AMATH 301
Homework 10
Due: Friday, March 15, 2019

Nonlinear Pendulum

Let's return to the linear pendulum from Homework 9,

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = \frac{-g}{l}\sin\theta, \qquad 0 \le t \le T. \tag{1}$$

with $g = 9.8$ and $l = 10.0$.

If we say that the pendulum is attached with a massless, rigid rod rather than a string, the equation is valid even when θ is large (the pendulum becomes inverted or upside down).

Problem 1 (Scorelator).

Consider the IVP

$$\begin{cases} \dfrac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = \dfrac{-g}{l}\sin\theta, \qquad 0 \le t \le T \\ \theta(0) = 3.1 \\ \theta'(0) = 0. \end{cases} \tag{2}$$

These initial conditions correspond to the pendulum initially in a near-vertical position ($\theta \approx \pi = 3.14159...$).

Solve the IVP using `ode45` with default settings for $T = 100$. Output the solution with grids-pacing $\Delta t = 0.1$. Save the first 500 rows of the numeric solution of $\theta(t)$ to **A1.dat**, and the first 500 rows of the numeric solution of $\phi(t)$ to **A2.dat**.

Problem 2 (Writeup).

Plot the numerical solutions for $\theta(t)$ and $\phi(t)$ from Problem 1 versus time on the same axes. Add a label for the horizontal axis (time), a title ('Physically Unrealistic Numerical Solution'), and add a legend. Save your figure as `non_conservative_pendulum.png`.

You should notice that the solution you obtained corresponds to a pendulum behaving in a very strange way: the pendulum rocks back and forth one time, and on the second time, it actually continues over the apex of its trajectory and begins rotating around the pivot over and over. This shouldn't happen in a conservative system, where energy is constant!

Problem 3 (Writeup).

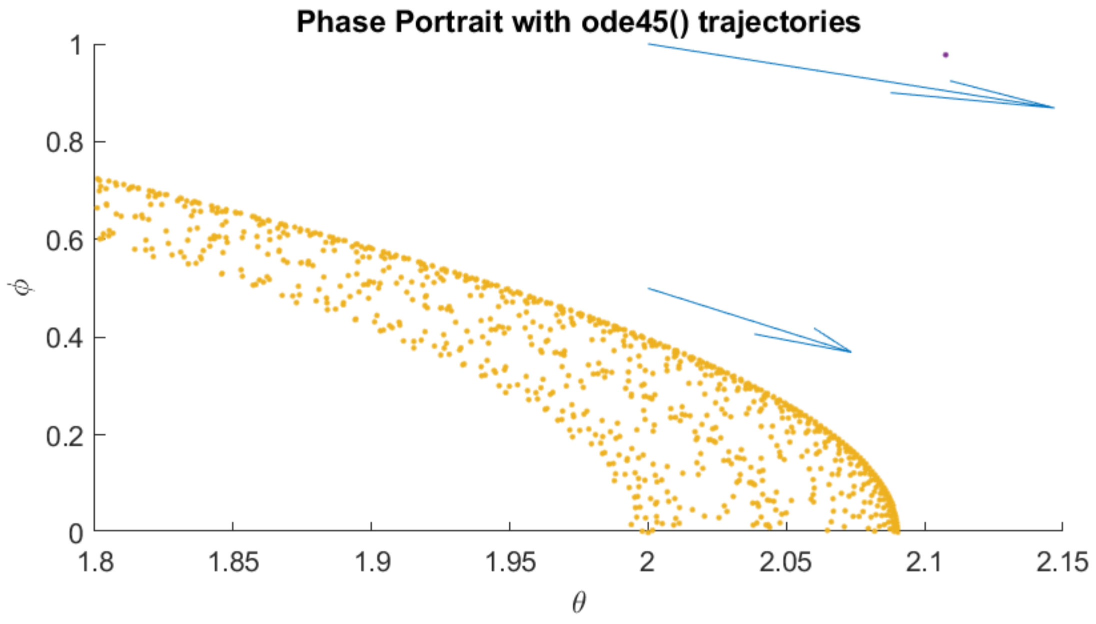
Construct a phase portrait for the system, like the one you made in Homework 9 for the nonlinear pendulum. Set the axes limits to be $-6 < \theta < 6$ and $-4 < \phi < 4$, and use quiver to plot a grid of arrows with a spacing of 0.5 from -6 to $+6$ on the θ axis and -4 to $+4$ on the ϕ axis.

Add trajectories corresponding to solutions of the system over the time interval $\mathbf{0 < t < T = 1000}$ (this end time T is larger than in the previous problem), and use the following three initial values:

- $\theta(0) = 1, \theta'(0) = 0.$
- $\theta(0) = 2, \theta'(0) = 0.$
- $\theta(0) = 3.1, \theta'(0) = 0.$

Plot each trajectory with small dots at each point (use `'.'`); Add labels for the axes and a title ("Phase Portrait with ode45 Trajectories"). In the next part of the homework we will generate the same phase portrait with a different numerical method, so distinguishing the two portraits is important. Save your figure as `pendulum_phase_ode45.png`.

From your phase portrait, you should see that the first two trajectories encircle the origin many times. Zooming in closer, the trajectories don't perfectly line up with themselves — the trajectories tend to either slip inwards to the origin, or spiral outwards. The effect is subtle, but indicates that energy is being lost or gained.



Symplectic Euler

Symplectic Euler is a modification of the standard Forward Euler scheme that works well for some systems, when the system has 'position-like' and 'velocity-like' variables x and v, and is of the form

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = F(x,v) \\ \dfrac{\mathrm{d}v}{\mathrm{d}t} = G(x,v). \end{cases}$$

When used properly, it conserves energy in the system. The scheme is as follows:

$$\begin{aligned} X_{k+1} &= X_k + \Delta t F(X_k, V_k) \\ V_{k+1} &= V_k + \Delta t G(X_{k+1}, V_k) \end{aligned}$$

Note that the update to V involves the most recently computed value of X.

Symplectic Euler is only first-order accurate, but if one is very concerned about the system gaining or losing energy, this low order may be acceptable.

Problem 4 (Scorelator).

Again consider the IVP from Problem 1,

$$\begin{cases} \dfrac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = \dfrac{-g}{l}\sin\theta, \qquad 0 \le t \le T \\ \theta(0) = 3.1 \\ \theta'(0) = 0. \end{cases} \tag{3}$$

with $T = 100$.

Solve the IVP using `symplecticEuler`, the symplectic Euler solver from class, with $T = 100$. Here θ is the 'position-like' variable and ϕ is the 'velocity-like' variable. Note that you do not need to convert these variables to cartesian coordinates; you don't need to do much work at all. Use a timestep size of $\Delta t = 0.1$. Save the first 500 rows of the numeric solution of $\theta(t)$ to **B1.dat**, and the first 500 rows of the numeric solution of $\phi(t)$ to **B2.dat**.

Recall that `symplecticEuler` is called slightly different than `ode45`. The 'ODEFUN' function must be split into two parts, one for the position variables and the other for the velocity variables:

```
[tsol,Xsol,Vsol] = symplecticEuler(f,g,tspan,X0,V0)
```

Problem 5 (Writeup).

Plot the numerical solutions for $\theta(t)$ and $\phi(t)$ from Problem 4 versus time on the same axes. Add a label for the horizontal axis (time), a title ('Energy-Conserving Numerical Solution'), and add a legend. Save your figure as `conservative_pendulum.png`.

The solution you obtain should look better than the one you obtained and plotted in Problems 1 and 2. In this solution, the pendulum does not behave unpredictably.

Problem 6 (Writeup).

Construct a phase portrait for the system, like you did in Problem 3 above, with the only difference being that the trajectories are generated with Symplectic Euler with a step size of $\Delta t = 0.1$ rather than `ode45`. Remember to solve up to time $T = 1000$. Save your figure as `pendulum_phase_sympEuler_dt_0.1.png`.

Now solve the system with the same initial conditions, but now solving the system with Symplectic Euler with $\Delta t = 0.5$ (a larger step size). Re-generate your phase portrait, and save your figure as `pendulum_phase_sympEuler_dt_0.5.png`.

You should see that the trajectories appear different than before. Energy is being conserved as desired (which you can observe by zooming in on the trajectories and seeing that the dots all fall on the same curve), but the loss of accuracy manifests in a distorted phase portrait.

Tensegrity

Tensegrity is an architectural principle where tensile and compressive forces are balanced to produce rigid structures. Rigid struts are connected by tendons such that the struts cannot move and do not touch one another. A simple tensegrity structure is shown below

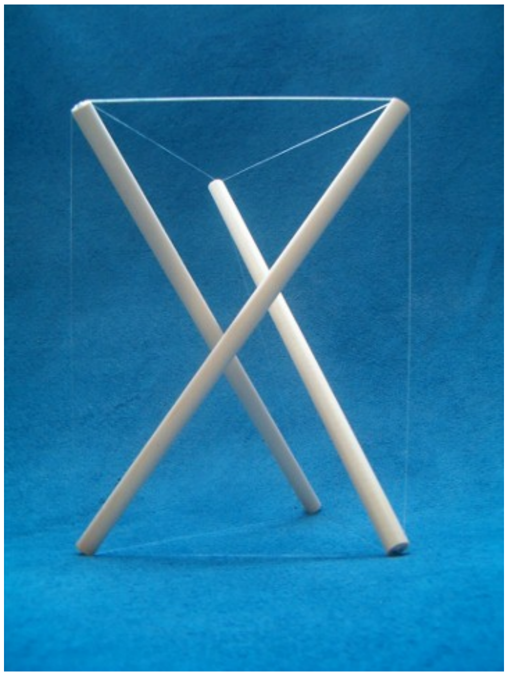
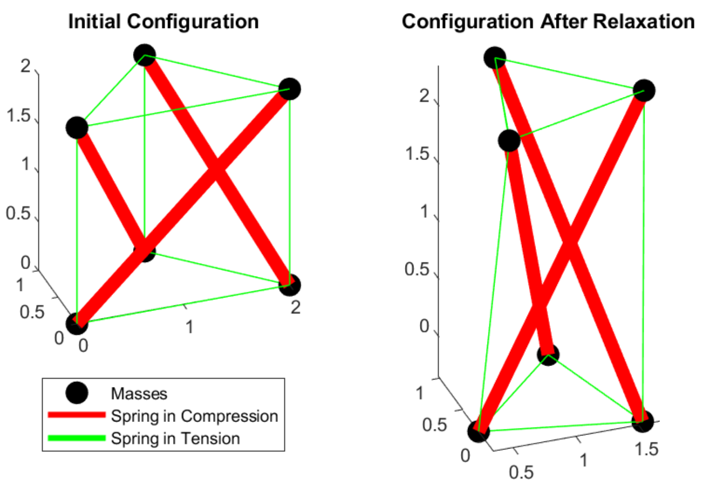


Figure 1: The T-prism tensegrity structure. Photo credit Marcelo Pars.

One approach to designing these structures is to model them as spring-mass systems. The endpoints of each rod are considered as nodes (masses), and each rod is a very stiff spring that connects two pairs of nodes. Each tendon is modeled as a short and easily-stretched spring. By advancing the system forward in time from an approximate initial condition, the system 'relaxes' as the springs push and pull the nodes to a stable resting position.



Each of the N nodes has a three-dimensional position, and we store these positions in $N \times 3$ matrix X. The nodes each have a velocity in three-dimensional space, which we store in a matrix V of the same size as X.

The spring-mass system obeys the following system of ordinary differential equations,

$$\begin{cases} \dfrac{\mathrm{d}\vec{X}}{\mathrm{d}t} = \vec{V} \\ \dfrac{\mathrm{d}\vec{V}}{\mathrm{d}t} = F_{\text{spring}}(\vec{X}) - c\vec{V}. \end{cases} \tag{4}$$

Here $f_{\text{spring}}(\vec{X})$ is a function that computes the force on each node due to the springs in the system. c is a frictional constant. Without friction, the system would oscillate and bounce around forever. With friction, the system tends towards rest and eventually settles down into a proper configuration for a tensegrity structure, $\vec{X}_{\text{rest}}$. At this point, the forces $F_{\text{spring}}(\vec{X}_{\text{rest}})$ should be zero (hence every element of the structure is at rest).

Because the springs have very different stiffnesses, this system tends to spiral out of control if energy is not conserved. For this reason, a method like symplectic Euler, that does not allow energy to grow, is favorable. In this case, the system actually does lose energy due to friction, but in a controlled manner; symplectic Euler ensures that energy is bled out of the system due to friction, so that the system will tend towards its rest state.

Problem 7 (Scorelator).

Load the file `tprism_spec.mat`. This file contains a rough initial configuration for the masses corresponding to the tensegrity pictured above. The variable `nodes` contains the initial configuration of the nodes: this is the initial value for the position matrix X. The variable `springs` contains a list of springs in the form of a matrix. Each row specifies a starting and ending node, the rest length of the spring, and the stiffness of the spring. Here we model the struts as springs with stiffness constant $k = 100$, and the tendons as springs with stiffness $k = 1$.

Use the function `calcSpringForces` that was supplied with this homework to calculate the forces on the nodes according to the list of springs. This function is called as follows:

```
F = calcSpringForces(X,springList);
```

Here `X` is the position of the nodes in the system, `springList` is the list of springs, and `F` is the resulting forces on each node. Save the forces on the nodes to file as **C1.dat**.

Problem 8 (Scorelator).

Write MATLAB functions `dXdt` and `dVdt` that compute the right-hand side functions for the system (4). Both functions have to be functions of t, X, V.

Use these functions and the supplied `symplecticEuler` function to solve the system (4). Set the friction coefficient $c = 5$. Use the variable `nodes` as the initial value of X, a matrix of zeros the same size as `nodes` as the initial value for V, and a time span of 0 to 50 in steps of $\Delta t = 0.01$.

Save the solution vectors for X and V output by `symplecticEuler` to the files **D1.dat** and **D2.dat**.

Problem 9 (Scorelator).

Extract the final positions of the nodes from the numeric solution you obtained in Problem 8 — this is the last row of the solution vector for X. These are the 'rest positions' for the nodes, so call this variable `Xrest`. Save this variable to file as **E1.dat**.

Problem 10 (Scorelator).

Use `calcSpringForces` to calculate the forces on the nodes in their new positions as well as the lengths of all of the springs when the system is at rest. Save the force matrix to file as **F1.dat**. You should find that the forces are very small (nearly zero).

The function `calcSpringForces` has a second use beyond calculating the forces on each node: if you ask for two output arguments instead of one, it will return the original list of springs with an additional column added: the current length of each spring.

```
[F,augmentedSpringList] = calcSpringForces(Xrest,springs);
```

Save a vector containing the length of each spring to file as **F2.dat**. The springs should be in the same order as the original `springs` matrix.

Problem 11 (Writeup).

The `drawSpringMassSystem` function can be used to draw spring-mass system. It is executed as follows,

```
drawSpringMassSystem(X,springs);
```

Here `X` is a matrix of node positions, and `springs` is a list of springs connecting the nodes. This function performs a task similar to the bridge-drawing problem from Homework 3.

Use `drawSpringMassSystem` to produce a plot of the system in its relaxed state. Use the `view` command to set the viewing angles to `view([-10 30])`. Save your figure to file as `tprism_side.png`. Then set the viewing angle to `view([0 90])`, and save your figure to file as `tprism_above.png`.