on the graph:    ')
UserTC=input ("Enter a mark for the color of the text on the graph,\
the same as the marker color:  ")

file1 = fopen('Results.txt', 'w');
fprintf (file1,'\tResults:\n');
fprintf (file1,'\tTime\t\tx coordinate\tz coordinate\n');
fprintf (file1,'\t%6.3f \t\t%6.3f\t\t%6.3f\n', [t; x; z]);
fclose (file1);

for i=1:length(t)
plot3 (x(i),0,z(i), UserMC, 'markersize',UserMS),      hold on,
if i==1
set (gca,'fontsize',12),     xlabel ('x coordinate [m]', 'fontsize',14)
ylabel ('y coordinate [m]', 'fontsize',14)
zlabel ('z coordinate [m]', 'fontsize',14)
title ('3D animation of a projectile motion trajectory', 'fontsize',16)
grid on,     ylim([-1 1])
text (0.01,0.7,max(z),['v_0='num2str(v0)'m/s,\theta='num2str(theta)'^o'],
'color',UserTC)
end
pause(0.01),   end
rotate3d on,        print(figure(1), 'Figure_projectile motion.tiff')
```

Figure 4. *The script file for example* 2a

Example 2b

The case of simultaneous movement of two projectiles is more interesting than the previous one. In this case, the trajectory of each projectile is shown as an animated line, in the color picked for the projectile (Fig. 5b). The movement of projectiles starts at the same time, but the time of flight is different, due to the different initial arguments (initial velocity and launch angle) which can be entered by the user after running the script. The scaling of coordinate axes remains fixed during the animation. An editable *text* command was used in the same way as presented in a previous example. This example implies the usage of the majority of commands presented in Fig. 4, but individually for each of the 2 projectiles. However, it is necessary to introduce numerous new commands as well. For instance, in order to achieve the effect of animated drawing of a line, it is necessary to define a row vector assigned to the *y* coordinate of each projectile (zero vector, of the same length as vectors t and z). Since row vectors t_1 and t_2 are generally not of the same length, and since it is not possible to know in advance which length will be the longer one, the iteration from 1 to n1=min([length(t1); length(t2)]), and subsequent iteration from n1 to n2=max([length(t1); length(t2)]), within the one

unified *for* loop (by application of *if* and *elseif* - content of *h=plot3( )* command in front of the first

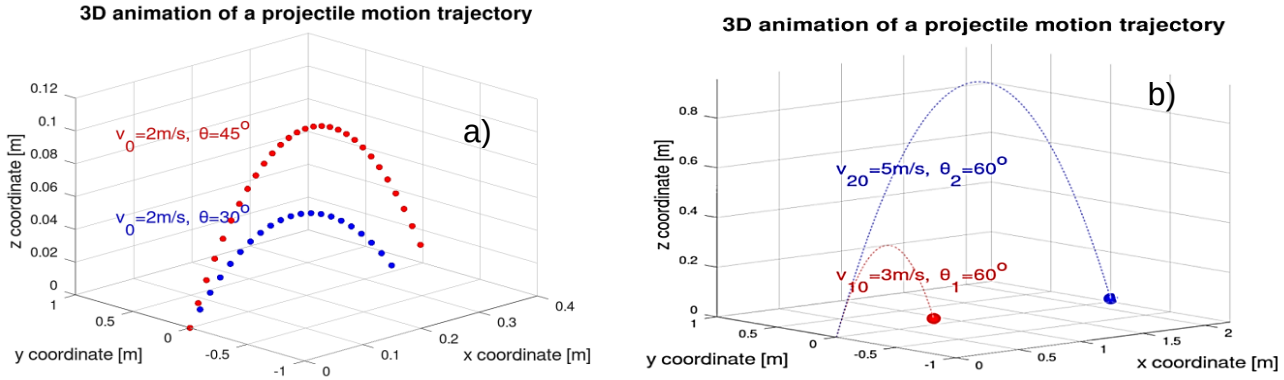

Figure 5. *Presentation of two approaches (a and b) to 3D animation of projectile motion trajectory. The left graph (a) results from the two subsequent runns of the same code. The right graph results from the one single running of the more complex code. On both graphs different colors correspond to different sets of initial arguments (v_0 and θ).*

```
UserC1=input ('Enter a mark for the color of the first projectile on the graph:   ')
UserC2=input ('Enter a mark for the color of the second projectile on the graph:   ')
n1=min([length(t1); length(t2)]);   n2=max([length(t1); length(t2)]);
x_low=min([x1(:); x2(:)]);      x_high=max([x1(:); x2(:)]);
z_low=min([z1(:); z2(:)]);      z_high=max([z1(:); z2(:)]);
figure
h=plot3(x1(1),y1(1),z1(1),UserC1,x1(1),y1(1),z1(1),'o',x2(1),y2(1),z2(1),UserC2,x2(1),y2(1),z2(1),'o');
set (gca,'fontsize',12),    xabel('x coordinate [m]','fontsize',14),    ylabel('y coordinate [m]','fontsize',14),
zlabel ('z coordinate [m]','fontsize',14),  title ('3D animation of a projectile motion trajectory','fontsize',16),  grid
on,   axis([x_low x_high -1 1 z_low z_high]);     hold on
for i=1:n1;
set(h(1), 'xdata',x1(1:i), 'ydata',y1(1:i), 'zdata',z1(1:i),'linestyle','--')
set(h(2), 'xdata',x1(i), 'ydata',y1(i), 'zdata',z1(i),'color', UserC1,'MarkerFaceColor',UserC1,'markersize',12)
set(h(3),'xdata',x2(1:i),'ydata',y2(1:i),'zdata',z2(1:i),'linestyle','--')
set(h(4), 'xdata',x2(i), 'ydata',y2(i), 'zdata',z2(i),'color', UserC2,'MarkerFaceColor',UserC2,'markersize',12)
pause (0.01),   end
for i=n1+1:n2;
set(h(3),'xdata',x2(1:i),'ydata',y2(1:i),'zdata',z2(1:i),'linestyle','--')
set(h(4),'xdata',x2(i),'ydata',y2(i),'zdata',z2(i),'color',UserC2,'MarkerFaceColor',UserC2,'markersize',12)
pause (0.01),   end
```

Figure 6. *Part of the script file for example* 2b

commands, which is not presented here) or within the two separated *for* loops (Fig. 6), can be a good solution for the achievement of the required animation on the 3D graph. After the last command in Fig. 6, commands for writing text on a graph (instead of a legend) would follow, accompanied with commands for manual rotation of a graph, for saving a graph as an image, etc.

In the commands related to animation of a graph in example 2b (Fig. 5b), it is especially important to pay attention to:

- the fact that writing a function handle for a plot function, as well as the adjusting of graph axes and title out of the *for* loop (in front of the loop), enables higher speed of the animation;

- the way of achieving fixed scaling on the axes during the animation, because this must provide visibility of simultaneous drawing of two trajectories on the same graph, where the range and maximum height of the projectile depend on the value of the initial velocity and launch angle at each new running the script;

for loop, and the way of adjusting this command through the loops.

Such a combination of commands enables that during the flight of the first projectile simultaneous drawing of the animated trajectories of both projectiles could be seen, while further animation implies continued drawing of the second projectile trajectory with the retained appearance of the trajectory of the first projectile.

4.3. Example 3

Example 3 refers to the creation and application (calling) of a function file, in tasks where projectile motion is considered. An example of a functional m-file that should enable calculation of the projectile horizontal range in an arbitrary task (for the launch from the origin of the coordinate system) is presented in Fig. 7. The chosen function name and the function file name is: f_Range_ProjectileMotion. The input variables in this function are the initial velocity, the angle of ejection in degrees and the gravitational acceleration. After calling this function in the

arbitrary script file (via the command: xR=f_Range_ProjectileMotion(v0,theta,g)), the function accepts specific numerical values related to its input variables (given in the form of scalars or vectors), processes them and generates numeric values of the projectile range.

```
function
xR=f_Range_ProjectileMotion(v0,theta,g)
xR=(sind(2*theta).*v0.^2)/g;
end
```

Figure 7. *Part of the script*

Thus, if in an arbitrary task the launch angle is given as a row vector, while the value of the initial velocity is defined via the *input* function, the commands shown in Fig. 8 could be used to determine the launch angle at which the projectile reaches the maximum range.

```
xRmax=max(xR);   for i=1:length(xR),
if  xR(i)==max(xR)
theta_xRmax=theta(i);   end,   end
```

Figure 8. *Part of the script*

The function analogue to the one presented in Fig. 7 (with the same input variables) can be written for the time of flight as well, where the body of the function would be defined via the time of flight equation.

```
tf=input('Enter an integer value of the upper limit of the time interval, in the range 7 - 10 s: tf=  ')
t=0:0.1:tf;
tB=tf/2    % the same as: tB=max(t)/2
th=0:0.1:tB;    n=length(th)
a=input('Enter an acceleration value in the range from 0.4 to 2 m/s^2: a=  ')
% The following equations can be applied to the motion in the first half of total time
 v=a*t;      x=a*(t.^2)/2;      y=zeros(1,length(t));
 xB=a*(tB.^2)/2;    vB=a*tB;
 a1=a*ones(1,length(th));   a2=-1*a*ones(length(th),length(t));
      figure
      subplot(211)
      h1=plot(t(1),v(1),'b-',t(1),v(1),'c-');
      set(gca,'fontsize',14),       xlabel('time (s)','fontsize',18),      ylabel('velocity (m/s)','fontsize',18)
      xlim([0  tf])   % or xlim([0 max(t)])
      ylim([0  1.5*vB])
      line('xdata',[tB,tB],'ydata',[0,1.5*vB],'linestyle','--','linewidth',1)
      grid on,      hold on

      subplot(212)
      h2=plot(t(1),a1(1),'b-', t(1),a2(1),'c-');
      set(gca,'fontsize',14),                          xlabel('time (s)','fontsize',18),
      ylabel('acceleration (m/s^2)','fontsize',18),          xlim([0   tf]),    ylim([-2*a  2*a])
      line('xdata',[0,tf],'ydata',[0,0],'linewidth',1)
      line('xdata',[tB,tB],'ydata',[ -2*a,2*a],'linestyle','--','linewidth',1)
      grid on,      hold on
disp 'Select a point for the tB label on the first subplot, using a keyboard'
for i=1:n
set(h1(1),'xdata',t(1:i),'ydata',v(1:i),'linewidth',3)
set(h2(1),'xdata',t(1:i),'ydata',a1(1:i),'linewidth',3)
pause(0.01),    end
% The following equations can be applied to the motion in the second half of total time
v=vB-a*(t-tB);      x=xB+vB*(t-tB)-a*((t-tB).^2)/2;  y=zeros(1,length(t));
for i=n:length(t)
   set(h1(2),'xdata', t(n:i),'ydata', v(n:i),'linewidth',3)
   set(h2(2),'xdata',t(n:i),'ydata',a2(n:i),'linewidth',3)
pause(0.01),    end
gtext({'t_B'},'fontsize',14)
```

Figure 9. *Part of the script file for example* 4

4.4. Example 4

The fourth example deals with the way of creating an animation in which:

- in the upper subgraph, an animation effect of the time dependence of the velocity is synchronized with the motion of particle allong x axis, where the motion is at first uniformly accelerated and then slowed down (with the same absolute value of acceleration);

- in the lower subgraph an animation effect of the time dependence of the acceleration can be seen.

The script file includes the *input* function, to prompt the user to enter the acceleration value and the upper limit (t_f) of the row vector assigned to time values in a certain range. The additional commands for auxiliary lines on both subgraphs are also included. The graph that is presented in Fig. 10 is

the snapshot made at the end of the animation, in the case when the values a=1 m/s^2 and t_f=10 s were entered after running the script.

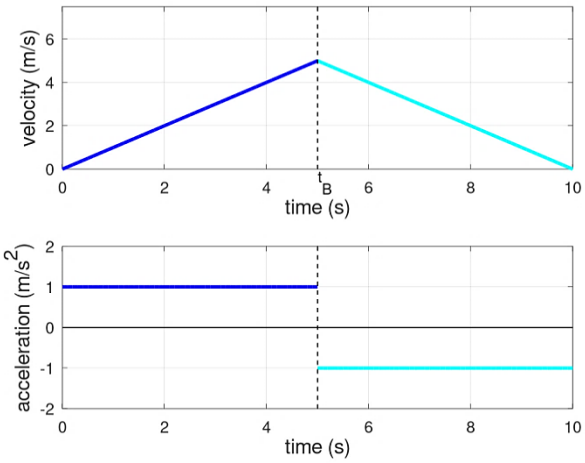

Figure 10. *Final snapshot of the animated graph related to accelerated/decelerated motion of the particle*

Fig. 9 shows the most important parts of the script used to create this animation. All commands, except of the couple of the so-called *se*t functions that define more specified way of performing plot function, are placed out of the loops, in order to speed up the animation. If the ordinate axis range is not limited for the lower subgraph, usage of the *gtext* command that should replace the legend on the graph, could be more convenient than text command. Utilization of the *gtext* command is also particularly convenient in examples such as 2a, because due to different values obtained on the ordinate (with different input values) it is not easy to predict the coordinates required for an appropriate text position.

5. CONCLUSION

The paper presents some examples of the application of an open-source GNU Octave software within the general physics course in undergraduate studies. GNU Octave is a multifunctional computer tool designed to process data in the form of matrices and is a major alternative to commercial software packages such as MATLAB. To demonstrate some of the possibilities of applying programming via Octave software within the undergraduate course of general physics, the creation of script files in the examples from the field of mechanical motion has been chosen in this paper. The examples have included: finding the general form of solutions for a system of equations (symbolic solutions), tabular formatting of results for a specific set of values of input variables, saving formatted results in the form of ASCII files, as well as 2D and 3D graphical display of results with animation effects for certain physical phenomena and processes. A brief glance at the creation and application of function files is also given. The importance to create script files that prompt the user to enter new values of various input parameters after each starting of the execution of a given script, in order to compare the changes on the graphs, and to reach a better understanding of certain physical relations and laws, was emphasized as well. Special attention has been paid to set various types of conditions that enable showing only results that have a physical meaning for the considered physical problem in the table and/or graph. Additionally, application of a decision making structure had been also used, for achieving targeted occurrence of warning notes in a case of obtaining physically illogical solutions of certain equations, as well as for performing animation according to different kinematic equations during the different time intervals. The animations and emphasized observations presented in this paper are methodologically important examples of developing and correlating knowledge in the field of informatics and general physics. The considered examples also represent a good educational basis for creating applets in general physics and related teaching areas.

ACKNOWLEDGEMENTS

This work was financially supported by the MESTD of the Republic of Serbia under the contract 451-03-68/2020-14/200105.

REFERENCES

[1] Eaton, J. W., Bateman, D., Hauberg, S., Wehbring, R. (2019). *GNU Octave, Edition 5 for Octave version 5.1.0*

[2] Stahel, A. (2019). *Octave and MATLAB for engineers*, Bern University of applied sciences, Switzerland.
https://web.sha1.bfh.science/Labs/PWF/Documentation/OctaveAtBFH.pdf

[3] Arras, K. (2009). Octave/Matlab tutorial, *Social Robotic Lab*, University of Freinburg. http://srl.informatik.uni-freiburg.de/downloadsdir/Octave-Matlab-Tutorial.pdf

[4] Linge, S., Langtangen, H. P. (2016). *Programming for computations – MATLAB/Octave, a gentle introduction to numerical simulations with MATLAB/Octave*, Springer Open.

[5] Quarteroni, A. and Saleri, F. (2010). *Scientific computing with MATLAB and Octave*, Second edition, Springer International edition, Springer.

[6] MATLAB numerical computing, Tutorials point – simply easy learning, 2014 Tutorials Point (I) Pvt. Ltd., www.tutorialspoint.com

[7] Pratap, R. (2010). *Getting started with MATLAB: a quick introduction for scientists and engineers*, Indian edition, Oxford University Press.