

Michael R. Buche
MAE 5730
Intermediate Dynamics and Vibrations
Final Computation Project
Due 12/6/17

MAE 5730

Intermediate Dynamics and Vibrations

Final Computation Project

Michael R. Buche

Fall 2017

48 Final Computation Project

N-Link Pendulum

The methods for the double pendulum are readily extended to a pendulum with N links. The relevant parameters for this system are

- The mass m_i and moment of inertia the about center of mass I^{G_i} for each link.
- The length ℓ_i and distance from upper hinge to center of mass d_i for each link.
- The strength of gravity g .
- The initial angles and angular velocities of the links.

This basic system conserves energy as all loads are conservative or are constraints that do no work. Therefore energy conservation may be utilized. The three methods used here to obtain the equations of motion for the N -link pendulum are

- (1) Minimal coordinates using angular momentum balance.
- (2) Minimal coordinates using Lagrange equations.
- (3) Maximal coordinates using angular momentum balance, linear momentum balance, and kinematic constraints.

This system is highly initial condition dependent. Some initial conditions cause a nice uniform motion that is easily integrated by the numerical solver of choice, `ODE45`, while other initial conditions cause chaotic motion that requires much more work out of `ODE45`. The angles θ_i for each link are measured relative to the vertical (gravity), where $\theta_i = 0$ is the equilibrium position. Two main initial condition types are used here, which are

- (1) Each link starting at the same initial angle $\frac{\pi}{4}$, while having zero initial angular velocities.
- (2) The initial angle for each link alternates between $\frac{\pi}{2}$ and ϕ , zig-zagging the links upwards and to the right, while having zero initial angular velocities.

For any amount of uniform links, the first set produces relative uniform oscillation in the solutions, while the second set produces chaotic motions. Additional features may be added to the problem, such as torques applied at the hinges, torsional springs at the hinges, or joint friction in the hinges. All three of these extra features are implemented here for method (1), while none are implemented for method (2), although they could be, and only torsional springs are implemented for method (3). Friction in the hinges cannot be implemented for method (3) because they dissipate energy and therefore energy is not conserved and Lagrange equations may not be used.

There are a quite a few `MATLAB` scripts that have been written to effectively handle this problem:

- `p48_ROOT.m` is the main file to be executed by the user. It allows them to select the method of obtaining the equations of motion, the number of links, the solution timespan, the initial conditions, the parameters for each link, and any extra features (torques, springs, friction). Saved workspaces for a variety of different cases are listed here, and can be loaded in order to skip the solving again. The tolerances for `ODE45`, plotting font size, and animation and gif settings are also set here.
- `p48_simple_system.m` is a function executed in `p48_ROOT.m` that sets simple parameters for the system of N links automatically. As it stands, it creates uniform links of unit length and mass with $I^G = \frac{1}{12} m\ell^2$ and $d = \frac{1}{2} \ell$, $g = 9.81$, and either the nice initial conditions (1) or the chaotic ones (2). All results shown here use this function. It also uniformly sets up torques, springs, and friction at the hinges if prompted to do so.

- `p48.RHS.m` is the right-hand-side function used by ODE45. It takes in the equations of motion in the form $\mathbf{M} \cdot \dot{\mathbf{z}} = \mathbf{b}$, evaluates the symbolic variables at \mathbf{z} , and inverts \mathbf{M} in order to output $\dot{\mathbf{z}} = \mathbf{M}^{-1} \cdot \mathbf{b}$ to ODE45.
- `p48.solve.m` is a function executed in `p48.ROOT.m` that symbolically obtains, simplifies, and solves (writes $\dot{\mathbf{z}} = \mathbf{A} \cdot \mathbf{z}$) the equations of motion for N links based on the user selected method and parameters, integrates the equations of motion in ODE45 based on the user selected timespan and tolerances, and finally stores and returns all of the relevant variables. Text is displayed in the command window to summarize the current system being solved. A progress bar, created by [1] and available at MATLAB Central, appears during the ODE45 integration and allows the user to monitor progress and terminate it early, keeping the current results.
- `p48.plot.m` is a function executed in `p48.ROOT.m` that plots the angles of each link wrapped to $[-\pi, \pi]$ and unwrapped (if any angles turn over), the trajectory of the end of each link, and the trajectory of the end of the last link.
- `p48.energy.m` is a function executed in `p48.ROOT.m` that plots the change in kinetic, potential, and total energy (compared to their initial values) for the system in time. Currently, the energy calculations take into account the extra torsional springs, but do not take into account friction or torques.
- `p48.animate.m` is a function executed in `p48.ROOT.m` that animates the N -link pendulum in real time or sped up according a timescale set by the user. Hinges and a timer can be shown or hidden by the user.
- `p48.gif.m` is a function executed in `p48.ROOT.m` that saves a gif of the animation with filename set by the user. The gif is created using a function `gif.m`, created by [2] and available and available at MATLAB Central.

Minimal Coordinates Using AMB

The minimal coordinates for this system are the angle of each link θ_i . The x direction is aligned with gravity, and the y direction is “to the right.” The equations of motion are obtained by writing the angular momentum balance about each hinge:

$$\mathbf{T}_j - k_j(\theta_j - \theta_{j-1}) - c_j(\dot{\theta}_j - \dot{\theta}_{j-1}) + \sum_{i=1}^N \left(\mathbf{r}_{G_i/A_j} \times m_i g \hat{\mathbf{i}} \right) = \sum_{i=1}^N \left(\mathbf{r}_{G_i/A_j} \times m_i \mathbf{a}_{G_i} + I^{G_i} \ddot{\theta}_i \hat{\mathbf{k}} \right) \quad (1)$$

where $\mathbf{T}_j = T_j \hat{\mathbf{k}}$ is the applied torque at the j^{th} hinge, k_j is the j^{th} spring constant of the torsional spring, c_j is the j^{th} friction constant, θ_j is the angle of the j^{th} link, θ_{j-1} is the angle of the preceding link (on the other side of the j^{th} hinge), \mathbf{r}_{G_i/A_j} is the position of the i^{th} link’s center of mass with respect to the j^{th} hinge, m_i is the mass of the i^{th} link, I^{G_i} is the moment of inertia about the center of mass of the i^{th} link, and \mathbf{a}_{G_i} is the acceleration of the center of mass of the i^{th} link. For the first link ($j = 1$), take $\theta_{j-1} = \dot{\theta}_{j-1} = 0$. This is all accomplished in `p48.solve.m`.

Minimal Coordinates Using Lagrange Equations

The potential energy of the system is

$$E_p = \frac{1}{2} \sum_{i=1}^N \left[k_i(\theta_i - \theta_{i-1})^2 - m_i \mathbf{r}_{G_i/O} \cdot \hat{\mathbf{i}} \right] \quad (2)$$

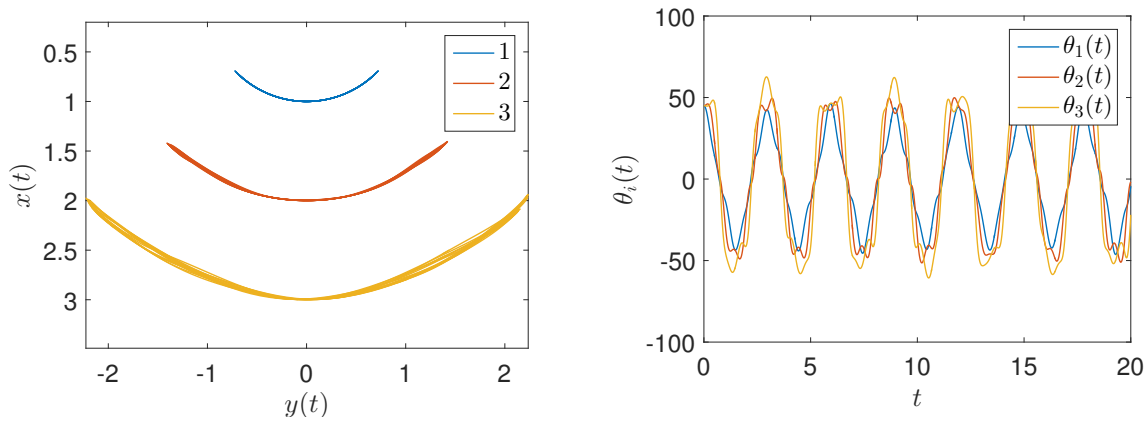


Figure 1: Minimal coordinates AMB method, 3 links, nice initial conditions. Plot of the trajectories of the ends of each link (left) and their angles (right).

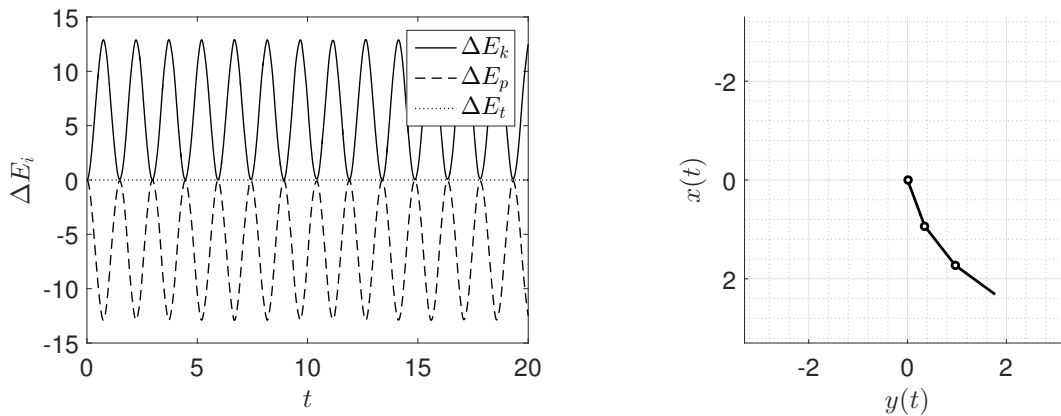


Figure 2: Minimal coordinates AMB method, 3 links, nice initial conditions. Plot of the changes in energy (left) and a screenshot of the animation (right). There is effectively no change in the total energy, as this simple motion causes small enough numerical error over time.

where $\theta_0 = 0$ and $\mathbf{r}_{G_i/O}$ is the position of the center of mass of the i^{th} link with respect to the origin. The kinetic energy of the system is

$$E_k = \frac{1}{2} \sum_{i=1}^N \left(m_i \mathbf{v}_{G_i} \cdot \mathbf{v}_{G_i} + I^{G_i} \dot{\theta}_i^2 \right) \quad (3)$$

where \mathbf{v}_{G_i} is the velocity of the center of mass of the i^{th} link. The Lagrangian $L = E_k - E_p$ can then be written and used in the Lagrange equations

$$\frac{\partial L}{\partial \theta_i} - \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\theta}_i} \right) = 0 \quad (4)$$

See that torsional springs are accounted for but applied torques are not (would be generalized forces). Friction cannot be used in this method. The equations of motion obtained here should be identical to those of the angular momentum balance method because they both use minimal coordinates. This is only true though if one takes the steps to simplify them to identical equations before integrating them. In simple cases (nice initial conditions) the motions will be identical, but in more complicated cases the motions will diverge. For example, even though $\dot{x} = f(x)$ and $\dot{x} \sin x = f(x) \sin x$ contain the same information and are the same equation, integrating them without simplifying the second equation may allow finite numerical

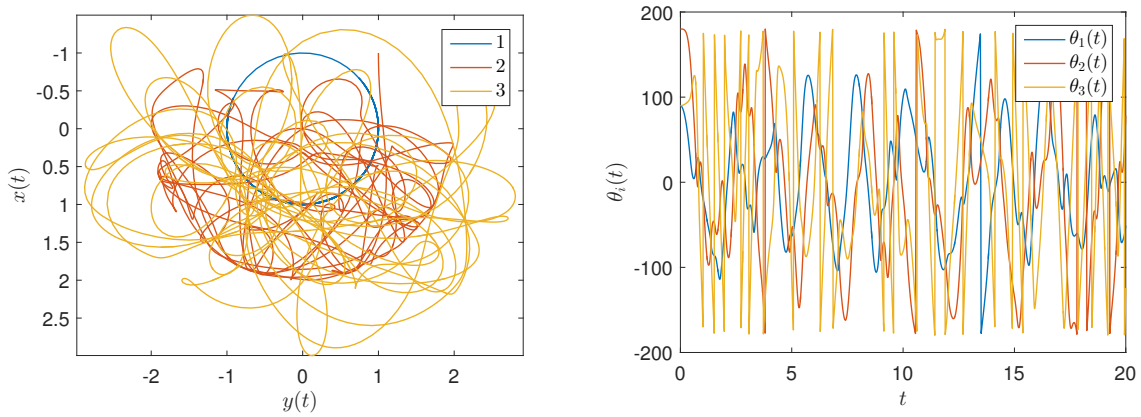


Figure 3: Minimal coordinates AMB method, 3 links, chaotic initial conditions. Plot of the trajectories of the ends of each link (left) and their angles (right).

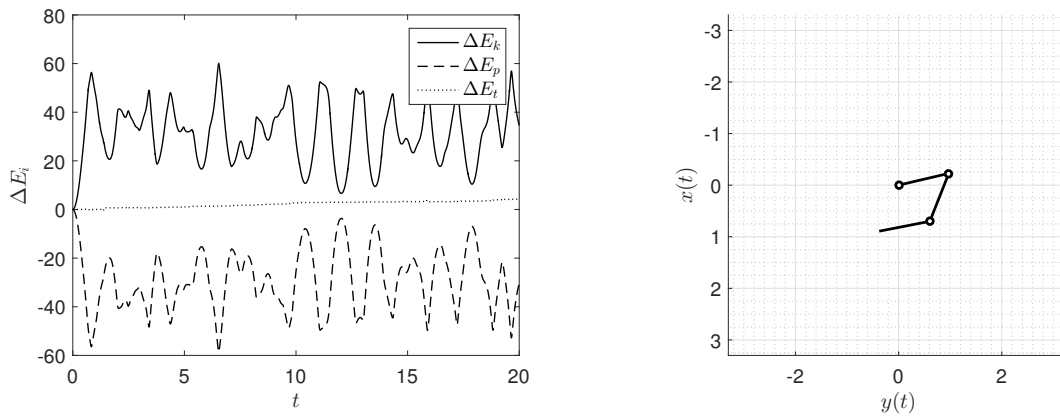


Figure 4: Minimal coordinates AMB method, 3 links, chaotic initial conditions. Plot of the changes in energy (left) and a screenshot of the animation (right). There is a small increasing change in the total energy, as this chaotic motion causes some numerical error.

precision to cause the solutions to become different over time. This simplification step is not taken in the scripts presented in order to illustrate this phenomenon.

Maximal Coordinates

The set of maximal coordinates are θ_i , x_{G_i} , and y_{G_i} . Additional unknowns are the reaction forces at each hinge \mathbf{R}_i . There are three relevant equations to describe the system:

- Angular momentum balance equations about each link's center of mass.
- Linear momentum balance equations on each link's center of mass.
- Kinematic constraint equations for $\theta_i = \theta_i(x_{G_i}, y_{G_i})$.

Note that kinematic constraint equations are necessarily equations relating two sets of redundant but individually minimal coordinates. θ_i is one set of minimal coordinates, but x_{G_i}, y_{G_i} could also be used as minimal coordinates. By using both of them and therefore more than needed (more coordinates than DOF), they cannot be independent and therefore must be related, and that is by a set of kinematic constraint

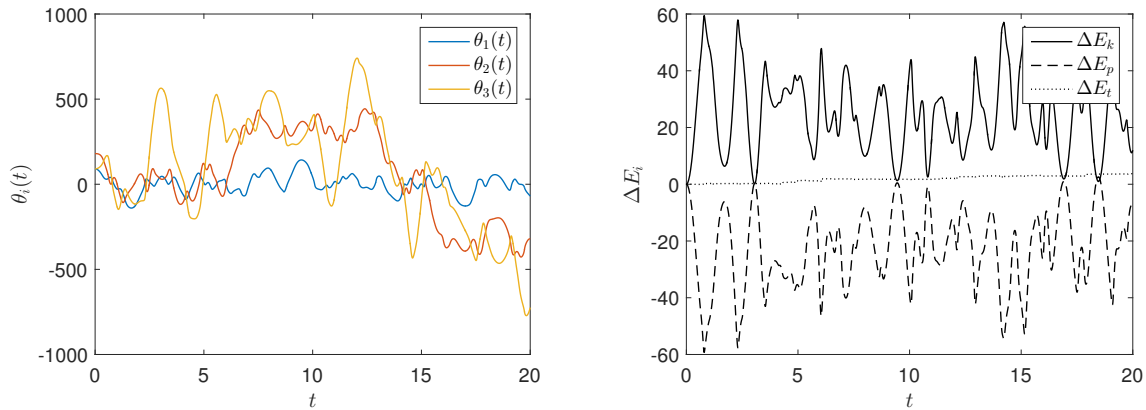


Figure 5: Minimal coordinates AMB method, 3 links, chaotic initial conditions. Torsional springs included. Plot of the angles of each link (left) and the changes in energy (right). The change in total energy only drifts a tad for this bouncy case. The angles here are plot unwrapped because repeated turning over puts more and more torques on the hinges from the torsional springs.

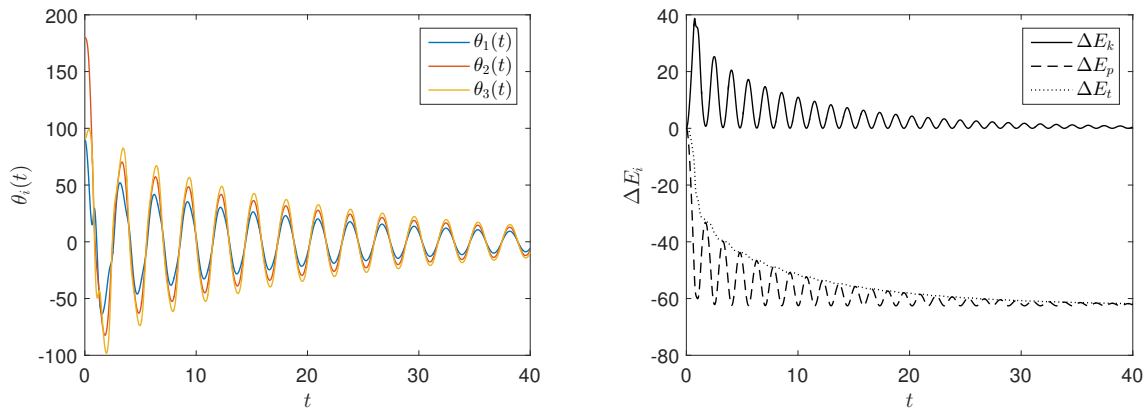


Figure 6: Minimal coordinates AMB method, 3 links, chaotic initial conditions. Torsional springs and hinge friction included. Plot of the angles of each link (left) and the changes in energy (right). The friction in the hinges prevents the initial conditions and springs from causing chaotic motion, and dampens the system more and more over time.

equations. The best set to use is writing the position of the center of mass of each link with respect to the origin in two ways (the Laurie Anderson equation, “let x equal x ”):

$$\mathbf{r}_{G_i/O} = x_{G_i} \hat{\mathbf{i}} + y_{G_i} \hat{\mathbf{j}} = x_{A_i} \hat{\mathbf{i}} + y_{A_i} \hat{\mathbf{j}} + d_i (\cos \theta_i \hat{\mathbf{i}} + \sin \theta_i \hat{\mathbf{j}}) \quad (5)$$

where one uses the recursion

$$x_{A_i} = x_{A_{i-1}} + \frac{\ell_i}{d_i} (x_{G_i} - x_{A_{i-1}}), \quad y_{A_i} = y_{A_{i-1}} + \frac{\ell_i}{d_i} (y_{G_i} - y_{A_{i-1}}) \quad (6)$$

with the initial points (note: A_1 is O)

$$x_{A_1} = \frac{\ell_1}{d_1} x_{G_1}, \quad y_{A_1} = \frac{\ell_1}{d_1} y_{G_1} \quad (7)$$

The kinematic constraint equations must be differentiated twice by t in order to write them with the AMB and LMB equations in the form $\dot{\mathbf{z}} = \mathbf{A} \cdot \mathbf{z}$, where \mathbf{A} is the mass-matrix inverted. This differentiation is what

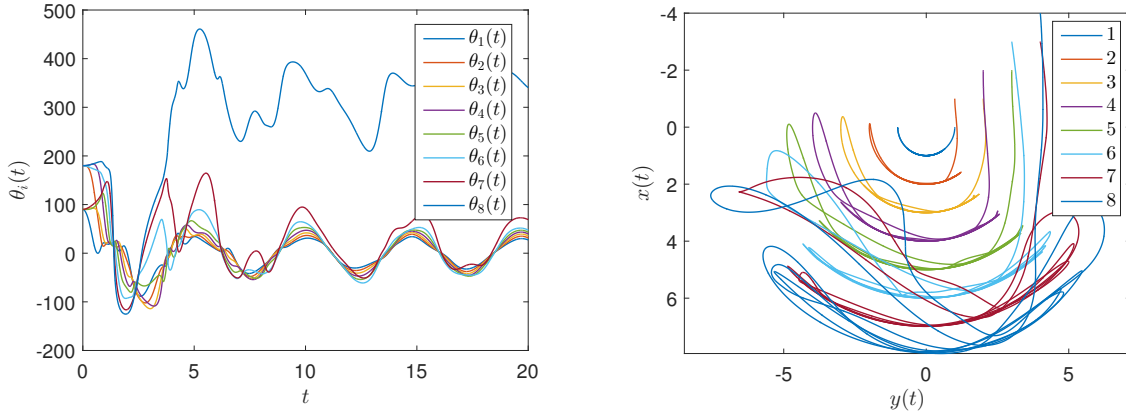


Figure 7: Minimal coordinates AMB method, 8 links, chaotic initial conditions. Torsional springs, hinge friction, and applied torques included. Plot of the angles of each link (left) and the trajectories of the end of each link (right). This case exemplifies one of the most complicated cases this set of scripts can handle: many links, chaotic initial conditions, applied torques, torsional springs, and friction.

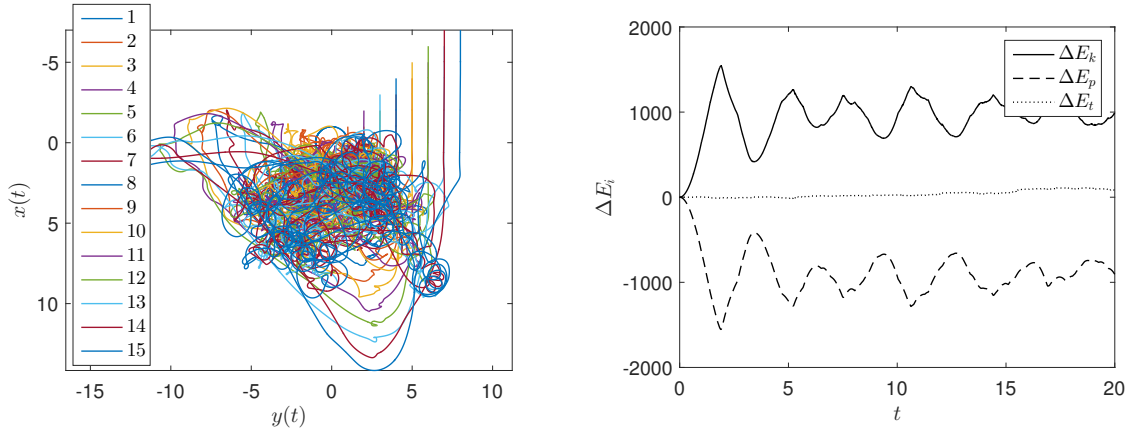


Figure 8: Minimal coordinates AMB method, 15 links, chaotic initial conditions. Plot of the trajectories of the end of each link (left) and the changes in energy (right). Even for this very large amount of links, the total energy only drifts a small amount over time relative to the mean values of the kinetic and potential energies.

causes the maximal coordinates methods to become inaccurate and drift over time. The links are no longer explicitly informed to remain attached at the hinges, but now only informed not to accelerate relative to one another. After they gain a bit of relative velocity due to finite numerical precision, they will begin to drift away from satisfying the original, undifferentiated kinematic constraints.

The linear momentum balance equations on each link take the form

$$m_i(\ddot{x}_{G_i} \hat{\mathbf{i}} + \ddot{y}_{G_i} \hat{\mathbf{j}}) = \mathbf{R}_{i+1} - \mathbf{R} + m_i g \hat{\mathbf{i}} \quad (8)$$

where $\mathbf{R}_{N+1} = \mathbf{0}$, and the angular momentum balance equations take the form

$$\mathbf{r}_{R_i/G_i} \times \mathbf{R}_i + \mathbf{r}_{R_{i+1}/G_i} \times \mathbf{R}_{i+1} = I^{G_i} \ddot{\theta}_i \hat{\mathbf{k}} \quad (9)$$

where the position vectors are

$$\mathbf{r}_{R_i/G_i} = -d_i(\cos \theta \hat{\mathbf{i}} + \sin \theta \hat{\mathbf{j}}), \quad \mathbf{r}_{R_{i+1}/G_i} = (\ell_i - d_i)(\cos \theta \hat{\mathbf{i}} + \sin \theta \hat{\mathbf{j}}) \quad (10)$$

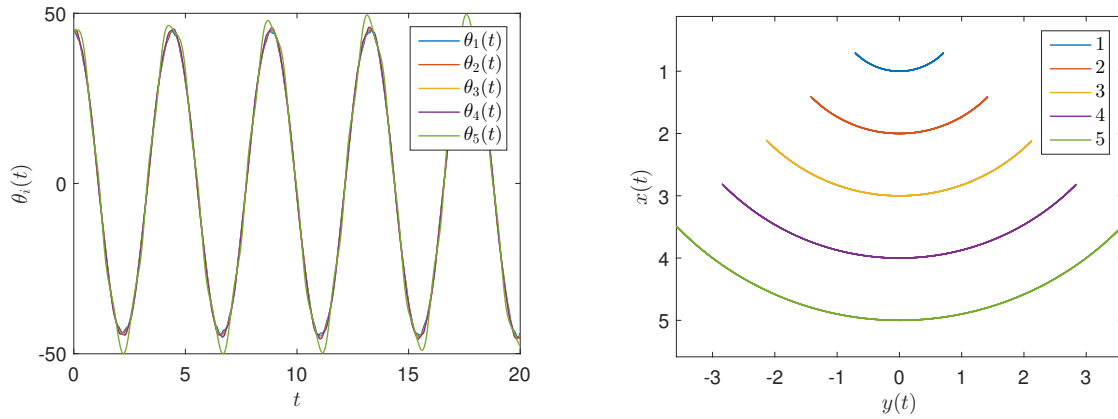


Figure 9: Minimal coordinates AMB method, 5 links, nice initial conditions. The case where the last link is much more massive than the other links. Plot of the angles each link (left) and the trajectories of the end of each link (right). The resulting motion is extremely uniform and matches that of a single pendulum. This is due to the last link being very massive compared to the other (100 times more massive), which for these nice initial conditions could be approximated as a point mass on a string: a simple pendulum.

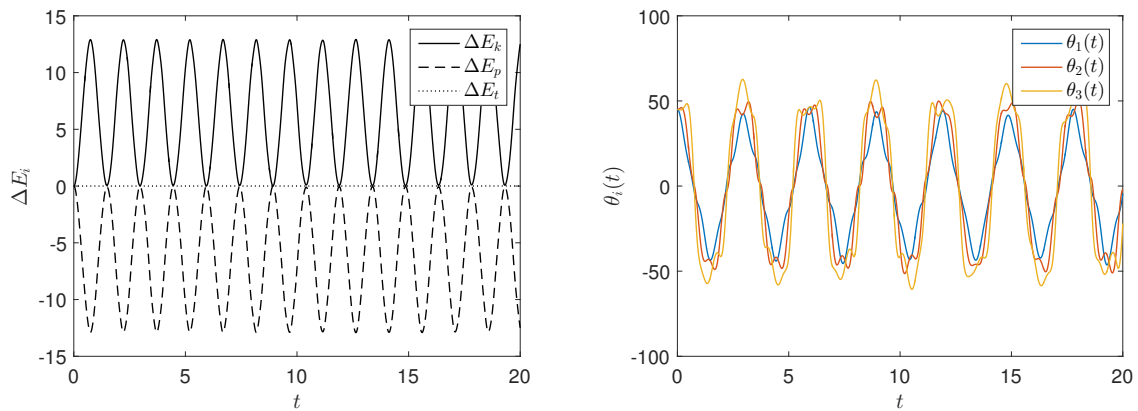


Figure 10: Minimal coordinates Lagrange equations method, 3 links, nice initial conditions. Plot of the changes in energy (left) and the angles of each link (right). For this simple case, not taking the simplification step does not affect the result, as these angles are identical to those of the AMB method in Fig. 1.

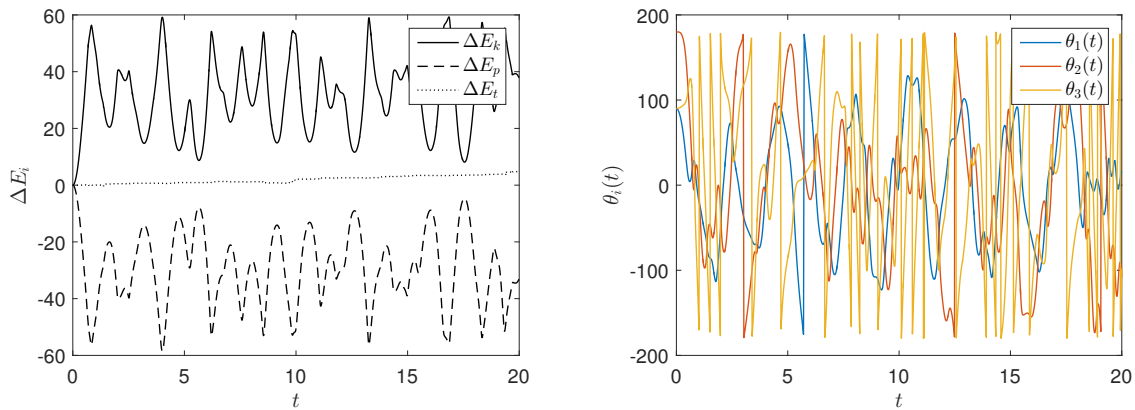


Figure 11: Minimal coordinates Lagrange equations method, 3 links, chaotic initial conditions. Plot of the changes in energy (left) and the angles of each link (right). For this more complex case, not taking the simplification step has caused these angle to become different from those of the AMB method in Fig. 3.

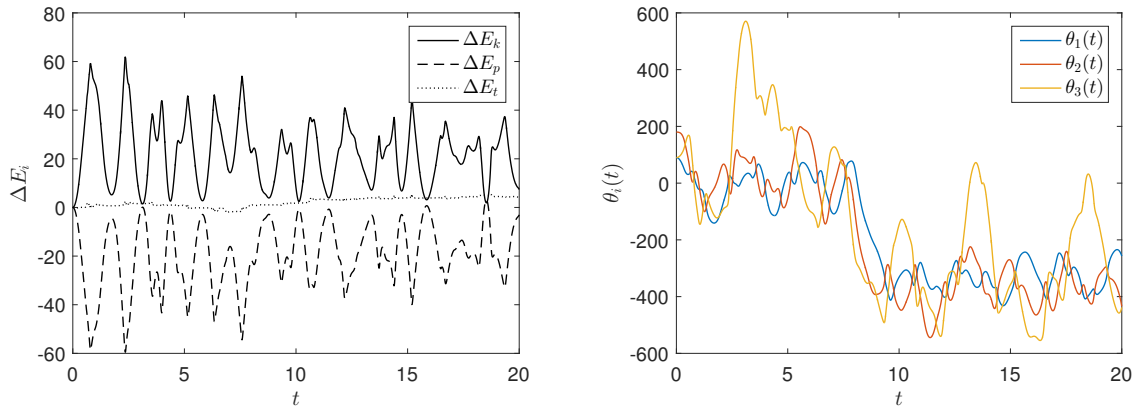


Figure 12: Minimal coordinates Lagrange equations method, 3 links, chaotic initial conditions. Torsional springs included. Plot of the changes in energy (left) and the angles of each link (right). This shows that this method also properly handles torsional springs, as the change in total energy only drift a bit and can be attributed to error from finite numerical precision over fast-changing chaotic motion.

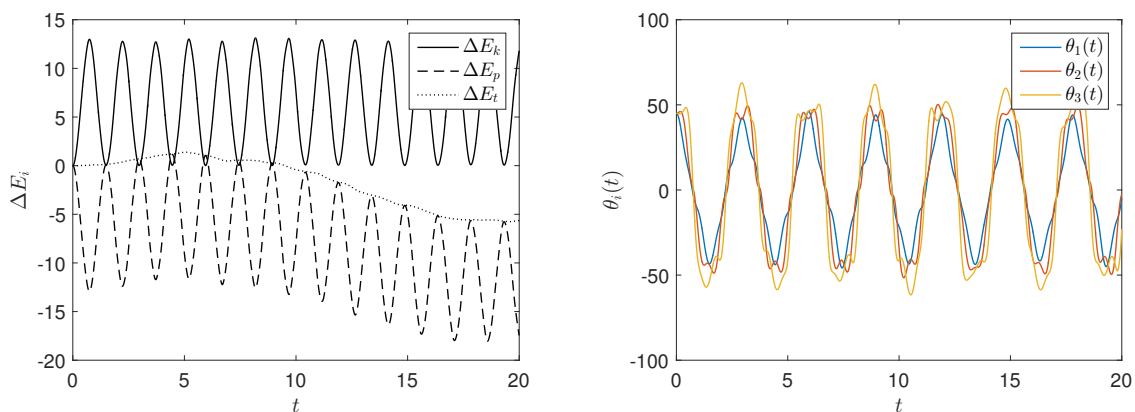


Figure 13: Maximal coordinates method, 3 links, nice initial conditions. Plot of the changes in energy (left) and the angles of each link (right). Even though the angles match the past cases, see that the change in total energy drifts a great deal, due to the drift in satisfying the kinematic constraints.

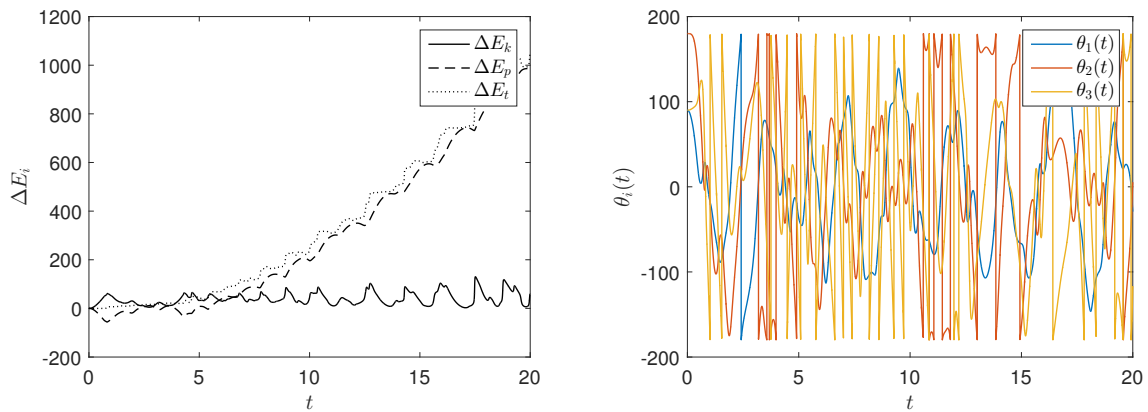


Figure 14: Maximal coordinates method, 3 links, chaotic initial conditions. Plot of the changes in energy (left) and the angles of each link (right). The change in total energy is much too high for this to be accurate. The drift in satisfying the kinematic constraints is very large due to the chaotic motion.

N-Link Closed Loop

Taking the free end of the last link of the N -link pendulum and constraining it to a fixed position creates an N -link closed loop. Two more unknowns are introduced, which are the x and y components of the reaction force at the end, \mathbf{R}_{end} , and two more equations are introduced, which are the kinematic constraint equations (x and y directions) $\mathbf{r}_{\text{end/O}} = \text{constant}$, which need to be differentiated twice. These extra equations and unknowns can be supplemented into the minimal coordinates angular momentum balance and maximal coordinates methods above. For the minimal coordinates angular momentum balance method, there is an additional moment $\mathbf{r}_{\text{end/A}_i} \times \mathbf{R}_{\text{end}}$ in each of the angular momentum balance equations. For the maximal coordinates method, the last link has an extra force \mathbf{R}_{end} applied. For the minimal coordinates Lagrange equations method, generalized forces must be used.

This part was attempted but not successfully completed. The corresponding additions to the scripts below for it have been taken out. The integration of the equations of motion for a four-bar mechanism failed any time two links overlapped in what is referred to as dead positions by [3]. Similar failure was observed for five bar mechanisms.

A.10 Fall 2017 Final Problem 4

Comment: This problem is included here to illustrate an idea concerning why for “nice” initial conditions and certain choices for the masses and moments of inertia of the links, all the links remain in-line throughout the motion. The problem is solved here for the case of three links (a final exam problem from the course), then for any number of links (something I added on myself).

A triple pendulum is released from rest in the position shown. All three links have the same moment of inertia I about their centers of mass. Two of the links, OA and AB, have mass $= 0^*$. Because OA and AB have no mass, you can, if you need to, locate their centers of mass any place you like. Find the angular accelerations, $\ddot{\theta}_{OA}$, $\ddot{\theta}_{AB}$, $\ddot{\theta}_{BG}$, of the three links, just after release. Give all three answers in terms of some or all of m , I , ℓ , and g .

*Conceptual clarification, not needed for solving the problem. The concept of an object with non-zero moment of inertia but with zero mass is a bit unphysical, but is mechanically sensible. A massless inertia is just as sensible as, say, an object with non-zero mass but with zero moment of inertia, otherwise known as ‘a point mass’. For example, the case of $I^G \neq 0$ and $m = 0$ does not cause difficulties in the project code you presented to Andy earlier this week. (*Assuming your code is working correctly!*) You can try this problem with your own code after the exam. You could get better and better approximations of such an object by making larger and larger objects with smaller and smaller mass. For example, imagine a rigid rod of length 12000 m that had a mass of 106 kg. It would have a notable inertia of $I^G = m\ell^2/12 = 12 \text{ kg} \cdot \text{m}^2$, and a, perhaps negligible, mass of 1 milligram.

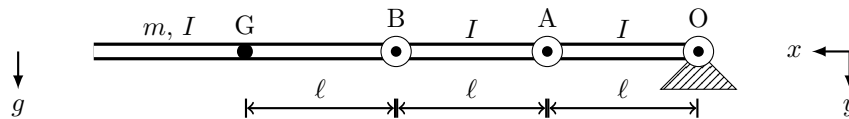


Figure 15: Diagram for Fall 2017 Final Problem 4.

One way to solve this problem is by writing the angular momentum balance about each hinge. In preparation, write the acceleration of G as

$$\mathbf{a}_G = \mathbf{a}_{A/O} + \mathbf{a}_{B/A} + \mathbf{a}_{G/B} \quad (11)$$

At this instant, each ${}_i\hat{\mathbf{e}}_r$ is aligned with $\hat{\mathbf{i}}$ and each ${}_i\hat{\mathbf{e}}_\theta$ is aligned with $\hat{\mathbf{j}}$, so

$$\mathbf{a}_G = -\ell(\dot{\theta}_{OA}^2 + \dot{\theta}_{AB}^2 + \dot{\theta}_{BG}^2)\hat{\mathbf{i}} + \ell(\ddot{\theta}_{OA} + \ddot{\theta}_{AB} + \ddot{\theta}_{BG})\hat{\mathbf{j}} \quad (12)$$

The angular momentum balances equations about O, A, and B dotted with $\hat{\mathbf{k}}$ are

$$(I + 3m\ell^2)(\ddot{\theta}_{OA} + \ddot{\theta}_{AB} + \ddot{\theta}_{BG}) = 3mg\ell \quad (13a)$$

$$I(\ddot{\theta}_{AB} + \ddot{\theta}_{BG}) + 2m\ell^2(\ddot{\theta}_{OA} + \ddot{\theta}_{AB} + \ddot{\theta}_{BG}) = 2mg\ell \quad (13b)$$

$$I\ddot{\theta}_{BG} + m\ell^2(\ddot{\theta}_{OA} + \ddot{\theta}_{AB} + \ddot{\theta}_{BG}) = mg\ell \quad (13c)$$

These equations are written in matrix form as

$$\begin{bmatrix} I + 3m\ell^2 & I + 3m\ell^2 & I + 3m\ell^2 \\ 2m\ell^2 & I + 2m\ell^2 & I + 2m\ell^2 \\ m\ell^2 & m\ell^2 & I + m\ell^2 \end{bmatrix} \cdot \begin{bmatrix} \ddot{\theta}_{OA} \\ \ddot{\theta}_{AB} \\ \ddot{\theta}_{BG} \end{bmatrix} = mg\ell \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \quad (14)$$

Subtract the last two rows from the first for

$$\begin{bmatrix} I & 0 & -I \\ 2m\ell^2 & I + 2m\ell^2 & I + 2m\ell^2 \\ m\ell^2 & m\ell^2 & I + m\ell^2 \end{bmatrix} \cdot \begin{bmatrix} \ddot{\theta}_{OA} \\ \ddot{\theta}_{AB} \\ \ddot{\theta}_{BG} \end{bmatrix} = mg\ell \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix} \quad (15)$$

Now subtract twice the third row from the second for

$$\begin{bmatrix} I & 0 & -I \\ 0 & I & -I \\ m\ell^2 & m\ell^2 & I + m\ell^2 \end{bmatrix} \cdot \begin{bmatrix} \ddot{\theta}_{OA} \\ \ddot{\theta}_{AB} \\ \ddot{\theta}_{BG} \end{bmatrix} = mg\ell \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (16)$$

Then consider the two equations from the first two rows: they show $\ddot{\theta}_{OA} = \ddot{\theta}_{BG}$ and $\ddot{\theta}_{AB} = \ddot{\theta}_{BG}$, and therefore all three are equal. Go back to Eq. (13a) to now write

$$\boxed{\ddot{\theta}_{OA} = \ddot{\theta}_{AB} = \ddot{\theta}_{BG} = \frac{mg\ell}{I + 3m\ell^2}} \quad (17)$$

A.10.1 Bonus For Fall 2017 Final Problem 4

This was not part of the original exam, and resulted from a thought a few months later. Consider an N -link pendulum with similar conditions as the above problem: negligible mass in each link except the last, all starting at initial angle θ_0 and initial velocity $\dot{\theta}_0$, but now of unequal lengths and unequal moment of inertias. Denote the hinges H_j , the link lengths ℓ_j , the link angles θ_j , and the link moment of inertias $I_j^{G_j}$ about their centers of mass, where H_1 is in initial (fixed) hinge, and m_N is the mass of the final link. Write the angular momentum balance about some intermediate j^{th} link, which is

$$\sum_{i=j}^N I_i^{G_i} \ddot{\theta}_i = m_N \left(\sum_{i=j}^N \ell_i \right) \left(g \sin \theta_0 - \sum_{i=1}^N \ell_i \ddot{\theta}_i \right) \quad (18)$$

Consider writing the same for the $j+1^{\text{th}}$ link and subtracting the two equations. Many of the terms cancel and the result is

$$I_j^{G_j} \ddot{\theta}_j = m_N \ell_j \left(g \sin \theta_0 - \sum_{i=1}^N \ell_i \ddot{\theta}_i \right) \quad (19)$$

which is then rearranged to receive

$$\frac{I_j^{G_j}}{\ell_j} \ddot{\theta}_j = m_N \left(g \sin \theta_0 - \sum_{i=1}^N \ell_i \ddot{\theta}_i \right) \quad (20)$$

See that for any j^{th} link, the right hand side of this equation will be identical, so now it has been retrieved that at this instant,

$$\frac{I_j^{G_j}}{\ell_j} \ddot{\theta}_j = \text{constant for all } j \quad (21)$$

This shows that for $\ddot{\theta}_j = \ddot{\theta}$ to be true (all links have the same $\ddot{\theta}$), it is only necessary that each link have the same $\frac{I_j^G}{\ell}$, not necessarily each of the two individual terms the same. Additionally note that there is nothing special about this instant or initial angle. Stepping through infinitesimally small time increments,

$$d\theta_j = \dot{\theta}_j dt + \frac{1}{2} \ddot{\theta}_j dt^2, \quad d\dot{\theta}_j = \ddot{\theta}_j dt \quad (22)$$

At the start, all links have the same angle, angular velocity, and angular acceleration. By the above, the increment in angular velocity and angle will be the same for each link, leaving them once again at the

same angle and angular velocity, and therefore the same angular acceleration. Summarizing, in an N -link pendulum where the mass of each link is negligible compared to a massive N^{th} link, where each link has the same $\frac{I_j^G}{\ell}$ and they all start at the same angle and angular velocity, they will remain at the same angle for all time, moving together as a simple pendulum. Using the above equations with $\alpha \equiv \frac{I_j^G}{m_N \ell_j} = \text{constant}$, it appears that the equation of motion for this homogenized system takes the form

$$\ddot{\theta} + \frac{g}{\alpha + \sum_{i=1}^N \ell_i} \sin \theta = 0 \quad (23)$$

An example of this phenomenon with 5 links is shown back in Fig. 9.

Relevant Scripts

Visit [MATLAB Central](#) to download the functions `ODEPROG.m` and `ODEABORT.m` created by [1] and `GIF.m` created by [2], used in the following scripts.

p48_ROOT.m

```
% Michael R. Buche
% MAE 5730 - Problem 48
% Final Computation Project
% Root file

clear; close all; clc;

% Method of obtaining equations of motion
p.method = 'minimal_AMB';
%p.method = 'minimal_Lagrange';
%p.method = 'maximal';

% Number of links
p.N = 5;

% Timespan
tspan = [0 20];

% Options for simple automatically set parameters
p.simple_system = true;
p.nice_initial_conditions = true;
p.torques = false;
p.springs = false;
p.friction = false;

% Parameters setup
if p.simple_system
    p = p48_simple_system(p);
else
    % define parameters yourself here!
end

% Plotting fontsize
p.fontsize = 16;

% ODE45 accuracy settings
opts.reltol = 1e-8;
opts.abstol = 1e-8;

%% Solve for the motion
[t,x_G,y_G,x_end,y_end,theta,theta_dot,v_G] = p48_solve(tspan,p);

%% Option to load saved runs and avoid waiting for solver
cd 'Saved Workspaces'
%load('minimal_AMB-chaotic-15_links-20_seconds.mat');
%load('minimal_AMB-chaotic-8_links-20_seconds_torques_springs_friction.mat');
%load('minimal_AMB-nice-6_links-20_seconds.mat');
%load('minimal_AMB-chaotic-6_links-20_seconds.mat');
```

```

%load('minimal_AMB-nice-5_links-20_seconds_point.mat');
%load('minimal_AMB-nice-3_links-20_seconds.mat');
%load('minimal_AMB-chaotic-3_links-20_seconds.mat');
%load('minimal_AMB-chaotic-3_links-40_seconds_springs_friction.mat');
%load('minimal_AMB-chaotic-3_links-20_seconds_springs.mat');
%load('minimal_Lagrange-nice-5_links-20_seconds_point.mat');
%load('minimal_Lagrange-nice-3_links-20_seconds.mat');
%load('minimal_Lagrange-chaotic-3_links-20_seconds.mat');
%load('minimal_Lagrange-chaotic-3_links-20_seconds_springs.mat');
%load('maximal-nice-3_links-20_seconds.mat');
%load('maximal-chaotic-3_links-20_seconds.mat');
cd ..

```

```

%% Plot the motion
p48_plot(t,x_end,y_end,theta,p)

```

```

%% Plot the changes in energy
p48_energy(t,x_G,v_G,theta,theta_dot,p);

```

```

%% Animate the motion
p.animation_timescale = 1;
p.timer = false; % F.Y.I. using this can make animation choppy
p.hinges = true;
p48_animate(t,x_end,y_end,p);

```

```

%% Create and save a gif
p.num_points = 50*tspan(2);
p.timer = true;
p.gif_file_name = 'name.gif';
p48_gif(t,x_end,y_end,p);

```

p48_simple_system.m

```

% Michael R. Buche
% MAE 5730 - Problem 48
% Final Computation Project
% Simple automatic parameters function

```

```

function p = p48_simple_system(p)

    % Uniform links of unit mass and unit length
    p.g = 9.81; % do not make zero
    p.l = ones(1,p.N);
    p.d = p.l/2;
    p.m = ones(1,p.N);
    p.I_G = p.m.*p.l.^2/12;

    % Links start at rest in line at 45 degrees
    if p.nice_initial_conditions
        theta0 = pi/4;
        p.icv = [theta0*ones(1,p.N) zeros(1,p.N)];

    % Links start at rest in an upward zig-zag
    else

```

```

        angle_1 = pi/2;
        angle_2 = pi;
        half_1 = angle_1*toeplitz(mod(1,2),mod(1:p.N,2));
        half_2 = angle_2*(1 - toeplitz(mod(1,2),mod(1:p.N,2)));
        p.icv = [half_1+half_2 zeros(1,p.N)];
    end

    % Torque applied at each hinge
    if p.torques
        p.T = ones(1,p.N);
    end

    % Linear torsional springs at each hinge
    if p.springs
        p.k = ones(1,p.N);
    end

    % Linear angular viscous friction at each hinge
    if p.friction
        p.c = ones(1,p.N);
    end
end

```

p48_solve.m

```

% Michael R. Buche
% MAE 5730 - Problem 48
% Final Computation Project
% Solve for the motion

function [t,fx_G,fy_G,fx_end,fy_end,ftheta,fdtheta,fv_G] = p48_solve(tspan,p)

    % Start timer to record time taken to solve
    tic

    % Halt certain cases that are not permissible or not yet set up
    if strcmp(p.method,'minimal_Lagrange')
        assert(~p.friction,'Cannot use Lagrange equations while there is friction!')
        assert(~p.torques,'Lagrange equations method not set up for external torques.')
    end
    if strcmp(p.method,'maximal')
        assert(~p.friction,'Maximal coordinates method not set up for friction.')
        assert(~p.torques,'Maximal coordinates method not set up for external torques.')
        assert(~p.springs,'Maximal coordinates method not set up for torsional springs.')
    end

    % Display some information about the system
    disp(['Initializing variables for ',num2str(p.N), ' link pendulum,'])
    if p.simple_system
        if p.nice_initial_conditions
            disp('    nice initial conditions,')
        else
            disp('    not nice initial conditions,')
        end
    end
end

```

```

    if p.torques
        disp('    torques applied at hinges,')
    else
        disp('    no applied torques,')
    end
    if p.springs
        disp('    torsional springs included,')
    else
        disp('    no torsional springs,')
    end
    if p.friction
        disp('    hinge frictions included,')
    else
        disp('    no hinge frictions,')
    end
end

% Fixed frame unit vectors
i_hat = [1 0 0];
j_hat = [0 1 0];
k_hat = [0 0 1];

% Minimal coordinates method
if strcmp(p.method,'minimal_AMB')
    disp('    using minimal coordinates and AMB...')

    % Initialize symbolic variables
    syms t
    theta = sym([]);
    er = sym([]);
    et = sym([]);
    rG_0 = sym([]);
    rG_A = sym([]);
    rA_B = sym([]);
    r_hingei_Gj = sym([]);
    aG_0 = sym([]);
    aG_A = sym([]);
    aA_B = sym([]);
    M_A = sym([]);
    H_dot_A = sym([]);
    eqns = sym([]);

    % Distance and acceleration vectors
    disp('Creating distance and acceleration vectors...')
    for i = 1:p.N
        theta(i) = symfun(sym(sprintf('theta_%d(t)', i)), t);
        er(i,:) = [cos(theta(i)) sin(theta(i)) 0];
        et(i,:) = cross(k_hat,er(i,:));
        rA_B(i,:) = p.l(i)*er(i,:);
        rG_A(i,:) = p.d(i)*er(i,:);
        rG_0(i,:) = rG_A(i,:);
        if (i > 1)
            for j = 1:(i-1)
                rG_0(i,:) = rG_0(i,:) + rA_B(j,:);
            end
        end
    end
end

```

```

        end
    end
    aA_B(i,:) = cross(diff(theta(i),2)*k_hat,rA_B(i,:))...
        - diff(theta(i))^2*rA_B(i,:);
    aG_A(i,:) = cross(diff(theta(i),2)*k_hat,rG_A(i,:))...
        - diff(theta(i))^2*rG_A(i,:);
    aG_0(i,:) = aG_A(i,:);
    if (i > 1)
        for j = 1:(i-1)
            aG_0(i,:) = aG_0(i,:) + aA_B(j,:);
        end
    end
end

% Distance from ith hinge to jth center of mass
disp('Creating additional distance vectors...')
for i = 1:p.N
    for j = 1:p.N
        r_hingei_Gj(i,:,j) = rG_A(j,:);
        if (j > 1)
            for k = i:(j-1)
                r_hingei_Gj(i,:,j) = r_hingei_Gj(i,:,j) + rA_B(k,:);
            end
        end
    end
end

% Angular momentum balance equations
disp('Writing equations of motion...')
for i = 1:p.N
    M_A(i,:) = cross(rG_A(i,:),p.m(i)*p.g*i_hat);
    H_dot_A(i,:) = p.m(i)*cross(rG_A(i,:),aG_0(i,:))...
        + p.I_G(i)*diff(theta(i),2)*k_hat;
    if (i < p.N)
        for j = (i+1):p.N
            M_A(i,:) = M_A(i,:) + cross(r_hingei_Gj(i,:,j),p.m(j)*p.g*i_hat);
            H_dot_A(i,:) = H_dot_A(i,:)...
                + p.m(j)*cross(r_hingei_Gj(i,:,j),aG_0(j,:))...
                + p.I_G(j)*diff(theta(j),2)*k_hat;
        end
    end
    if p.torques
        M_A(i,3) = M_A(i,3) + p.T(i);
    end
    if p.springs
        if i == 1
            M_A(1,3) = M_A(1,3) - p.k(1)*theta(1);
        else
            M_A(i,3) = M_A(i,3) - p.k(i)*(theta(i) - theta(i-1));
        end
    end
    if p.friction
        if i == 1
            M_A(1,3) = M_A(1,3) - p.c(1)*diff(theta(1));
        end
    end
end

```

```

        else
            M_A(i,3) = M_A(i,3) - p.c(i)*(diff(theta(i)) - diff(theta(i-1)));
        end
    end
    eqns(i) = M_A(i,3) - H_dot_A(i,3);
end
end

% Maximal coordinates method
if strcmp(p.method,'maximal')
    disp('    using maximal coordinates...')

    % Initialize symbolic variables
    syms t
    theta = sym([]);
    x_G = sym([]);
    y_G = sym([]);
    x_end = sym(zeros(1,p.N));
    y_end = sym(zeros(1,p.N));
    R_x = sym([]);
    R_y = sym([]);
    eqns_LMB = sym([]);
    eqns_AMB = sym([]);
    eqns_con = sym([]);

    % Symbolic functions of time
    for i = 1:p.N
        theta(i) = symfun(sym(sprintf('theta_%d(t)', i)), t);
        x_G(i) = symfun(sym(sprintf('x_G_%d(t)', i)), t);
        y_G(i) = symfun(sym(sprintf('y_G_%d(t)', i)), t);
        R_x(i) = symfun(sym(sprintf('R_x_%d(t)', i)), t);
        R_y(i) = symfun(sym(sprintf('R_y_%d(t)', i)), t);
    end

    % Linear and angular momentum balance and constraint equations
    disp('Writing equations of motion')
    disp('    and constraint equations...')
    for i = 1:p.N
        if (i == 1)
            eqns_LMB(1,1) = p.m(1)*diff(x_G(1),2) - p.m(1)*p.g - R_x(1) - R_x(2);
            eqns_LMB(1,2) = p.m(1)*diff(y_G(1),2) - R_y(1) - R_y(2);
            eqns_AMB(1) = -p.d(1)*(R_x(1)*sin(theta(1)) - R_y(1)*cos(theta(1)))...
                - (p.l(1) - p.d(1))*(R_y(2)*cos(theta(1)) - R_x(2)*sin(theta(1)))...
                + p.I_G(1)*diff(theta(1),2);
            x_end(1) = p.l(1)*x_G(1)/p.d(1);
            y_end(1) = p.l(1)*y_G(1)/p.d(1);
            eqns_con(1,1) = diff(x_G(1) - p.d(1)*cos(theta(1)),2);
            eqns_con(1,2) = diff(y_G(1) - p.d(1)*sin(theta(1)),2);
        else
            if (i < p.N)
                eqns_LMB(i,1) = p.m(i)*diff(x_G(i),2) - p.m(1)*p.g...
                    + R_x(i) - R_x(i+1);
                eqns_LMB(i,2) = p.m(i)*diff(y_G(i),2) + R_y(i) - R_y(i+1);
                eqns_AMB(i) = -p.d(i)*(R_y(i)*cos(theta(i))...

```

```

        - R_x(i)*sin(theta(i)))...
        - (p.l(i) - p.d(i))*(R_y(i+1)*cos(theta(i)))...
        - R_x(i+1)*sin(theta(i)))...
        + p.I_G(i)*diff(theta(i),2);
    else
        eqns_LMB(p.N,1) = p.m(p.N)*diff(x_G(p.N),2) - p.m(1)*p.g + R_x(p.N);
        eqns_LMB(p.N,2) = p.m(p.N)*diff(y_G(p.N),2) + R_y(p.N);
        eqns_AMB(p.N) = -p.d(p.N)*(R_y(p.N)*cos(theta(p.N)))...
            - R_x(p.N)*sin(theta(p.N)))...
            + p.I_G(p.N)*diff(theta(p.N),2);
    end
    x_end(i) = x_end(i-1) + p.l(i)*(x_G(i) - x_end(i-1))/p.d(i);
    y_end(i) = y_end(i-1) + p.l(i)*(y_G(i) - y_end(i-1))/p.d(i);
    eqns_con(i,1) = diff(x_G(i) - x_end(i-1) - p.d(i)*cos(theta(i)),2);
    eqns_con(i,2) = diff(y_G(i) - y_end(i-1) - p.d(i)*sin(theta(i)),2);
end
end
eqns = [reshape(eqns_LMB,[2*p.N 1]); eqns_AMB.']; reshape(eqns_con,[2*p.N 1]);
end

% Lagrange equations method
if strcmp(p.method,'minimal_Lagrange')
    disp('    using minimal coordinates and Lagrange equations...')

    % Initialize symbolic variables
    syms t
    theta = sym([]);
    theta_p = sym([]);
    dtheta_p = sym([]);
    er = sym([]);
    et = sym([]);
    rG_0 = sym([]);
    rG_A = sym([]);
    rA_B = sym([]);
    v_G = sym([]);
    Ek = sym();
    Ep = sym();
    eqns = sym([]);

    % Distance and velocity vectors, kinetic and potential energy
    disp('Creating distance and velocity vectors...')
    for i = 1:p.N
        theta(i) = symfun(sym(sprintf('theta_%d(t)', i)), t);
        theta_p(i) = sym(sprintf('theta_p_%d', i));
        dtheta_p(i) = sym(sprintf('dtheta_p_%d', i));
        er(i,:) = [cos(theta_p(i)) sin(theta_p(i)) 0];
        et(i,:) = cross(k_hat,er(i,:));
        rA_B(i,:) = p.l(i)*er(i,:);
        rG_A(i,:) = p.d(i)*er(i,:);
        rG_0(i,:) = rG_A(i,:);
        if (i > 1)
            v_G(i,:) = v_G(i-1,:) + dtheta_p(i-1)*(p.l(i-1)-p.d(i-1))*et(i-1,:)+...
                + dtheta_p(i)*p.d(i)*et(i,:);
            for j = 1:(i-1)

```

```

        rG_0(i,:) = rG_0(i,:) + rA_B(j,:);
    end
else
    v_G(1,:) = dtheta_p(1)*p.d(1)*et(1,:);
end
Ek = Ek + 1/2*(p.m(i)*sum(v_G(i,:).^2) + p.I_G(i)*dtheta_p(i)^2);
Ep = Ep - p.m(i)*p.g*rG_0(i,:)*i_hat.';
if p.springs
    if i == 1
        Ep = Ep + 1/2*p.k(1)*theta_p(1)^2;
    else
        Ep = Ep + 1/2*p.k(i)*(theta_p(i) - theta_p(i-1))^2;
    end
end
end

% Lagrange equations
disp('Writing Lagrange equations...')
L = Ek - Ep;
for i = 1:p.N
    eqns(i) = subs(diff(L,theta_p(i)),[theta_p dtheta_p],[theta diff(theta)])...
        - diff(subs((diff(L,dtheta_p(i))),[theta_p dtheta_p],[theta diff(theta)]),t);
end
end

% Integrate the equations in ODE45
if (strcmp(p.method,'minimal_AMB') || strcmp(p.method,'minimal_Lagrange'))
    vars = theta;
elseif strcmp(p.method,'maximal')
    vars = [R_x R_y theta x_G y_G];
end
[new_eqns,new_vars] = reduceDifferentialOrder(eqns,vars);
[M,b] = massMatrixForm(new_eqns,new_vars);
if strcmp(p.method,'maximal')
    M_1 = M(:,2*p.N+1:end);
    c = sym('c%d', [1 2*p.N]);
    A = equationsToMatrix(subs(new_eqns(1:3*p.N),[R_x R_y],c),c);
    M_2 = [A; zeros(numel(new_vars)-numel(A(:,1)),2*p.N)];
    M_new = [M_2 M_1];
    q.all_vars = new_vars(2*p.N+1:end);
    q.M = M_new;
    q.b = subs(b,[R_x R_y],zeros(1,2*p.N));
    q.offset = 2*p.N;
else
    q.all_vars = new_vars;
    q.M = M;
    q.b = b;
    q.offset = 0;
end
disp('Integrating in ODE45...')
q.num_vars = numel(q.all_vars);

% Functions odeprog.m and odeabort.m created by Tim Franklin
% Downloaded from MATLAB Central File Exchange

```

```

opts = odeset('OutputFcn',@odeprog,'Events',@odeabort);

if (strcmp(p.method,'minimal_AMB') || strcmp(p.method,'minimal_Lagrange'))
    q.type = 1;
elseif strcmp(p.method,'maximal')
    q.type = 0;
    x_G_0 = zeros(1,p.N);
    y_G_0 = zeros(1,p.N);
    v_G_x_0 = zeros(1,p.N);
    v_G_y_0 = zeros(1,p.N);
    for i = 1:p.N
        if (i == 1)
            x_G_0(1) = p.d(1)*cos(p.icv(1));
            y_G_0(1) = p.d(1)*sin(p.icv(1));
            v_G_x_0(1) = -p.d(1)*p.icv(p.N+1)*sin(p.icv(1));
            v_G_y_0(1) = p.d(1)*p.icv(p.N+1)*cos(p.icv(1));
        else
            x_G_0(i) = x_G_0(i-1) + (p.l(i-1) - p.d(i-1))*cos(p.icv(i-1))...
                + p.d(i)*cos(p.icv(i));
            y_G_0(i) = y_G_0(i-1) + (p.l(i-1) - p.d(i-1))*sin(p.icv(i-1))...
                + p.d(i)*sin(p.icv(i));
            v_G_x_0(i) = v_G_x_0(i-1)...
                - (p.l(i) - p.d(i))*p.icv(p.N+i-1)*sin(p.icv(i-1))...
                - p.d(i)*p.icv(p.N+i)*sin(p.icv(i));
            v_G_y_0(i) = v_G_y_0(i-1)...
                + (p.l(i) - p.d(i))*p.icv(p.N+i-1)*cos(p.icv(i-1))...
                + p.d(i)*p.icv(p.N+i)*cos(p.icv(i));
        end
    end
    p.icv = [p.icv(1:p.N) x_G_0 y_G_0 p.icv(p.N+1:2*p.N) v_G_x_0 v_G_y_0].';
end
[t,z] = ode45(@p48_RHS,tspan,p.icv,opts,q);

% Store the variables
ftheta = zeros(numel(t),p.N);
fdtheta = zeros(numel(t),p.N);
fx_G = zeros(numel(t),p.N);
fy_G = zeros(numel(t),p.N);
fx_end = zeros(numel(t),p.N);
fy_end = zeros(numel(t),p.N);
fer = zeros(numel(t),3,p.N);
fet = zeros(numel(t),3,p.N);
fv_G = zeros(numel(t),3,p.N);

% Angles of links
for i = 1:p.N
    if (strcmp(p.method,'minimal_AMB') || strcmp(p.method,'minimal_Lagrange'))
        ftheta(:,i) = z(:,i);
        fdtheta(:,i) = z(:,i+p.N);
    elseif strcmp(p.method,'maximal')
        ftheta(:,i) = z(:,i);
        fdtheta(:,i) = z(:,i+3*p.N);
    end
end
end

```

```

% Positions and velocities of link ends and centers of mass
for i = 1:p.N
    fx = 0;
    fy = 0;
    for j = 1:i-1
        fx = fx + p.l(j)*cos(ftheta(:,j));
        fy = fy + p.l(j)*sin(ftheta(:,j));
    end
    fx_end(:,i) = fx + p.l(i)*cos(ftheta(:,i));
    fy_end(:,i) = fy + p.l(i)*sin(ftheta(:,i));
    fer(:, :, i) = [cos(ftheta(:,i)) sin(ftheta(:,i)) zeros(numel(t),1)];
    if strcmp(p.method,'maximal')
        fx_G(:,i) = z(:,i+p.N);
        fy_G(:,i) = z(:,i+2*p.N);
        fv_G(:, :, i) = [z(:,i+4*p.N) z(:,i+5*p.N) zeros(numel(t),1)];
    else
        fx_G(:,i) = fx + p.d(i)*cos(ftheta(:,i));
        fy_G(:,i) = fy + p.d(i)*sin(ftheta(:,i));
        for j = 1:numel(t)
            fet(j, :, i) = cross(k_hat, fer(j, :, i));
            if (i == 1)
                fv_G(j, :, 1) = fdtheta(j,1)*p.d(1)*fet(j, :, 1);
            else
                fv_G(j, :, i) = fv_G(j, :, i-1)...
                    + fdtheta(j,i-1)*(p.l(i-1)-p.d(i-1))*fet(j, :, i-1)...
                    + fdtheta(j,i)*p.d(i)*fet(j, :, i);
            end
        end
    end
end
end

% Output time taken to solve
time = toc;
if (time < 60)
    disp(['Took ', num2str(toc), ' seconds to solve.'])
else
    disp(['Took ', num2str(toc/60), ' minutes to solve.'])
end
end

```

p48_RHS.m

```

% Michael R. Buche
% MAE 5730 - Problem 48
% Final Computation Project
% Right-hand side function

```

```

function dz = p48_RHS(~,z,q)
    M = vpa(subs(q.M,q.all_vars,z));
    b = vpa(subs(q.b,q.all_vars,z));
    x = M\b;
    dz = zeros(q.num_vars,1);
    for i = 1:q.num_vars

```

```

        dz(i) = x(i+q.offset);
    end
end

```

p48_plot.m

```

% Michael R. Buche
% MAE 5730 - Problem 48
% Final Computation Project
% Plot the motion

function p48_plot(t,x_end,y_end,theta,p)

% Plot the angles of each link if any turn over
links_wrap_around = max(max(abs(theta - wrapToPi(theta)))) > 0;
if links_wrap_around
    figure
    for i = 1:p.N
        plot(t,rad2deg(theta(:,i)),'linewidth',1);
        hold on
        leg{i} = ['$\theta_{'} sprintf('%d',i) '(t)$'];
    end
    hold off
    legendo = legend(leg);
    set(legendo,'Interpreter','latex','FontSize',p.fontsize)
    xlabel('$t$', 'Interpreter','LaTeX');
    ylabel('$\theta_i(t)$', 'Interpreter','LaTeX');
    set(gca,'FontSize',p.fontsize);
end

% Plot the angles of each link wrapped to pi
figure
for i = 1:p.N
    plot(t,rad2deg(wrapToPi(theta(:,i))),'linewidth',1);
    hold on
    leg{i} = ['$\theta_{'} sprintf('%d',i) '(t)$'];
end
hold off
legendo = legend(leg);
set(legendo,'Interpreter','latex','FontSize',p.fontsize)
xlabel('$t$', 'Interpreter','LaTeX');
ylabel('$\theta_i(t)$', 'Interpreter','LaTeX');
set(gca,'FontSize',p.fontsize);

% Plot the path of the ends of each link
figure
for i = 1:p.N
    plot(y_end(:,i),x_end(:,i),'linewidth',1);
    hold on
    leg{i} = sprintf('%d',i);
end
hold off
set(gca,'YDir','reverse')
legendo = legend(leg);

```

```

set(legendo,'Interpreter','latex','FontSize',p.fontsize)
xlabel('$y(t)$','Interpreter','LaTeX');
ylabel('$x(t)$','Interpreter','LaTeX');
set(gca,'FontSize',p.fontsize);
axis equal

% Plot the path of the end of the last link
figure
plot(y_end(:,end),x_end(:,end),'k','linewidth',1);
legun = sprintf('%d',p.N);
set(gca,'YDir','reverse')
legendo = legend(legun);
set(legendo,'Interpreter','latex','FontSize',p.fontsize)
xlabel('$y(t)$','Interpreter','LaTeX');
ylabel('$x(t)$','Interpreter','LaTeX');
set(gca,'FontSize',p.fontsize);
axis equal
end

```

p48_energy.m

```

% Michael R. Buche
% MAE 5730 - Problem 48
% Final Computation Project
% Plot total energy

function p48_energy(t,x_G,v_G,theta,dtheta,p)

% Kinetic and potential energy
Ek = zeros(1,numel(t));
Ep = zeros(1,numel(t));
for i = 1:numel(t)
    Ek(i) = 1/2*(p.m*squeeze(dot(v_G(i,:),v_G(i,:)))) + p.I_G*dtheta(i,:).^2;
    Ep(i) = -p.m*p.g*x_G(i,:).';
    if p.springs
        for j = 1:p.N
            if j == 1
                Ep(i) = Ep(i) + 1/2*p.k(1)*(theta(i,1))^2;
            else
                Ep(i) = Ep(i) + 1/2*p.k(j)*(theta(i,j) - theta(i,j-1))^2;
            end
        end
    end
end
end

% Total energy
Et = Ek + Ep;

% Plot energy versus time
figure
plot(t,Ek-Ek(1),'k',t,Ep-Ep(1),'k--',t,Et-Et(1),'k:', 'linewidth',1)
legendo = legend('$\Delta E_k$', '$\Delta E_p$', '$\Delta E_t$');
set(legendo,'Interpreter','latex','FontSize',p.fontsize)
xlabel('$t$', 'Interpreter','LaTeX');

```

```

        ylabel('$\Delta E_i$', 'Interpreter', 'LaTeX');
        set(gca, 'fontsize', p.fontsize);
    end

```

p48_animate.m

```

% Michael R. Buche
% MAE 5730 - Problem 48
% Final Computation Project
% Animate the motion

function p48_animate(t,x_end,y_end,p)

    % Create figure and plot axes
    figure;
    grid on
    grid minor
    axis equal
    axis([-1 1 -1 1]*1.1*sum(p.1))
    hold on
    set(gca, 'YDir', 'reverse')
    xlabel('$y(t)$', 'Interpreter', 'LaTeX');
    ylabel('$x(t)$', 'Interpreter', 'LaTeX');
    set(gca, 'fontsize', p.fontsize);

    % Create links and hinges
    link = zeros(1,p.N);
    hinge = zeros(1,p.N);
    line_width = 2;
    if p.hinges
        hinge_size = 5;
    else
        hinge_size = 1;
    end
    link(1) = plot([0 y_end(1,1)], [0 x_end(1,1)], 'k', 'linewidth', line_width);
    hinge(1) = plot(0,0, 'ko', 'markersize', hinge_size, ...
        'MarkerFaceColor', 'w', 'linewidth', line_width);
    for i = 2:p.N
        link(i) = plot([y_end(1,i-1) y_end(1,i)], [x_end(1,i-1) x_end(1,i)], ...
            'k', 'linewidth', line_width);
        hinge(i) = plot(y_end(1,i-1), x_end(1,i-1), 'ko', 'markersize', hinge_size, ...
            'MarkerFaceColor', 'w', 'linewidth', line_width);
    end

    % Create timer textbox if opted to do so
    if p.timer
        timer_text = annotation(gcf, 'textbox', [0.825 0.75 0.1 0.1], ...
            'string', {'$t = 0.00$'}, 'fontsize', p.fontsize, 'Interpreter', 'LaTeX');
    end

    % Update positions of the links and hinges in real time
    pause(1)
    xint = zeros(1,p.N);
    yint = zeros(1,p.N);

```

```

time = 0;
tic

while time < t(end)/p.animation_timescale
    for i = 1:p.N
        xint(i) = interp1(t,x_end(:,i),time*p.animation_timescale);
        yint(i) = interp1(t,y_end(:,i),time*p.animation_timescale);
        if i == 1
            set(link(i),'xdata',[0 yint(1)],'ydata',[0 xint(1)]);
        else
            set(hinge(i),'xdata',yint(i-1),'ydata',xint(i-1))
            set(link(i),'xdata',[yint(i-1) yint(i)],'ydata',[xint(i-1) xint(i)])
        end
    end
    drawnow;
    time = toc;
    % Update timer
    if p.timer
        set(timer_text,'string',sprintf('$t = %.2f$',time*p.animation_timescale));
    end
end
end

```

p48_gif.m

```

% Michael R. Buche
% MAE 5730 - Problem 48
% Final Computation Project
% Make GIF of animation

function p48_gif(t,x_end,y_end,p)

    % Make sure too many points are not selected
    if (p.num_points > numel(t))
        disp('Too many points selected for GIF.')
        return
    end
    % Create figure and plot axes
    figure;
    grid on
    grid minor
    axis equal
    axis([-1 1 -1 1]*1.1*sum(p.l))
    hold on
    set(gca,'YDir','reverse')
    xlabel('$y(t)$','Interpreter','LaTeX');
    ylabel('$x(t)$','Interpreter','LaTeX');
    set(gca,'fontsize',p.fontsize);

    % Create links and hinges
    link = zeros(1,p.N);
    hinge = zeros(1,p.N);
    line_width = 2;
    if p.hinges

```

```

        hinge_size = 5;
    else
        hinge_size = 1;
    end
    link(1) = plot([0 y_end(1,1)], [0 x_end(1,1)], 'k', 'linewidth', line_width);
    hinge(1) = plot(0,0, 'ko', 'markersize', hinge_size, ...
        'MarkerFaceColor', 'w', 'linewidth', line_width);
    for i = 2:p.N
        link(i) = plot([y_end(1,i-1) y_end(1,i)], [x_end(1,i-1) x_end(1,i)], ...
            'k', 'linewidth', line_width);
        hinge(i) = plot(y_end(1,i-1), x_end(1,i-1), 'ko', 'markersize', hinge_size, ...
            'MarkerFaceColor', 'w', 'linewidth', line_width);
    end

    % Create timer textbox if opted to do so
    if p.timer
        timer_text = annotation(gcf, 'textbox', [0.825 0.75 0.1 0.1], ...
            'string', {'$t = 0.00$'}, 'fontsize', p.fontsize, 'Interpreter', 'LaTeX');
    end

    % Interpolate times and positions to evenly spaced and less points
    t_int = linspace(t(1), t(end), p.num_points);
    delay_time = t_int(3) - t_int(2);
    fx_end_int = zeros(numel(t_int), p.N);
    fy_end_int = zeros(numel(t_int), p.N);
    for i = 1:p.N
        fx_end_int(:, i) = interp1(t, x_end(:, i), t_int).';
        fy_end_int(:, i) = interp1(t, y_end(:, i), t_int).';
    end

    % Function gif.m created by Chad Greene
    % Downloaded from MATLAB Central File Exchange
    pause(1)
    gif(p.gif_file_name, 'DelayTime', delay_time, 'frame', gcf);

    % Update positions of the links and hinges frame by frame
    for j = 1:numel(t_int)
        for i = 1:p.N
            if i == 1
                set(link(i), 'xdata', [0 fy_end_int(j,1)], 'ydata', [0 fx_end_int(j,1)]);
            else
                set(hinge(i), 'xdata', fy_end_int(j,i-1), 'ydata', fx_end_int(j,i-1))
                set(link(i), 'xdata', [fy_end_int(j,i-1) fy_end_int(j,i)], ...
                    'ydata', [fx_end_int(j,i-1) fx_end_int(j,i)])
            end
        end
        drawnow;
        % Update timer if engaged
        if p.timer
            set(timer_text, 'string', sprintf('$t = %.2f$', t_int(j)));
        end
        % Make gif
        gif;
    end
end

```

end

Bibliography

- [1] Tim Franklin. *ODE Progress Bar and Interrupt*. 2006. Retrieved from MATLAB Central on 19 November 2017, at <https://www.mathworks.com/matlabcentral/fileexchange/9904-ode-progress-bar-and-interrupt>.
- [2] Chad Greene. *An Easy gif Generator*. 2017. Retrieved from MATLAB Central on 20 November 2017, at <https://www.mathworks.com/matlabcentral/fileexchange/63239-gif>.
- [3] Michael M Stanisic. *Mechanisms and Machines: Kinematics, Dynamics, and Synthesis*. Cengage Learning, 2014.