



Journal of Applied and Computational Mechanics



Research Paper

Development of an Educational Code of Deriving Equations of Motion and Analyzing Dynamic Characteristics of Multibody Closed Chain Systems using GNU Octave for a Beginner

Yonghui Park 

Department of Mechanical Engineering, Yuhan University, 590, Gyeongin-ro, Bucheon-si, Gyeonggi-do, Republic of Korea

Received July 19 2021; Revised October 13 2021; Accepted for publication October 13 2021.

Corresponding author: Y. Park (yonghuiPark216@gmail.com)

© 2022 Published by Shahid Chamran University of Ahvaz

Abstract. In this study, an automatic GNU Octave code, a free high-level language, for the educational purposes was developed to derive equations of motion and constrain equations of a multibody closed chain system and to calculate the response of the system. The code for calculating the dynamic response was developed by formulating several equations in symbolic expression and extracting differential-algebraic equations in matrix form. The code has a similar structure to the previous code for the open chained system, but it deals with the constraint equation and different numerical integration. The examples of closed chain systems provide an additional procedure to derive the constraint equations by using Lagrangian multiplication theory and to solve the differential-algebraic equations using the Runge-Kutta method. The code was made to understand the theory of analysis and the structure of calculation easily. In addition, the code has an automatic process of the derivation of the Lagrange equation and the constraint equations in matrix form after inputting the number of symbolic information such as position and velocity coordinates and design variables of the system that the user wants to review. The code was validated by comparing the dynamic response of the four-bar linkage with the same design variables and initial conditions of the previous work. By using the code, the reader's ability to exchange information such as symbols and matrices will be expected to be improved.

Keywords: GNU Octave, Multi-body dynamics, Closed chain, Lagrange multiplier, Differential Algebraic Equation, Automation.

1. Introduction

Multibody dynamics analysis deals with the dynamic characteristics of a multibody system. It has been developed mainly in machinery, aerospace, and robots. The mechanical field dealing with closed-chain systems has attracted much interest in deriving the optimal design through the sensitivity analysis between design variables and dynamic characteristics in the design process. In comparison, the aerospace field dealing with open-chain systems focuses on research on the safety of the system due to its enlarged range of motion [1]. Research on multibody dynamics analysis has focused on improving the deriving equations of motion, expanding the range of analysis, deriving efficient computational algorithms by improving hardware computational performance, and improving the software for user convenience. About the kind of multibody dynamics software, overseas software such as ADAMS, MESA VERDE, and NEWEUL have been commercialized, and RecurDyn and DAFUL programs have been commercialized in Korea [2]. In this way, multibody dynamics has become an essential discipline for the design process by calculating dynamic characteristics such as velocity, acceleration, and other essential physical quantities to check the system has on proper operating condition as we designed. Even though the activity is considered as an essential process in the design process from system-level in large companies dealing with system development and assembling, small and medium-sized parts manufacturers that manufacture mechanical elements and parts only conduct at the level of simple strength and stiffness evaluation without checking the dynamic characteristics. It has not been able to accumulate engineering capabilities from the system level. In other words, there were not enough opportunities for learning multibody dynamics analysis, and they could not set up the application direction on how to use the analysis in their engineering work. In particular, in a situation where it is difficult to secure a market continuously by selling a product from the element-level in the large companies due to an extreme competition among lots of the small and medium-sized companies, product development in relying on the specific large companies, can no longer be competitive. Based on engineering skill-up, they need to expand the market by giving sufficient engineering information on how the product can be effective in your assembled product. From this point of view, multibody dynamics can play a very good role in enhancing the ability to look at the design from the system's point of view. Accordingly, it is necessary to develop a free program that can be partially used by small and medium-sized parts makers and prospective workers to understand the basic theory of multibody dynamics and apply the analysis in production activities.



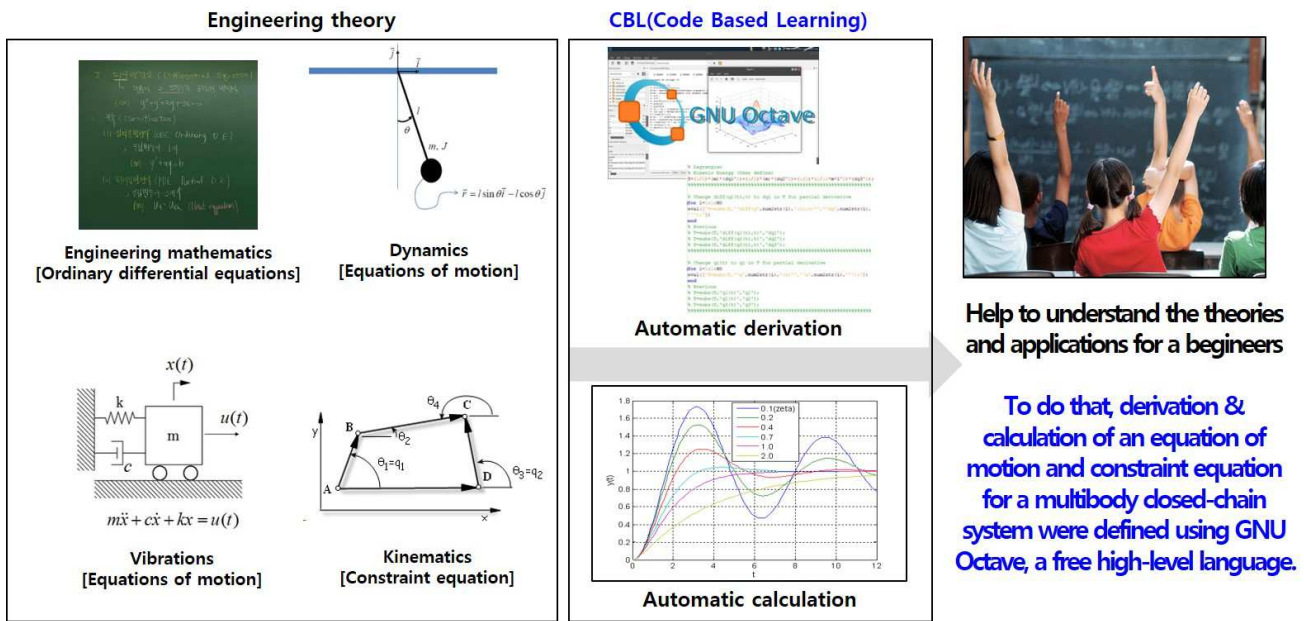


Fig. 1. The concept of the automatic code

However, several commercial software that can automatically induce equations of motion and calculate dynamics has already been developed [3], and a theoretical study on how to derive dynamics models for efficient control in the robot industry [4-7] has been conducted, so the necessity of program development is somewhat ambiguous. In particular, recently, a MATLAB package program was developed that automatically derives the linear equation of motion of a robot and calculates the dynamic characteristics using the TMT method [8-9]. Moreover, the educational MATLAB software was already provided for graduate and undergraduate students by exercising a closed-chain 3D robot and McPherson car suspension system [10]. The students in courses of 12 hours of theory and 15 hours of personal work acquired usage skills to solve several real exam exercises. However, we need the free-software with sufficient explanations such as how to derive an equation, how to apply the equation to a code's function, how to convert a symbolic expression to numeric expression, for engineers for small businesses and students that have no idea of dynamics. The previous work developed a dynamics analysis GNU Octave code for multibody dynamics analysis, so various part manufacturers that produce traditional mechanical elements such as fastening, power transmission, and shafts can cultivate design capabilities from a system perspective [11]. Using the GNU Octave program and the symbolic package, the code for deriving the equation of motion and transforming the matrix of the multi-DOF system was developed, and the code for deriving a numerical solution using the Newmark integration method was combined. The definition of the Lagrangian equation, derivation of the equation of motion, and the process of extracting the mass-damping-stiffness matrix and force vector were shown in the code to induce an understanding of the mechanism of dynamic analysis. This code cannot apply a multibody open chain system only, therefore a range of applications should be expanded.

In this study, an automatic GNU Octave code for the educational purposes was developed to derive an equation of motion and a constraint equation of a multibody closed chain system and to calculate the response of the system (Fig. 1). The code for calculating the dynamic response was developed by extracting the differential-algebraic equation in the form of a matrix. Unlike the existing open system mechanical system dynamic characteristics analysis code [11], the matrix derivation process by adding a constraint equation is presented as a code. In addition, after inputting the symbolic coordinates and design variables of the system to be reviewed by the user, a code was developed to automatically process the Lagrange equation and the constraint equation in matrix form.

2. Theoretical Background

2.1 Differential-algebraic Equation

The coordinates of the objects constituting the open system mechanical system are independent variables, and the Lagrangian (Eq. (1)) is defined in consideration of the kinetic energy of the system and the conserved or non-conserved energy according to the relative motion between the coordinates. Eq. (2) is based on the principle that an object passes through a path where the integral value of the time difference between kinetic energy and potential energy is the minimum among all paths when an object is confined for a specific time and moves from one point to another. With Eq. (2), the translational-rotational equation of motion (Eq. (3-4)) is in the form of a second-order differential equation [12].

$$L = T - V \tag{1}$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = \frac{d}{dt} \frac{\partial (T - V)}{\partial \dot{q}} - \frac{\partial (T - V)}{\partial q} = Q \tag{2}$$

$$F = M\ddot{x} + C\dot{x} + Kx \tag{3}$$

$$T = I\ddot{\theta} + C_t\dot{\theta} + K_t\theta \tag{4}$$

However, in a closed chain system that repeatedly performs a certain motion, it must be considered when calculating the response as a constraint exists between objects and mutually affects the motion (Eq. (5)). In other words, it is necessary to extract the optimal value that satisfies the constraint condition from the response candidates for each hour in Eqs. (3-4) [13]. The



Lagrange multiplier method finds the extreme value of a multivariate function when there is a constraint. It finds the extreme value that satisfies both the multivariate function and the constraint. The tangent of the two functions is parallel [14]. In defining the equation of motion (Eq. (6)), the virtual work due to the virtual displacement δq in the state where no actual motion has occurred can be defined as in Eq. (7). The partial differentiation of the constraint equation Φ concerning the virtual displacement and the virtual displacement must satisfy Eq. (8). The equation of motion in which the Lagrangian multiplication theory is introduced can be derived as in Eqs. (9-10).

$$\Phi = 0 \tag{5}$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} - Q = 0 \tag{6}$$

$$\left(\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} - Q \right) \delta q^T = 0 \tag{7}$$

$$\Phi_q \delta q = 0 \tag{8}$$

$$\left(\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} - Q \right) \delta q^T - \lambda \Phi_q \delta q = 0 \tag{9}$$

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = Q + \Phi_q^T \lambda \tag{10}$$

By deriving the equation of motion from Eq. (10) and applying the Lagrangian multiplier by obtaining the acceleration analysis equation from the constraint equation, we can get the differential-algebraic equation (Eq. (11)). To satisfy the reliability of solutions, the constraint violation stabilization method, coordinate partitioning method, the hybrid method can be used by changing $-(\Phi_q \dot{q})_q \dot{q} - 2\Phi_{qt} \dot{q} - \Phi_{tt}$. However, this issue is the kind of selecting a numerical method that gives a numerical solution more accurately, we just consider the fundamental formulation of the equation of motion, Eq. (11).

$$\begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} Q \\ -(\Phi_q \dot{q})_q \dot{q} - 2\Phi_{qt} \dot{q} - \Phi_{tt} \end{bmatrix} \tag{11}$$

In this way, we need much effort to understand the Lagrangian multipliers and other theories underlying Lagrange mechanics. However, there is no great difficulty in understanding the process of mechanically inducing the equations of motion by using them. The developed code contains contents that can automatically process the equations of motion and constrained equations using the above equations. Just given the system, if we have the number of degrees of freedom and the generalized coordinate system, and the kinetic energy, potential energy, and constraint equation can be extracted and entered into the code, we derive the differential-algebraic equation easily.

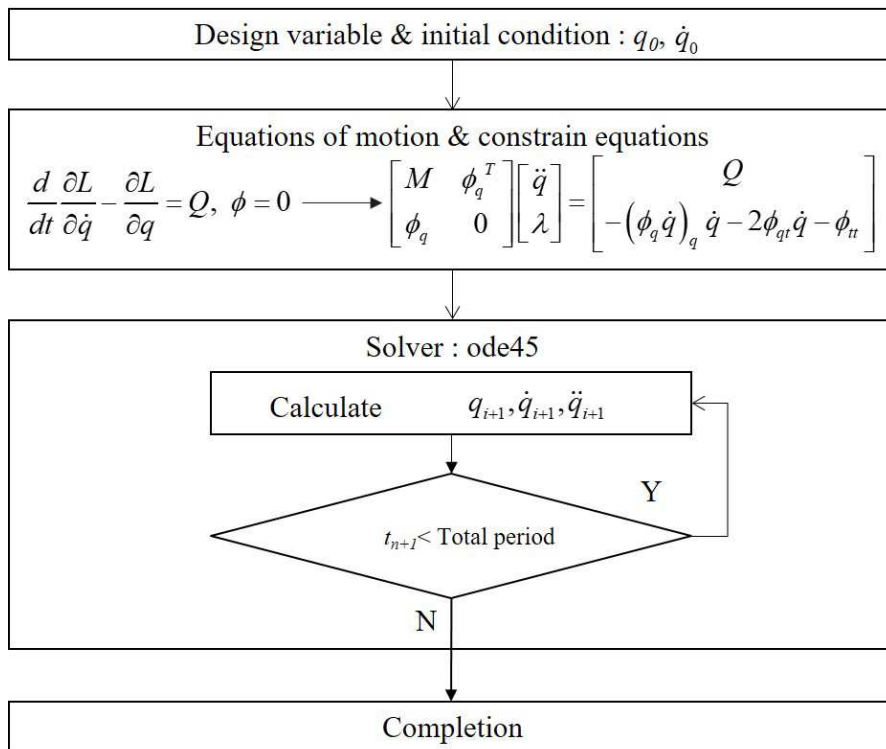


Fig. 2. Flow of the numerical integration



2.2 Numerical integration

To calculate the response of a closed chain system, we need a numerical integration for solving differential-algebraic equations for generalized coordinates. In the open-chain system mechanical system response, the previous work uses the Newmark numerical integration method to calculate the displacement, velocity, and acceleration of each time when a specific unit time passes sequentially based on the assumptions about the time integration method, such as the average acceleration method and the constant acceleration method [11]. However, the Newmark numerical integration method cannot be used as a solver because we need additional iterative calculations to derive an optimal solution that satisfies the algebraic equation, including the Lagrangian multiplier in the differential equation. Accordingly, we chose the ode solver in GNU Octave to get the response. The ode45 solver is based on the Runge-Kutta method to reduce the cutting error by considering the higher-order term of the Taylor series used for calculating the differential value. During iterative calculations, the code tries to find the optimal solution by discriminating the elements by the algebraic equation [15-16].

Since the whole equations in the form of a matrix are applied to the solver through the calculation procedure in Fig. 2, it is not a big problem in applying the numerical integration method. However, it is essential to understand the analysis process. Suppose user can define the number of degrees of freedom, the coordinates that are required to express the system response of a closed chain system, and the equations of motion and the constraint equations; In that case, the code prepares them as a function, matrix, and vector to formulate them in the ode45 solver.

3. Code

3.1 System definition

Fig. 3 is a single pendulum selected to explain the use of the automatic calculation code. One end is placed on pin support and is under gravity. To present the motion of a single pendulum with mass m and length l in the two-dimensional plane, there must be three coordinates x_G, y_G, θ in the Cartesian coordinate system to express translational and rotational motion, and the degree of freedom is three. In Fig. 3, the rotational displacement θ of the single pendulum supported by the pin can be expressed as a relative motion according to the rotational motion coordinate θ at the center. If the system does not have pin support, the Lagrangian L can be easily obtained after deriving kinetic energy (Eq. 12) and potential energy (Eq. 13) without any constraints.

$$T = \frac{1}{2} m \dot{x}_G^2 + \frac{1}{2} m \dot{y}_G^2 + \frac{1}{2} \frac{1}{12} ml^2 \dot{\theta}^2 \quad (12)$$

$$V = mgy_g \quad (13)$$

However, dependent coordinates occur in the closed chain system, not independent coordinates, interrelating with movements between objects or coordinates within objects. The constraint equation is to organize the correlation between these coordinates, and the remaining independent coordinates, excluding all dependent coordinates, are referred to as generalized coordinates. The x_G and y_G coordinates depend on θ through the constraint equations (Eqs. (14-15)) in the single pendulum.

$$x_G = \frac{l}{2} \cos \theta \quad (14)$$

$$y_G = -\frac{l}{2} \sin \theta \quad (15)$$

Regarding the generalized coordinates, there is no need to go through an intermediate process if anyone defines the independent generalized coordinates of the closed chain system at once. However, it is not easy to omit them in practice. Accordingly, we need to deduce the kinetic energy and potential energy for all coordinates by considering the closed chain system as an open-chain system. We then additionally derive the constraint equations between each coordinate. Lastly, we need to derive it as the differential-algebraic equations in Eq. (11).

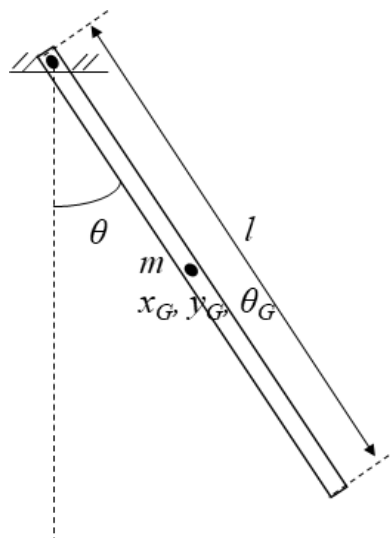


Fig. 3. Single pendulum



<pre> clc clear all close all pkg load symbolic %Part 1-----% %% Derivation of the constraint equation % Number of degree of freedom and constraint equation ND=3; NC=2; unconservation_MakingEquation save temp_dtuc.mat NC ND dtuc Tvar clear all load('temp_dtuc.mat') MakingEquation </pre>	<pre> % Mass matrix and force vector [M]=mmass(Eq,ND,Tvar); [Generalized1]=mstiff(Eq,ND,Tvar); % Gravity force vector Gravityterms [Gravity]=mgravity(dgv,ND,Tvar); save Matrix.mat M Generalized1 Gravity %Part 2-----% %% Derivation of the constraint equation ConstraintEquation [Gamma,Qq]=mconst(Qq,Gamma,ND,Tvar); %Part 3-----% %% Numerical integration Numerical analysis DAE </pre>
---	--

Fig. 4. msain_script

<pre> pkg load symbolic % Symbolic expression about coordinates, velocities, and % design variables (User define) for i=1:1:ND eval(['syms q',num2str(i),' dq',num2str(i)]) end % Previous code % syms q1 q2 q3 ... % coordinates % dq1 dq2 dq3 ... % velocities syms m l ... % mass, length t g ... % time, gravity %***** % Tvar=[time, coordinates, velocities, the others] % for defining all functions regarding 'Tvar' automatically % and for using ode45 solver Tvar=[t]; for i=1:1:ND eval(['Tvar=[Tvar q',num2str(i),'];']) end for i=1:1:ND eval(['Tvar=[Tvar dq',num2str(i),'];']) end Tvar=[Tvar m l g]; %***** % Define temporary coordinates with the time variable for i=1:1:ND eval(['tmp',num2str(i),'=[char(q',num2str(i),' ''(t)'';')]) end %Previous %tmp1=[char(q1) '(t)']; % %tmp2=[char(q2) '(t)']; % %tmp3=[char(q3) '(t)']; % %***** % Define temporary velocities with the time variable for i=1:1:ND eval(['tmp',num2str(1000+i),'=[char(dq',num2str(i),' (t)'';')]) end %Previous code %tmp1001=[char(dq1) '(t)']; % %tmp1002=[char(dq2) '(t)']; % %tmp1003=[char(dq3) '(t)']; % %***** % Substitute the temporary coordinates into the coordinates % and differentiate it regarding the time variable for i=1:1:ND eval(['q',num2str(i),'=subs(q',num2str(i),'q',num2str(i)),'tmp',num2str(i),');']) eval(['dq',num2str(i),'=diff(q',num2str(i),'t')']) end % Previous % q1=subs(q1,q1,tmp1); % dq1=diff(q1,t); % q2=subs(q2,q2,tmp2); % dq2=diff(q2,t); % q3=subs(q3,q3,tmp3); % dq3=diff(q3,t); %***** </pre>	<pre> % Lagrangian % Kinetic Energy (User define) T=(1/2)*(m)*(dq1^2)+(1/2)*(m)*(dq2^2)+(1/2)*(1/12*m*l^2) *(dq3^2); %***** % Change diff(qi(t),t) to dq_i in T for the next partial derivative for i=1:1:ND eval(['T=subs(T,'diff(q',num2str(i),'(t),t)','dq',num 2str(i),')']) end % Previous % T=subs(T,'diff(q1(t),t)','dq1'); % T=subs(T,'diff(q2(t),t)','dq2'); % T=subs(T,'diff(q3(t),t)','dq3'); %***** % Change qi(t) to qi in T for the next partial derivative for i=1:1:ND eval(['T=subs(T,'q',num2str(i),'(t)','q',num2str(i),' ')']) end % Previous % T=subs(T,'q1(t)','q1'); % T=subs(T,'q2(t)','q2'); % T=subs(T,'q3(t)','q3'); %***** % Partial derivate of T regarding each coordinate for i=1:1:ND eval(['dtuc(',num2str(i),')=diff(T,'q',num2str(i),')'); ']) end % Previous % dtuc(1)=diff(T,'q1'); % dtuc(2)=diff(T,'q2'); % dtuc(3)=diff(T,'q3'); for i=1:1:length(dtuc); for j=1:1:ND eval(['dtuc(',num2str(i),')=subs(dtuc(',num2str(i),'),' diff(dq',num2str(j),'(t),t)','ddq',num2str(j),'(t)')'); eval(['dtuc(',num2str(i),')=subs(dtuc(',num2str(i),'),' dq',num2str(j),','dq',num2str(j),'(t)')'); eval(['dtuc(',num2str(i),')=subs(dtuc(',num2str(i),'),' q',num2str(j),','q',num2str(j),'(t)')');]) % Previous % dtuc(i)=subs(dtuc(i),'diff(dq1(t),t)','ddq1(t)'); % dtuc(i)=subs(dtuc(i),'dq1','dq1(t)'); % dtuc(i)=subs(dtuc(i),'q1','q1(t)'); end end </pre>
---	--

Fig. 5. unconservation_MakingEquation : Eqs. (11-12), (16)



```

pkg load symbolic

% Symbolic expression about coordinates, velocities, and
% design variables (User define)
for i=1:1:ND
eval(['syms q',num2str(i),' dq',num2str(i)])
end
syms m l ... % mass, length
t g ... % time, gravity
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Tvar=[time, coordinates, velocities, the others]
% for defining all functions regarding 'Tvar'
automatically
% and for using ode45 solver
Tvar=[t];
for i=1:1:ND
eval(['Tvar=[Tvar q',num2str(i),'];'])
end
for i=1:1:ND
eval(['Tvar=[Tvar dq',num2str(i),'];'])
end
Tvar=[Tvar m l g];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Define temporary coordinates with the time variable
for i=1:1:ND
eval(['tmp',num2str(i),'=[char(q',num2str(i),'
'(t)');'])
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Define temporary velocities with the time variable
for i=1:1:ND
eval(['tmp',num2str(10000+i),'=[char(dq',num2str(i),'
'(t)');'])
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Substitute the temporary coordinates into the
coordinates
% and differentiate it regarding the time variable
for i=1:1:ND
eval(['q',num2str(i),'=subs(q',num2str(i),'q',num2str(i)
),'tmp',num2str(i),');'])
eval(['dq',num2str(i),'=diff(q',num2str(i),'t');'])
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Lagrangian
% Kinetic Energy (User define)
T=(1/2)*(m)*(dq1^2)+(1/2)*(m)*(dq2^2)+(1/2)*(1/12*m*l^2)
*(dq3^2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Change diff(qi(t),t) to dq_i in T for the next partial
derivative
for i=1:1:ND
eval(['T=subs(T,'diff(q',num2str(i),'t),t','dq',num
2str(i),');'])
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Define partial derivative of T regarding each velocity
for i=1:1:ND
eval(['tt(',num2str(i),'=diff(T,'dq',num2str(i),');'
)])
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Define temporary velocities with the time variable
again
for i=1:1:ND
eval(['tmp',num2str(10000+i),'=[char(dq',num2str(i),'
'(t)');'])
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Change qi and dq_i to qi(t) and dq_i(t) respectively in
tt(i)
for i=1:1:length(tt)
for j=1:1:ND;
eval(['tt(i)=subs(tt(i),'dq',num2str(j),'','aaa',num2
str(j),');'])
eval(['tt(i)=subs(tt(i),'q',num2str(j),'','q',num2str
(j),'(t)');'])
eval(['tt(i)=subs(tt(i),'dq',num2str(j),'','dq',num2s
tr(j),'(t)');'])
eval(['tt(i)=subs(tt(i),'aaa',num2str(j),'','dq',num2
str(j),'(t)');'])
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Define time derivate of tt
for i=1:1:length(tt);
dtt(i)=diff(tt(i),'t');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Potential energy term
V=sym(0);

% Change qi(t) to qi in V
for i=1:1:ND
eval(['V=subs(V,'q',num2str(i),'(t)','q',num2str(i),'
');'])
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Partial derivate of V regarding each coordinate
for i=1:1:ND
eval(['dv(i)=diff(V,'q',num2str(i),');'])
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Change qi to qi(t) in dv(i) for the next time
derivative
for i=1:1:length(dv)
for j=1:1:ND;
eval(['dv(i)=subs(dv(i),'q',num2str(j),'','q',num2str
(j),'(t)');'])
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Final Equation : Lagrangian regarding each coordinate
for i=1:1:length(tt);
Eq(i)=dtt(i)-dttuc(i)+dv(i);
end

% Change 'diff' and 'diff(diff)' to 'd' and 'dd'
respectively
for i=1:1:length(Eq)
for j=1:1:ND;
eval(['Eq(i)=subs(Eq(i),'diff(dq',num2str(j),'(t),t)'
,'ddq',num2str(j),'(t)');'])
eval(['Eq(i)=subs(Eq(i),'diff(q',num2str(j),'(t),t)'
,'dq',num2str(j),'(t)');'])
eval(['Eq(i)=subs(Eq(i),'q',num2str(j),'(t)','q',num2
str(j),'(t)');'])
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Expand Lagrangian
for i=1:1:length(Eq);
Eq(i)=expand(Eq(i));
end

```

Fig. 6. MakingEquation : Eqs. (11-13), (17-18)

3.2 Derivation of the equations of motion and the gravity vector

This section deals with deriving equations of motion with three degrees of freedom into GNU Octave based on Eqs. (12-13) derived from the system definition. Although the contents of the previous study [11] are similar to each other, in the linear study, to use the development code to handle other systems, all codes had to be customized. The difference is that the process is automated. Fig. 4 is a single pendulum dynamics analysis *main_script*, consisting of three parts: derivation of equations of motion, derivation of constrained equation, and numerical analysis. The difference between the existing development code and the



previous code is that the damping matrix and stiffness matrix are transposed to external force vector terms rather than organizing a matrix regarding velocity and displacement coordinates because it needs to be summarized in Eq. (11), suitable for the application of the ode45 solver.

In the first code *unconservation_MakingEquation* that composes the derivation of the equation of motion - Part 1, coordinates q_1, q_2, q_3 , coordinates dq_1, dq_2, dq_3 and system variables m, l, k, t , and g are defined as symbolic variables (Fig. 5). Here, the coordinates q_1, q_2 , and q_3 represent x_G, y_G , and θ , respectively. One thing to note is that, unlike other variables, coordinates and speeds change over time, so (t) add a time component and differentiate the displacement coordinates concerning time to redefine the speed. When the essential variables are defined, the total kinetic energy T in the system is calculated. In the case of previous among the comments in the code, it indicates the past method that the user manually entered according to the system's coordinates. The existing development code receives the number of coordinates and speeds based on the number of coordinates ND given in the problem. It automatically declares variables, partial derivatives, and variable substitution. That is, the parts that the user should define are design variables and kinetic energy.

```
function [MM]=mmass (Eq,ND,Tvar)
S=Eq;

for i=1:length(S);
for j=1:ND
eval(['S(i)=subs(S(i),'ddq',num2str(j),'(t)','ddq',num2str(j),'');'])
eval(['S(i)=subs(S(i),'dq',num2str(j),'(t)','dq',num2str(j),'');'])
eval(['S(i)=subs(S(i),'cos(q',num2str(j),'(t))','cos(disp',num2str(j),'');'])
eval(['S(i)=subs(S(i),'sin(q',num2str(j),'(t))','sin(disp',num2str(j),'');'])
eval(['S(i)=subs(S(i),'cos(q',num2str(j),'(t))','cos(disp',num2str(j),'');'])
eval(['S(i)=subs(S(i),'cos(q',num2str(j),'(t))^2','cos(disp',num2str(j),'^2');'])
eval(['S(i)=subs(S(i),'sin(q',num2str(j),'(t))^2','sin(disp',num2str(j),'^2');'])
eval(['S(i)=subs(S(i),'sin(q',num2str(j),'(t))','sin(disp',num2str(j),'^2');'])
eval(['S(i)=subs(S(i),'q',num2str(j),'(t)^2','disp',num2str(j),'^2');'])
eval(['S(i)=subs(S(i),'q',num2str(j),'(t)','q',num2str(j),'');'])
end
end
%Previous
%S(i)=subs(S(i),'ddq1(t)','ddq1');
%S(i)=subs(S(i),'ddq2(t)','ddq2');
%S(i)=subs(S(i),'ddq3(t)','ddq3');

%S(i)=subs(S(i),'dq1(t)','dq1');
%S(i)=subs(S(i),'dq2(t)','dq2');
%S(i)=subs(S(i),'dq3(t)','dq3');

%S(i)=subs(S(i),'cos(q1(t))','cos(disp1)');
%S(i)=subs(S(i),'cos(q2(t))','cos(disp2)');
%S(i)=subs(S(i),'cos(q3(t))','cos(disp3)');

%S(i)=subs(S(i),'sin(q1(t))','sin(disp1)');
%S(i)=subs(S(i),'sin(q2(t))','sin(disp2)');
%S(i)=subs(S(i),'sin(q3(t))','sin(disp3)');

%S(i)=subs(S(i),'cos(q1(t))^2','cos(disp1)^2');
%S(i)=subs(S(i),'cos(q2(t))^2','cos(disp2)^2');
%S(i)=subs(S(i),'cos(q3(t))^2','cos(disp3)^2');

%S(i)=subs(S(i),'sin(q1(t))^2','sin(disp1)^2');
%S(i)=subs(S(i),'sin(q2(t))^2','sin(disp2)^2');
%S(i)=subs(S(i),'sin(q3(t))^2','sin(disp3)^2');

%S(i)=subs(S(i),'sin(q1(t))','sin(disp1)');
%S(i)=subs(S(i),'sin(q2(t))','sin(disp2)');
%S(i)=subs(S(i),'sin(q3(t))','sin(disp3)');

%S(i)=subs(S(i),'q1(t)^2','disp1^2');
%S(i)=subs(S(i),'q2(t)^2','disp2^2');
%S(i)=subs(S(i),'q3(t)^2','disp3^2');

%S(i)=subs(S(i),'q1(t)','q1');
%S(i)=subs(S(i),'q2(t)','q2');
%S(i)=subs(S(i),'q3(t)','q3');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Constant=S;
for i=1:length(Constant);
for j=1:ND
eval(['Constant(i)=subs(Constant(i),'ddq',num2str(j),'','0');'])
end
end
%Previous
%Constant(i)=subs(Constant(i),'ddq1','0');
%Constant(i)=subs(Constant(i),'ddq2','0');
%Constant(i)=subs(Constant(i),'ddq3','0');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S=S-Constant;

for i=1:length(S);
S(i)=expand(S(i));
end

for i=1:length(S);
mS(i)=S(i);
end

for i=1:length(Constant);
CConstant(i)=Constant(i);
end

for j=1:length(mS);
mM=mS;
clear UUU
UUU=zeros(length(mS),1);
UUU(j,1)=1;
for jj=1:ND
eval(['aaa',num2str(jj),'=UUU(',num2str(jj),',1);'])
%Previous
%aaa1=UUU(1,1);
%aaa2=UUU(2,1);
%aaa3=UUU(3,1);
end

for i=1:length(mS);
for ii=1:ND
eval(['mM(i)=subs(mM(i),'ddq',num2str(ii),'','aaa',num2str(ii),'');'])
%Previous
%mM(i)=subs(mM(i),'ddq1',aaa1);
%mM(i)=subs(mM(i),'ddq2',aaa2);
%mM(i)=subs(mM(i),'ddq3',aaa3);
MM(i,j)=mM(i);
end
end
end

%Define the function regarding Tvar
function_handle(MM,'vars',Tvar,'file','Mass')

for i=1:ND
eval(['MM=subs(MM,'ddq',num2str(i),'','accl',num2str(i),'');'])
eval(['MM=subs(MM,'dq',num2str(i),'','vel',num2str(i),'');'])
eval(['MM=subs(MM,'q',num2str(i),'','disp',num2str(i),'');'])
end
```

Fig. 7. *m*mass : Eq. (19)



<pre> function [CConstant]=mstiff(Eq,ND,Tvar) S=Eq; for i=1:length(S); for j=1:1:ND eval(['S(i)=subs(S(i),'ddq',num2str(j),'(t)',''0'');'])) eval(['S(i)=subs(S(i),'dq',num2str(j),'(t)',''vel',num 2str(j),'');']) eval(['S(i)=subs(S(i),'q',num2str(j),'(t)',''disp',num 2str(j),'');']) eval(['S(i)=subs(S(i),'cos(q',num2str(j),'(t)',''cos(disp',num2str(j),'');']) eval(['S(i)=subs(S(i),'sin(q',num2str(j),'(t)',''sin(disp',num2str(j),'');']) end end %% for i=1:length(S); S(i)=expand(S(i)); end Constant=transpose(S); CConstant=Constant; %Define the function regarding Tvar function_handle(CConstant,'vars',Tvar,'file','Generalize d1') for i=1:length(CConstant) for j=1:1:ND eval(['CConstant(i)=subs(CConstant(i),'ddq',num2str(j), '','','accl',num2str(j),'');']) eval(['CConstant(i)=subs(CConstant(i),'dq',num2str(j), '','','vel',num2str(j),'');']) eval(['CConstant(i)=subs(CConstant(i),'q',num2str(j),' ','disp',num2str(j),'');']) end end </pre>	<pre> function [dgvv]=mgravity(dgv,ND,Tvar) S=dgv; for i=1:length(S); for j=1:1:ND; eval(['S(i)=subs(S(i),'q',num2str(j),'(t)',''disp',num 2str(j),'');']) end end %% for i=1:length(S) dgvv(i)=S(i); end dgvv=transpose(dgvv); %Define the function regarding Tvar function_handle(dgvv,'vars',Tvar,'file','Gravity') for i=1:1:ND eval(['dgvv=subs(dgvv,'ddq',num2str(i),'','','accl',num2 str(i),'');']) eval(['dgvv=subs(dgvv,'dq',num2str(i),'','','vel',num2st r(i),'');']) eval(['dgvv=subs(dgvv,'q',num2str(i),'','','disp',num2st r(i),'');']) end </pre>
---	--

Fig. 8. *mstiff* & *mgravity* : Eq. (19)

Through this code, we can calculate the box mark term in Eq. (16), and it is declared as a *dtuc* vector and saved as a temporary file through sequential calculation of the main script. For details of the code, please refer to the previous study [11].

$$\frac{d}{dt} \left(\frac{\partial T}{\partial \dot{q}} - \frac{\partial V}{\partial \dot{q}} \right) - \left(\frac{\partial T}{\partial q} - \frac{\partial V}{\partial q} \right) = Q + \Phi_q^T \lambda \quad (16)$$

The second code, *MakingEquation* code, is the process of deriving the equation of motion. The code from the definition of the symbolic variable to the total kinetic energy T is the same as *unconservation_MakingEquation*. The explanation is omitted as the partial differential process in Eqs. (17-18) is not different from the previous code. (Fig. 6) As the *unconservation_MakingEquation* defined above, if the user defines only the degrees of freedom, kinetic energy, and potential energy, it can be seen that the kinetic energy equation, including symbolic variables, is automatically derived. The *Gravityterms* code that is not shown in the paper and calculates the gravitational term from the potential energy caused by gravity should be derived by following a derivation process similar to that of *MakingEquation* code. The Jacobian matrix Φ_q and the Lagrange multiplier λ based on the constraint equation are dealt with separately in Section 3.4.

$$\frac{d}{dt} \frac{\partial T}{\partial \dot{q}} - \frac{d}{dt} \frac{\partial V}{\partial \dot{q}} - \left(\frac{\partial T}{\partial q} - \frac{\partial V}{\partial q} \right) = Q + \Phi_q^T \lambda \quad (17)$$

$$\frac{d}{dt} \frac{\partial T}{\partial \dot{q}} - \frac{d}{dt} \frac{\partial V}{\partial \dot{q}} - \left(\frac{\partial T}{\partial q} - \frac{\partial V}{\partial q} \right) = Q + \Phi_q^T \lambda \quad (18)$$

3.3 Matrix-vectorization from the equations of motion

To correctly apply the equation of motion and gravity in the form of symbolic expression derived in Section 3.2 to the *ode45* solver, it is necessary to convert the M mass matrix and the external force Q vector in Eq. (19). In *main_script*-Part 1, *m*mass derives the M matrix (Fig. 7) in the equation of motion, *m*stiff derives the Q vector (Fig. 8) in the equation of motion, and *m*gravity derives the code that functions the gravity applied in the Q vector. In detail, *m*mass code removes all terms other than the acceleration vector. It extracts the mass matrix by expressing the remaining terms as (mass matrix) \times (acceleration vector). *m*stiff and *m*gravity calculate the remaining external force terms excluding (mass matrix) \times (acceleration vector).

At this time, *Tvar*, including time, coordinates, speed, and design variables introduced in Section 3.2, is used as input variables in the three codes in earnest. *Tvar* is used to automatically organize the process necessary for automatic response calculation by accepting the degrees of freedom defined by the user in inputting information on the closed chain system. For example, the gravitational acceleration g is not used to function the M matrix but is used to function the gravitational term in the Q vector. That is because g can be selectively used as an input variable or not, a function of a different structure can appear. In this case, the user has to change the code from time to time, creating an effective variable processing flow chart. If *Tvar* is equally applied as an input variable and unnecessary variables in functionalization are treated with a dummy.



$$\begin{bmatrix} \mathbf{M} & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{Q} \\ -(\Phi_q \dot{\mathbf{q}})_q \dot{\mathbf{q}} - 2\Phi_{qt} \dot{\mathbf{q}} - \Phi_{tt} \end{bmatrix} \tag{19}$$

3.4 Derivation of constraint equations and matrix-vectorization

In this section, in *main_script*-Part 2, we derive the introduced acceleration analysis γ vector from the constraint equation to find the optimal solution that satisfies both the equation of motion and the constraint equation using the Jacobian Φ_q matrix and the Lagrangian multiplier method (Eq. 20). Defining the constraints and partial differentiation are processed in the *ConstraintEquation* code same as the previously derivation equations of motion and matrix-vectorization. *mconst* code preforms functionalization (Fig. 9-10). As two constraint equations and three coordinates are derived in the single pendulum, Φ_q becomes a two by three matrix, and γ becomes a three by one vector.

$$\begin{bmatrix} \mathbf{M} & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{Q} \\ -(\Phi_q \dot{\mathbf{q}})_q \dot{\mathbf{q}} - 2\Phi_{qt} \dot{\mathbf{q}} - \Phi_{tt} \end{bmatrix} \tag{20}$$

<pre> pkg load symbolic % Symbolic expression about coordinates, velocities, and % design variables (User define) for i=1:1:ND eval(['syms q',num2str(i),' dq',num2str(i)]) end syms m l ... % mass, length t g ... % time, gravity %% % Tvar=[time, coordinates, velocities, the others] % for defining all functions regarding 'Tvar' automatically % and for using ode45 solver Tvar=[t]; for i=1:1:ND eval(['Tvar=[Tvar q',num2str(i),''];']) end for i=1:1:ND eval(['Tvar=[Tvar dq',num2str(i),''];']) end Tvar=[Tvar m l g]; %% % Define temporary coordinates with the time variable for i=1:1:ND eval(['tmp',num2str(i),'=[char(q',num2str(i),' ' '(t)'';']) end %% % Define temporary velocities with the time variable for i=1:1:ND eval(['tmp',num2str(10000+i),'=[char(dq',num2str(i),' ' '(t)'';']) end %% % Constraint Equation (User define with NC) Q(1,1)=q1-(1/2)*sin(q3); Q(2,1)=q2+(1/2)*cos(q3); %% % Partial derivative of Q : Qq_dq_q_dq(1st term) % 1st stage for i=1:1:NC for j=1:1:ND eval(['Qq(i,j)=diff(Q(i),'q',num2str(j),'');']) end end dq=[dq1;dq2;dq3]; </pre>	<pre> % 2nd stage Qq_dq=Qq*dq; % 3rd stage for i=1:1:NC for j=1:1:ND eval(['Qq_dq_q(i,j)=diff(Qq_dq(i),'q',num2str(j),'');']) end end % 4th stage Qq_dq_q_dq=Qq_dq_q*dq; %% % Partial derivative of Q : 2Qq_t_dq(2nd term) % 1st stage for i=1:1:NC for j=1:1:ND Qq_t(i,j)=diff(Qq(i,j),'t'); end end % 2nd stage Qq_t_dq=Qq_t*dq; %% % Partial derivative of Q : Qtt(3rd term) % 1st stage for i=1:1:NC Qt(i,1)=diff(Q(i,1),'t'); end % 2nd stage for i=1:1:NC Qtt(i,1)=diff(Qt(i,1),'t'); end %% % Define Gamma Gamma=Qq_dq_q_dq+Qq_t_dq+Qtt; </pre>
--	--

Fig. 9. ConstraintEquation : Eqs.(14-15), (20)

```

function [S,H]=mconst(Qq,Gamma,ND,Tvar)
S=Gamma;
H=Qq;

% Define the function regarding Tvar
function_handle(S,'vars',Tvar,'file','Gamma')
function_handle(H,'vars',Tvar,'file','Qq')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                
```

Fig. 10. mconst : Eq.(20)



```

clc
close all;
clear all;
pkg load symbolic
% load the saved mat file to extract Tvar
load('temp_dtuc.mat')

% Set the parameters
Tspan = 0.0:0.001:3; % Time interval
m = 1.0; % Mass
l = 10; % Length
g = 9.81; % Gravity acceleration
thetal = 45/180*pi; % Initial angle

% Substitute values in Tvar to input it into 'pendulum'
function
Tvar=subs(Tvar,'m',m);
Tvar=subs(Tvar,'l',l);
Tvar=subs(Tvar,'g',g);
Tvar_num=Tvar(2*ND+1+1:end);
Tvar_num=double(Tvar_num);
Tvar_num_cell=num2cell(Tvar_num);

% Determine the initial conditions
q = zeros(3,1);
q(1) = 1/2*sin(thetal);
q(2) = -1/2*cos(thetal);
q(3) = thetal;
qdot = zeros(3,1); % All initial velocities are zero

Z0 = [q' qdot']; % Initial Condition Vector
options = odeset('RelTol',1.0e-9,'AbsTol',1.0e-6);
[Tout,Z] = ode45('pendulum',Tspan,Z0,Tvar_num_cell{:},
ND,NC,options);

x1 = Z(:,1);
y1 = Z(:,2);
Thetal= Z(:,3);
figure(1);
plot(x1,y1);
grid on;
xlabel('x-position(m)','fontsize',20);
ylabel('y-position(m)','fontsize',20);
hold on
plot(x1(1),y1(1),'k*')
plot(x1(end),y1(end),'r*')
h=legend('Path','Initial coordinate','Final
coordinate',
"location","north");
set(h,"fontsize",18)
xlim([-5 5])
ylim([-5 5])
axis square

figure(2);
plot(Tout, Thetal, 'k');
grid;
title('Angle of Crank');
xlabel('Time - seconds');
ylabel('Angle - radians');
xldot = Z(:,1+9);
yldot = Z(:,2+9);
Thetaldot = Z(:,3+9);
figure(3);
plot(Tout, xldot, 'k', Tout, yldot, 'k--', Tout,
Thetaldot, 'k-');
grid;
title('Velocities of Crank');
xlabel('Time - seconds');
ylabel('Velocities - meters/s and angular velocity -
rad/sec. ');
legend('Horizontal Velocity','Vertical
Velocity','Angular Velocity');

```

Fig. 11. Numerical_analysis_DAE

```

function zdot = pendulum(t,z,m,l,g,ND,NC)

% Separate coordinates and velocities
q = z(1:ND); % coordinate data
qdot = z(ND+1:end); % velocity data

% Arrange 'Tvar'
Tvar=[t transpose(z) m l g];
Tvar_cell=num2cell(Tvar);

% Define 'Mass' matrix
M=Mass(Tvar_cell{:});

% Define 'Gravity' and 'Generalized force' vector
Gravity1=-Gravity(Tvar_cell{:});
Generalized=-Generalized1(Tvar_cell{:});
Q=Gravity1+Generalized;

% Define 'Qq' matrix
Qq1=Qq(Tvar_cell{:});

% Define 'Gamma' vector
Gamma1=-Gamma(Tvar_cell{:});
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Build the left side of matrix
C=[M Qq1; Qq1 zeros(NC,NC)];
D=inv(C);

% Build the right side of vector
R=[Q;Gamma1];

% Calculate the solution
ACC=C\R;
qdd=ACC(1:ND);

% Determine the time derivative of the state vector
zdot=[qdot' qdd]';

```

Fig. 12. Pendulum

3.5 Calculation of the dynamic response

By executing the code introduced in Sections 3.1 to 3.4, all matrices and vectors shown in Eq. (11) can be called from the pre-defined functions. This information is called from the *pendulum* function to calculate acceleration and Lagrange multipliers based on the current response and velocity. Then, the Runge-Kutta method of the ode45 solver uses the information to calculate the response such as displacement and velocity (Fig. 11-12).

Table 1 shows the initial response and design variables. When the result of ode45 is derived, the single pendulum performs parabolic motion by gravity. (Fig. 13).

Table 1. Design variable and simulation condition

Design variable	Value	Simulation condition	Value
m	1 kg	t	0 ~ 3 sec
l	10 m	Δt	0.001 sec
g	9.81 m/s ²	Relative Tolerance	1×10 ⁻⁹
θ_{ni}	45°	Absolute Tolerance	1×10 ⁻⁶



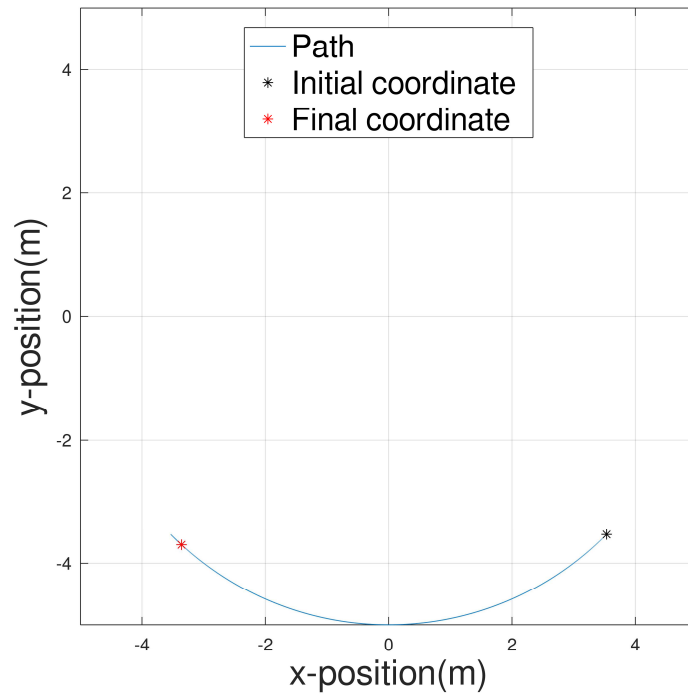


Fig. 13. x-y coordinates of the single pendulum

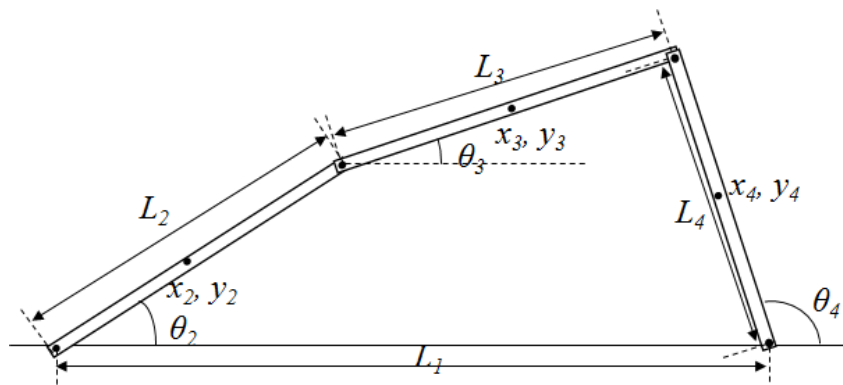


Fig. 14. 4-bar linkage

4. Code Application

4.1 System definition

In this section, we reviewed whether a user can use the code easily by using the derivation of the equations of motion and the response calculation code when an arbitrary system is given. Fig. 14 is the 4-bar linkage, and the initial posture, speed, and external force were applied by reference in previous studies using Simulink [17].

Before manipulating the code, a user must define the number of coordinates, design variables, kinetic energy, potential energy, external force, and constraint equations. Regarding the system coordinates, as there are three links with three x, y, θ coordinates, it consists of a total of 9 coordinates ($x_2, y_2, \theta_2, x_3, y_3, \theta_3, x_4, y_4, \theta_4$). The kinetic energy of each link is as shown in Eq. (21), and there is no potential energy by the spring that provides conserved energy in the system. Eq. (22) is the gravitational force due to the mass of each link, and it is summarized as a gravity vector through $mgravity$. Finally, a total of eight constraint equations were derived through the location analysis based on the geometric relationship review between each coordinate system (Eq. (23)).

$$T = \frac{1}{2} m_2 \dot{x}_2^2 + \frac{1}{2} m_2 \dot{y}_2^2 + \frac{1}{2} \left(\frac{1}{12} m_2 L_2^2 \right) \dot{\theta}_2^2 + \frac{1}{2} m_3 \dot{x}_3^2 + \frac{1}{2} m_3 \dot{y}_3^2 + \frac{1}{2} \left(\frac{1}{12} m_3 L_3^2 \right) \dot{\theta}_3^2 + \frac{1}{2} m_4 \dot{x}_4^2 + \frac{1}{2} m_4 \dot{y}_4^2 + \frac{1}{2} \left(\frac{1}{12} m_4 L_4^2 \right) \dot{\theta}_4^2 \tag{21}$$

$$V = m_2 g y_2 + m_3 g y_3 + m_4 g y_4 \tag{22}$$



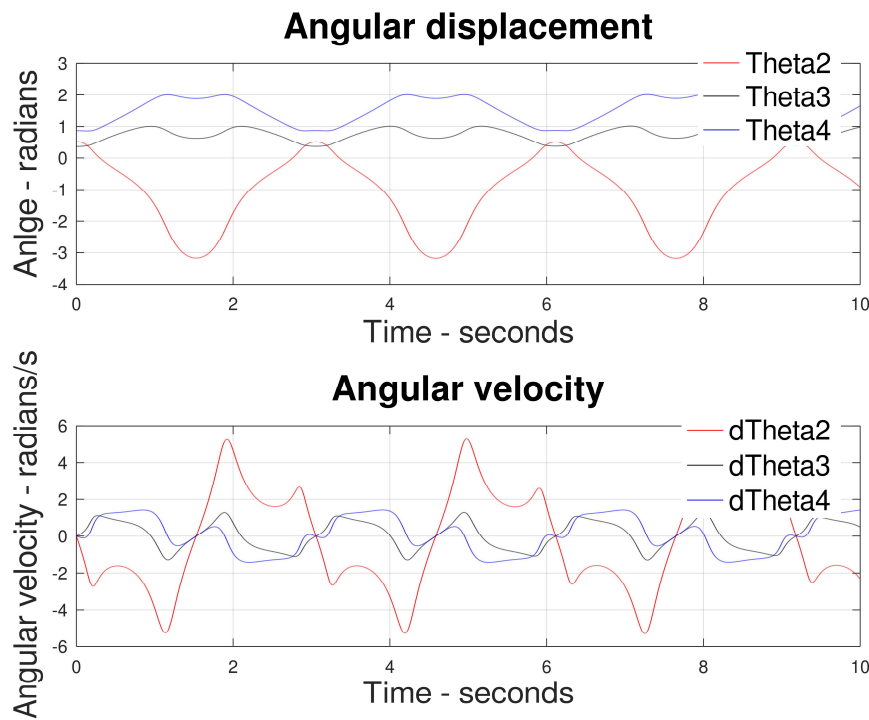


Fig. 15. Angular displacements (Theta) and velocities (dTheta) of the 4-bar linkage

$$\begin{aligned}
 x_2 &= \frac{L_2}{2} \cos \theta_2 \dots \langle 1 \rangle \\
 y_2 &= \frac{L_2}{2} \sin \theta_2 \dots \langle 2 \rangle \\
 x_3 &= L_2 \cos \theta_2 + \frac{L_3}{2} \cos \theta_3 \dots \langle 3 \rangle \\
 y_3 &= L_2 \sin \theta_2 + \frac{L_3}{2} \sin \theta_3 \dots \langle 4 \rangle \\
 x_4 &= \frac{L_1 + (L_2 \cos \theta_2 + L_3 \cos \theta_3)}{2} \dots \langle 5 \rangle \\
 y_4 &= \frac{0 + (L_2 \sin \theta_2 + L_3 \sin \theta_3)}{2} \dots \langle 6 \rangle \\
 L_2 \cos \theta_2 + L_3 \cos \theta_3 &= L_1 + L_4 \cos \theta_4 \dots \langle 7 \rangle \\
 L_2 \sin \theta_2 + L_3 \sin \theta_3 &= 0 + L_4 \sin \theta_4 \dots \langle 8 \rangle
 \end{aligned} \tag{23}$$

Before using the code, it is necessary to follow the guidelines below as equations are automatically processed according to the number of coordinates.

-To apply the standardized coordinates, the coordinates $(x_2, y_2, \theta_2, x_3, y_3, \theta_3, x_4, y_4, \theta_4)$ are used in Eqs. (21-23), are converted to $(q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9)$

-Update the number of coordinates and constraint equations in *main_script*

-Kinetic energy, potential energy, external force, and constraint equation update

4.2 Response calculation

The results obtained from deriving the equations of motion and the constraint equations, and matrix transformation code, and the results of substituting the design variables, initial condition Case 1 from suggested in the previous study [17] into the numerical analysis code show responses in Fig. 15. The response result is the extraction of the time history for the same physical quantity as the time history of angular displacement and velocity in the case of 1 in the previous study. As the two results are the same, the derivation of the system equation and response calculation is correctly performed. The previous study used MATLAB/Simulink which is a commercial software, to obtain the system response, so this code has the same quality to solve a numerical problem compared to MATLAB/Simulink. However, the previous study derived the equations of motion and the differential-algebraic equations by hand and modeled the block diagram regarding the equations of motion and the differential-algebraic equation manually, this code has the strength to understand the whole process of dynamics analysis for a closed chain system.

5. Conclusion

This paper developed the dynamic analysis code for a closed chain system that improved the previous derivation of an equation of motion and response calculation code for an open-chain system. The GNU Octave code was developed to derive equations of motion using Lagrange mechanics and apply the ode45 numerical integration method so that users unfamiliar with



multibody dynamics can easily calculate system dynamics. The target system was selected to validate the code, and the system equations and responses were calculated correctly. Moreover, the user's understanding was induced by introducing the basics, procedures, and application-related explanations necessary for using the code. Through a series of processes, a user can easily derive a response by mechanically approaching the system definition, such as energy calculation and variable selection, to the matrix transformation problem that occurs in deriving the equation of motion of a more complex mechanical system. In the future, this code will be applied in an undergraduate class or major club to check the goods and find out the weakness. In addition, it is planned to expand to a GUI-based program to develop a free code that can quickly prepare dynamics analysis in the 3D modeling stage to increase its usability in the industry and to use it as educational materials for university dynamics and vibrations.

Acknowledgments

This work was supported by Yuhan University.

Conflict of Interest

The author declared no potential conflicts of interest with respect to the research, authorship and publication of this article.

Funding

The author received no financial support for the research, authorship and publication of this article.


Data Availability Statements

The datasets generated and/or analyzed during the current study are available from the corresponding author on reasonable request.

References

- [1] Yoo, W. S., Park, S. J., Dmitrochenko, O., Pogorelov, D., Verification of Absolute Nodal Coordinate Formulation in Flexible Multibody Dynamics via Physical Experiments of Large Deformation Problems, *Journal of Computational and Nonlinear Dynamics*, 1(1), 2006, 81-93.
- [2] Yoo, W. S., Kim, T. Y., Jung, S., History and Future of Multi-body Dynamics, *Transactions of the Korean Society of Mechanical Engineers - A*, 43(7), 2019, 483-491.
- [3] Munro, N., *Symbolic methods in control system analysis and design*, Institution of Engineering and Technology, Stevenage, 1999.
- [4] Van Khang, N., Kronecker product and a new matrix form of Lagrangian equations with multipliers for constrained multi-body systems, *Mechanics Research Communications*, 38(4), 2011, 294-299.
- [5] Negrut, D., Dyer, A., *ADAMS/Solver Primer*, MSC Software Documentation, Ann Arbor, 2004.
- [6] Bonev, I. A., *Geometric analysis of parallel mechanisms*, Université Laval, Quebec City, 2002.
- [7] vander Linde, R. Q., Schwab, A. L., *Lecture Notes Multi-body Dynamics B*, Delft University, Delft, 1998.
- [8] Sadati, S. M. H., Naghibi, S. E., Naraghi, M., An automatic algorithm to derive linear vector form of Lagrangian equation of motion with collision and constraint, *Procedia Computer Science*, 76, 2015, 217-222.
- [9] Sadati, S. M. H., Naghibi, S. E., Shiva, A., Zschaler, S., Hauser, H., Walker, I., Althoefer, K., Nanayakkara, T., AutoTMDyn: A Matlab software package to drive TMT Lagrange dynamics of series rigid-and continuum-link mechanisms, *IROS 2018 Workshop on Soft Robotic Modeling and Control: Bringing Together Articulated Soft Robots and Soft-Bodied Robots*, Madrid, Spain, 2018.
- [10] de Jalón, J. G., Callejo, A., A straight methodology to include multibody dynamics in graduate and undergraduate subjects, *Mechanism and Machine Theory*, 46(2), 2011, 168-182.
- [11] Park, Y. H., An Automatic Program of Generation of Equation of Motion and Dynamic Analysis for Multi-body Mechanical System Using GNU Octave, *Journal of Applied and Computational Mechanics*, 7(3), 2021, 1687-1697.
- [12] Yoo, H. H., Kane's methodology for derivation of equation of motion, *Journal of KSNVE*, 14(4), 2004, 40-47.
- [13] Yang, K. D., Bae, D. S., Yang, S. M., Choi, C. K., An Index 2 Differential-Algebraic Equation Formulation for Multibody System Dynamics, *Transactions of the Korean Society of Mechanical Engineers*, 19(11), 1995, 2769-2775.
- [14] Lee, D. C., Lee, S. H., Han, C. S., A Formulation of the Differential Equation on the Equations of motion and Dynamic Analysis for the Constrained Multibody Systems, *Transaction of the Korean Society of Automotive Engineers*, 5(1), 1997, 154-161.
- [15] Eaton, J. W., Bateman, D., Hauberg, S., *Gnu octave*, London: Network theory, London, 1997.
- [16] Ateş, A., Yeroğlu, C. Two degrees of freedom FOPID control loop design via SMDO algorithm, *Pamukkale University Journal of Engineering Sciences*, 22(8), 2016, 671-676.
- [17] Arda, M., Dynamic analysis of a four-bar linkage mechanism, *Machines. Technologies. Materials*, 14(5), 2020, 186-190.

ORCID iD

Yonghui Park  <https://orcid.org/0000-0001-6716-6935>



© 2022 Shahid Chamran University of Ahvaz, Ahvaz, Iran. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC 4.0 license) (<http://creativecommons.org/licenses/by-nc/4.0/>).

How to cite this article: Park Y.H. Development of an Educational Code of Deriving Equations of Motion and Analyzing Dynamic Characteristics of Multibody Closed Chain Systems using GNU Octave for a Beginner, *J. Appl. Comput. Mech.*, 8(1), 2022, 232-244. <https://doi.org/10.22055/JACM.2021.38021.3132>

Publisher's Note Shahid Chamran University of Ahvaz remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

