COMS W4167: Mass-Spring Systems

Programming Assignment 1

Due 10:00:00 PM Wed., Feb. 11th, 2026 (EST)

Introduction

Welcome to COMS 4167, Physically Based Computer Animation!

Our first *theme* for this course focuses on how to *get things moving*, i.e., how to integrate a dynamical system forward in time. We will implement a particle system that moves under external forces, and advance this system forward in time using a number of different integration methods, discovering the advantages and disadvantages of each.

Chapter 1

Notes

Mechanics deals with the kinematics and dynamics of a mechanical system. Kinematics *describes* the motion of a system (*e.g.*, as relations among position, velocity, and acceleration) while dynamics identifies the *causes* of the motion of a system (*e.g.*, the forces).

1.1 Kinematics

Configuration

A mechanical system can typically be in one of many possible positions. We will denote a specific position, or *configuration*, by $\mathbf{q}$. We use the boldface notation in typeset documents (and an underline when writing on the blackboard).

Since a system could be in one of many configurations, we can also refer to the set of *all* possible configurations as the *configuration space*, Q.

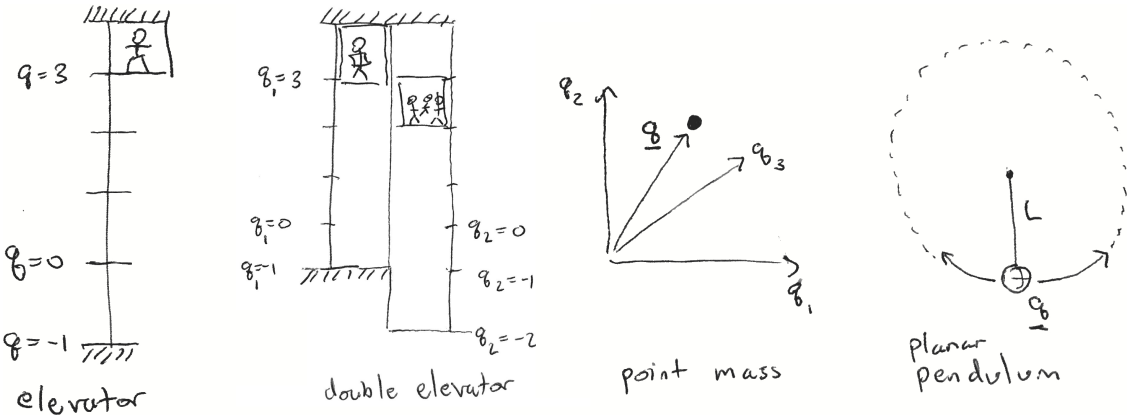

Figure 1.1: *Examples of configuration spaces.* From left to right: $Q \equiv [-1, 3]$, $Q \equiv [-1, 3] \times [-2, 3]$, $Q \equiv \mathbb{R}^3$, $Q \equiv S^1(L)$ (the circle or radius L).

Consider the examples in Figure 1.1:

- For an elevator that can travel only up and down, from floor -1 to floor 3, the configuration space is $Q \equiv [-1, 3]$, an interval subset of the real number line.
- A double-shaft elevator system has one elevator that can reach the basement (floor -1), and another that can reach all the way to the double-basement (floor -2); the configuration space $Q \equiv [-1, 3] \times [-2, 3]$

is the product of two intervals, i.e., any point $(q_1, q_2) = \mathbf{q} \in Q$ can be specified by the position of each of the elevators individually.

- For a point mass, or particle, in three dimensions, the configuration space is $Q \equiv \mathbb{R}^3$.
- For a pendulum of length L anchored at the origin and restricted to swing on the plane, the configuration space is a circle of radius L centered about the origin, each point on the circle corresponding to a possible position of the pendulum's bob.
- If the pendulum lives instead in three dimensions so that it can swing out of the plane, then its configuration space is given by the surface of a sphere.

Degrees of Freedom

We can talk about the dimensionality of a configuration space:

- The elevator has a one-dimensional configuration space Q, because (away from the limits, $q = -1$ and $q = 3$) the configuration space has the same topology as the real number line $\mathbb{R}^1$, i.e., it can be represented by a single real-valued coordinate (the height).
- The particle has a three-dimensional configuration space, because its configuration can be represented by three real-valued coordinates.
- The pendulum has a one-dimensional configuration space Q, because at any point on the circle $S^1(R)$, it can move only forwards and backwards, just like the elevator; its configuration can be represented by a signel real-valued coordinate (e.g., an angle measured relative to the positive x axis.

We say that a mechanical system with a k-dimensional configuration space has k ***degrees of freedom.***

We could say a lot more about these configuration spaces. For example, the elevator's configuration space is one-dimensional *with boundary* (it has limits $-1 \leq q \leq 3$, versus the pendulum's one-dimensional configuration space which has no boundary and is *periodic*, versus the particle's configuration space which neither has a boundary nor is periodic (it is infinite).

Trajectory

Suppose that we are interested in the motion of the system over some interval of time $[0, T] \subset \mathbb{R}$. The *trajectory* of the system is curve in configuration space, $\mathbf{q} : [0, T] \mapsto Q$, which maps any instant in time t to the configuration $\mathbf{q}(t)$ at that instant.

Referring to Fig. 1.2,

- the trajectory of our elevator, $\mathbf{q}(t) : [0, T] \mapsto [-1, 3]$, is a function that returns the height of the elevator for any given instant in time t, so that as t advances $\mathbf{q}(t)$ traces out a path that can go up and down along the interval $[-1, 3]$;
- the trajectory of a particle, $\mathbf{q}(t) : [0, T] \mapsto \mathbb{R}^3$, is a function that returns the three-dimensional position of the particle for any given instant in time t, so that as t advances $\mathbf{q}(t)$ traces out a path in three-dimensions;
- the trajectory of a planar pendulum is a function that returns a position on the circle for any given instant in time t, so that as t advances $\mathbf{q}(t)$ traces out a path along the circle, possibly doubling back on itself as the pendulum swings.
- *think for yourself:* what does the trajectory fo the double elevator look like?
- *think for yourself:* what does the trajectory of a spherical pendulum look like?

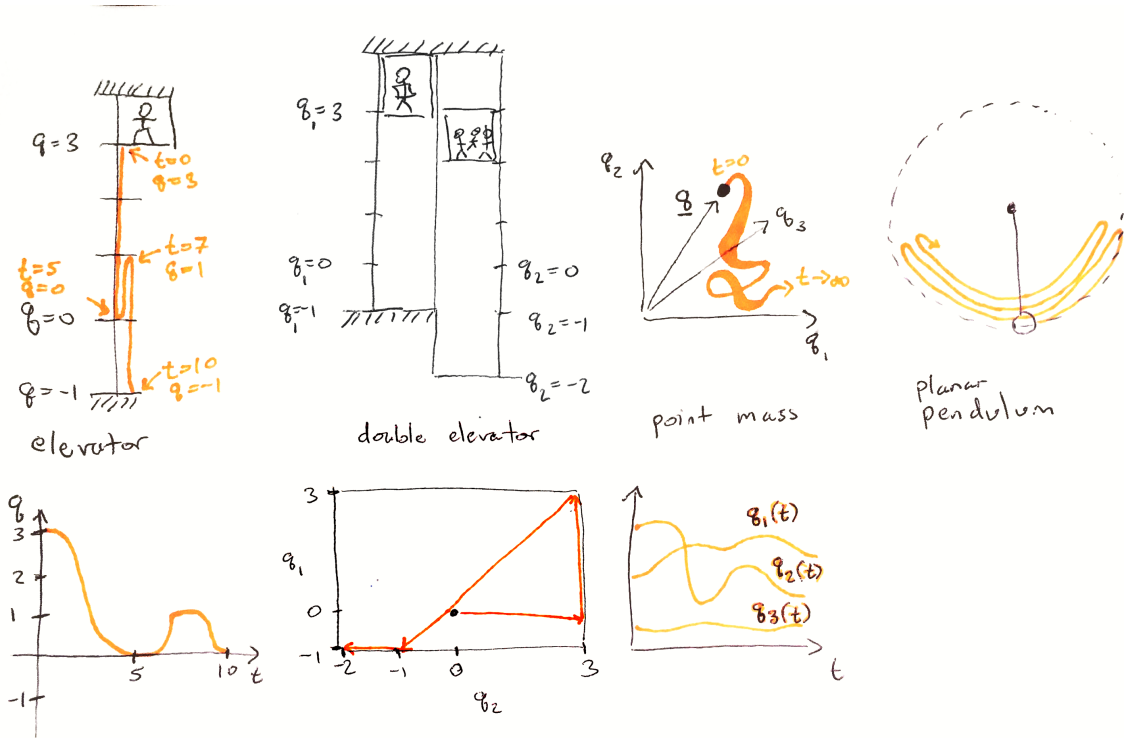

Figure 1.2: *Examples of trajectories.* From left to right: the configuration of the elevator evolves over a down-up-down trajectory *(top)*, which we can graph a real-valued function of time *(bottom)*; the configuration of the double-shafted elevator system can be plotted as a parametric curve $\mathbf{q}(t)$ in the rectangle $[-1,3]\times[-2,3] \subset \mathbb{R}^2$ *(bottom)*; *think for yourself:* can you trace out the path of the two elevators in configuration space *(above)*? The configuration of the particle evolves in a three dimensional curve *(top)*, as plotted in the graph of the three coordinates as function of time *(bottom)*; the pendulum swings back and forth in a trajectory that traverses an arc of the circle $S^1(L)$ back and forth. What coordinate system could we use to graph the trajectory of the pendulum?

Velocity

The velocity of the object, also called the *configurational velocity* or the *tangent vector* to the trajectory, $\mathbf{v}(t) = \dot{\mathbf{q}}(t) = \mathrm{d}\mathbf{q}(t)/\mathrm{d}t$, gives the instantaneous *rate* and *direction* of motion of the system. When the velocity is written in coordinates, the number of coefficients needed is the same as the dimensionality of the configuration space.

Referring to Fig. 1.3,

- an elevator rises and falls with velocity $\dot{q}(t) = \frac{dq}{dt}$;
- the velocity of the double elevator can be expressed in terms of two coefficients: $\dot{\mathbf{q}}(t) = (\dot{q}_1(t), \dot{q}_2(t))$.
- the velocity of a particle is given in coordinates by $\dot{\mathbf{q}}(t) = \frac{d\mathbf{q}}{dt} = (\frac{dq_1}{dt}, \frac{dq_2}{dt}, \frac{dq_1}{dt})$. Its speed is $v(t) = |\dot{\mathbf{q}}(t)|$ while its direction of motion is the unit vector $\hat{\mathbf{v}}(t) = \frac{\mathbf{v}(t)}{v(t)}$;
- for a planar pendulum, we can think of the configuration as being a radius of the circle (connecting the origin to a point on the circle), and the configurational velocity is always tangent to the circle, and therefore perpendicular to the radius;
- *think for yourself:* what are the only possible directions for the tangent vector for a spherical pendulum?

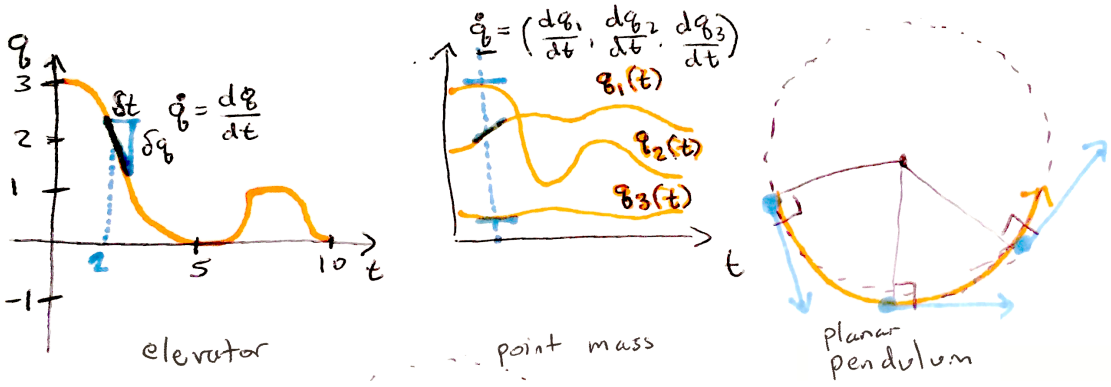

Figure 1.3: *Examples of velocities.*

State

Consider a projectile of given mass m and initial position $\mathbf{q}_0$, subject to only the force of gravity (see Fig. 1.4). How much more information do we need to predict the trajectory? Knowing the configuration is not sufficient, because different velocities will lead to different trajectories. If we know the position *and velocity*, then we can predict the trajectory under the influence of gravity (or other forces).

We define the *state* $\mathbf{y}$ of a mechanical system as a vector that concatenates the configuration and the configurational velocity, i.e., $\mathbf{y} = (\mathbf{q}, \mathbf{p})$. Because $\mathbf{q}$ and $\mathbf{p}$ have the same dimensions (they both require the same number of coefficients when written in coordinates), the state vector $\mathbf{y}$ is always of an even dimension.

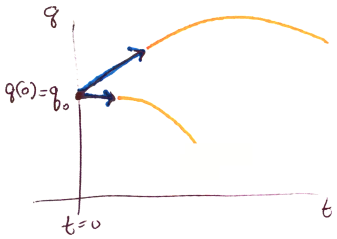

Figure 1.4: A projectile's trajectory depends on initial momentum.

Think for yourself: What is the dimensionality of the state for each of the systems surveyed in our figures above?

Sometimes, we define state in terms of *momentum* (instead of velocity); either approach is acceptable, as long as we remember to account for the mass in calculations that use the state. Let us elaborate on mass and momentum.

Mass, Momentum, and Kinetic Energy

Closely related to velocity is *momentum* $\mathbf{p} = M\dot{\mathbf{q}}$, the velocity scaled by the mass. While we often think of mass as a real-valued scalar m, observe that we have written it here as a matrix M. Two natural questions, then, are what are the dimensions of M, and, why do we write mass as a matrix? We can answer the first question by "type-checking:" if Q is k-dimensional, then $\mathbf{p}$ and $\dot{\mathbf{q}}$ are k-dimensional vectors. Two such vectors can be linearly related either by a scalar constant of proportionality, as in $\mathbf{p} = m\dot{\mathbf{q}}$, or by a $k \times k$ square matrix, as in $\mathbf{p} = M\dot{\mathbf{q}}$. We will need the latter relation, as it is more general than the former (*think for yourself: how can you implement the former in the latter?*). For example,

- for the particle, M is a 3×3 matrix,
- for the double elevator, M is a 2×2 matrix;
- of course, for a one-dimensional system, such as our single elevator, M is equivalently a 1×1 matrix or a scalar.

Why do we opt for the generality of the matrix? For a single elevator, $p = mv$. What about for the double elevator? Let the elevators have distinct masses m_1 and m_2. Considering each elevator in isolation,

we have $p_1 = m_1\dot{q}_1$ and $p_2 = m_2\dot{q}_2$. For the ensemble as a whole, we express the relation between $\mathbf{p} = (p_1, p_2)$ and $\dot{\mathbf{q}} = (\dot{q}_1, \dot{q}_2)$ via the linear system

$$\underbrace{\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}}_{\mathbf{p}} = \underbrace{\begin{pmatrix} m_1 & 0 \\ 0 & m_2 \end{pmatrix}}_{M} \underbrace{\begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \end{pmatrix}}_{\dot{\mathbf{q}}} .$$

Thus, for the double-elevator system, M is a 2×2 diagonal matrix with distinct diagonal entries.

Although we always think of a mass *matrix*, sometimes it does indeed act as a scalar. Consider a single particle in three dimensions; here $\mathbf{p} \in \mathbb{R}^3$ and $\dot{\mathbf{q}} \in \mathbb{R}^3$, and

$$\underbrace{\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}}_{\mathbf{p}} = \underbrace{\begin{pmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{pmatrix}}_{M} \underbrace{\begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix}}_{\dot{\mathbf{q}}} ,$$

which is equivalent to $\mathbf{p} = m\dot{\mathbf{q}}$.

Kinetic Energy The energy stored in the motion of a mechanical system is called *kinetic energy*. By definition, the kinetic energy is

$$\text{K.E.} = \frac{1}{2}\dot{\mathbf{q}}^T M \dot{\mathbf{q}} \tag{1.1}$$

$$= \frac{1}{2}\mathbf{p}^T \dot{\mathbf{q}} . \tag{1.2}$$

Observe how the product of the momentum and the velocity gives twice the kinetic energy. Whenever the product of two related quantities gives twice the kinetic energy, we say that the two quantities are *dual* under the kinetic energy. Thus, $\mathbf{p}$ and $\dot{\mathbf{q}}$ are dual quantities.

Energy is a *scalar* quantity. In Computer Science, we often think of scalars as any real-valued quantity. In physics, however, *scalars* have a more special role: they are quantities that *do not depend on the choice of coordinate system.* Thus, the kinetic energy of a mechanical system is a real number that does not depend on the choice of coordinates used to represent the configuration space. We elaborate on this while exploring matrices further.

Off-diagonals in mass matrix In some cases, the mass matrix is not diagonal. Let us explore the two elevator system further. Suppose we select a new choice of coordinates to describe the same configuration space. Let $q_{\text{sum}} = \frac{1}{\sqrt{2}}(q_1 + q_2)$, and $q_{\text{dif}} = \frac{1}{\sqrt{2}}(q_2 - q_1)$, i.e., we capture the positions of the two elevators in terms of their (scaled) sum and difference. In matrix form, this is

$$\underbrace{\begin{pmatrix} q_{\text{sum}} \\ q_{\text{dif}} \end{pmatrix}}_{\tilde{\mathbf{q}}} = \underbrace{\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}}_{R} \underbrace{\begin{pmatrix} q_1 \\ q_2 \end{pmatrix}}_{\mathbf{q}} ,$$

where $\mathbf{q}$ and $\tilde{\mathbf{q}}$ are the coordinates of the configuration expressed equivalently in the old and new coordinate systems. Since this system is invertible, we can go back to the original coordinate system via

$$\underbrace{\begin{pmatrix} q_1 \\ q_2 \end{pmatrix}}_{\mathbf{q}} = \underbrace{\begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}}_{R^{-1}} \underbrace{\begin{pmatrix} q_{\text{sum}} \\ q_{\text{dif}} \end{pmatrix}}_{\tilde{\mathbf{q}}} \tag{1.3}$$

By definition, the momentum in the new coordinate system is proportional to the velocity, $\tilde{\mathbf{p}} = \tilde{M}\dot{\tilde{\mathbf{q}}}$. What are the entries of the matrix $\tilde{M}$? To answer this question, we make use of a fundamental principle:

the kinetic energy of a system depends on its state, but not on the choice of coordinate system. If $(\mathbf{p}, \dot{\mathbf{q}})$ and $(\tilde{\mathbf{p}}, \dot{\tilde{\mathbf{q}}})$ represent the same state in two different coordinate systems, then the two representations must compute the same kinetic energy K.E. $= \mathbf{p}^T \dot{\mathbf{q}} = \tilde{\mathbf{p}}^T \dot{\tilde{\mathbf{q}}}$. Expanding using $\mathbf{p} = M\dot{\mathbf{q}}$

$$\frac{1}{2}\dot{\mathbf{q}}^T M \dot{\mathbf{q}} = \frac{1}{2}\dot{\tilde{\mathbf{q}}}^T \tilde{M} \dot{\tilde{\mathbf{q}}}$$

and substituting (1.4) gives

$$\frac{1}{2}\dot{\tilde{\mathbf{q}}}^T \underbrace{R^{-T} M R^{-1}}_{\tilde{M}} \dot{\tilde{\mathbf{q}}} = \frac{1}{2}\dot{\tilde{\mathbf{q}}}^T \tilde{M} \dot{\tilde{\mathbf{q}}}$$

from which we obtain

$$\tilde{M} = \frac{1}{2}\begin{pmatrix} m_1 + m_2 & m_2 - m_1 \\ m_2 - m_1 & m_1 + m_2 \end{pmatrix}. \tag{1.4}$$

Thus, in the new coordinates, the mass matrix is no longer diagonal (in fact, it is dense).

The mass matrix is always a symmetric matrix; the deepest reason for this is that it is a *metric*, but I think that a more direct intuitive explanation is possible as well: *Think for yourself:* Can you examine the definition of kinetic energy and convince yourself that allowing the mass matrix to be assymetric does not "buy" any additional flexibility in defining the physical system?

1.2 Newtonian Mechanics

Newton's laws Newtonian mechanics is based on Newton's three laws, which can be summarized as follows:

1. A body persists in its state of motion (at rest or moving at constant velocity) unless acted upon by an outside force.

2. The net force on a body equals the time rate of change of its momentum: $\mathbf{F} = \dot{\mathbf{p}}$. For a system with constant mass, this simplifies to $\mathbf{F} = \mathrm{d}(M\mathbf{v})/\mathrm{d}t = M\dot{\mathbf{v}} = M\mathbf{a}$.

3. When body A exerts a force on body B, body B exerts an equal (in magnitude) and opposite (in direction) force on body A.

Think for yourself: Consider three objects resting on the floor in a vertical stack. There is a constant force $\mathbf{f}_g$ on each pointing downwards due to gravity. Using Newton's laws, work out the additional forces acting on each object.

Equations of motion Assuming that we can determine the net forces acting on an object, Newton's second law allows us to determine the trajectory of that object given some initial state. The state of a particle is given by its position, $\mathbf{q}$, and momentum, $\mathbf{p}$, so that Newton's second law can be written as

$$\begin{pmatrix} \dot{\mathbf{q}}(t) \\ \dot{\mathbf{p}}(t) \end{pmatrix} = \begin{pmatrix} M^{-1}\mathbf{p}(t) \\ \mathbf{f}\left(\mathbf{q}(t), \mathbf{p}(t), t\right) \end{pmatrix} \tag{1.5}$$

This presents the equation of motion determining the particle's trajectory as two coupled first-order ordinary differential equations (ODEs). Here, we have explicitly allowed the forces to depend on time t. In certain cases, we may restrict our attention to forces that only depend on position and velocity, but not explicitly on time.

Using the relation $\mathbf{p} = M\dot{\mathbf{q}} = M\mathbf{v}$, we can rewrite the equations of motion in terms of position and velocity:

$$\begin{pmatrix} \dot{\mathbf{q}}(t) \\ M\dot{\mathbf{v}}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{v}(t) \\ \mathbf{f}\left(\mathbf{q}(t), \dot{\mathbf{q}}(t), t\right) \end{pmatrix}. \tag{1.6}$$

Our first goal in physical simulation is to formulate forces that describe the physical system we want to simulate, and then to solve (1.5) numerically to obtain the trajectory of the system through time.

Chapter 2

Assignment

2.1 Starter Code

After you download the starter code .zip file, **please read the README.md file carefully**. The README.md provides details about how to set up a Python virtual environment, and how to launch programs for each programming assignment. It also provides an overview of the code structure.

Before you start coding, please understand the data structure of the starter code, get a sense of where the information (such as spring stiffness, particle mass, inverse of mass, positions, forces, and velocities) are stored. You may refer to the "Code Overview" section in README.md as well.

Grading. Since in this semester, we completely redesigned the programming assignments, we will announce the grading criteria shortly. To get full credits, you need to make sure your program runs correctly for all scenes (i.e., all sceneXX.yml files) under scenes/pa1/. We will provide reference videos for you to compare with your simulation results visually (or qualitatively). Note that qualitative comparison of your results to the reference video *can not* gaurantee your implementation is absolutely correct.

Scene configuration. Your simulation will be configured through a scene file, which specifies the time integrator type, timestep size, particles and springs in the scene, and optionally visualization details. The files are in YAML format, and the data entries should be self-explained. Please refer to the scene files under scenes/pa1/ and read the comments in those files. Here is a brief breakdown:

```
solver:
  # integrator type. Choose from
  # - explicit_euler
  # - symplectic_euler
  # - midpoint
  type: midpoint  # symplectic_euler # explicit_euler
  timestep: 0.0005   # timestep size
```

This specifies time integrator type and timestep size (in seconds).

```
# A list of particles added explicitly
particles:
  # 1st particle
  - pos: [0, 0, 2]
    vel: [0, 0, 0]  # optional (default (0, 0, 0))
    fixed: true     # optional (default flase)
    mass: 0.1
```

```
  # 2nd particle
  - pos: [0, 0, 1]
    vel: [0, 0, 0]
    fixed: false
    radius: 0.2     # optional
    mass: 0.2
```

A list of particles. Here, fixed indicates if a particle should be fixed (never move). radius is the radius of the particle for visualization (not for simulation as we don't have particle-particle collection supported in this PA).

```
springs:
  # 1st spring
  # particle_ids are a pair of Ids indidcating which particles this spring
  # connects to. The Ids are zero-based (the first particle listed above has
  # an ID=0).
  - particle_ids: [0, 1]
    stiffness: 20.
    #damping: 0.5   # optional
    #rest_length: 0.4 # optional
```

A list of springs, each of which connects two particles specified by particle IDs. The particle IDs are *zero-based*.

2.2 Required Features for PA-1

You are asked to implement the following three time integrator algorithms:

2.2.1 Explicit Euler

We can discretize Newton's second law using explicit Euler, giving:

$$\mathbf{q}^{n+1} = \mathbf{q}^n + h\dot{\mathbf{q}}^n$$
$$\dot{\mathbf{q}}^{n+1} = \dot{\mathbf{q}}^n + hM^{-1}\mathbf{F}(\mathbf{q}^n, \dot{\mathbf{q}}^n)$$

Observe that both the position and the velocity update depend only on the position and velocity at the previous timestep.

TODO: Edit the provided source file src/nemo/solvers/explicit_euler.py, in particular the step method in that file, to compute the updated position and velocity using explicit Euler.

2.2.2 Symplectic Euler

From a *finite differencing* point of view, *Explicit Euler* can be interpreted as the following

$$\frac{\mathbf{q}^{n+1} - \mathbf{q}^n}{h} = \dot{\mathbf{q}}^n$$
$$\frac{\dot{\mathbf{q}}^{n+1} - \dot{\mathbf{q}}^n}{h} = \ddot{\mathbf{q}}^n = M^{-1}\mathbf{F}(\mathbf{q}^n, \dot{\mathbf{q}}^n)$$

Now take a closer look at the first equation. The equation uses $\dot{\mathbf{q}}^n$, the instantaneous change rate of $\mathbf{q}$ at the last time step n, to represent the average change rate of $\mathbf{q}$ between the current time step n and the next time step $n+1$. Of course this is only *approximately* true, and we'll look at the error introduced by this approximation (i.e. the *discretization error*) later in the course. Because the instantaneous point in

time picked to represent the average is the left endpoint (i.e. time step n) of the interval, this discretization scheme is called *forward differencing*. The second equation is thus the same forward differencing applied to $\dot{\mathbf{q}}$. It is important to realize that this is not the only reasonable way to do the approximation; for example $\dot{\mathbf{q}}^{n+1}$ is obviously as good an approximation of the average change rate between time step n and $n+1$ as is $\dot{\mathbf{q}}^{n+1}$. Using the right endpoint (i.e. time step $n+1$) is naturally called *backward differencing*. This yields the following integrator:

$$\frac{\mathbf{q}^{n+1} - \mathbf{q}^n}{h} = \dot{\mathbf{q}}^{n+1}$$
$$\frac{\dot{\mathbf{q}}^{n+1} - \dot{\mathbf{q}}^n}{h} = \ddot{\mathbf{q}}^{n+1} = M^{-1}\mathbf{F}(\mathbf{q}^{n+1}, \dot{\mathbf{q}}^{n+1})$$

The problem with this formulation is that the unknowns (the values for time step $n+1$) are present on both sides of the formula and it becomes an equation that needs to be solved, rather than a direct evaluation. We'll follow down this path next week – it leads to an important class of integrators (called *Implicit* integrators) that possess desirable properties at the cost of more computation. On the other hand, what else can we do without requiring solving equations? One possible scheme is to use forward differencing for $\dot{\mathbf{q}}$, and backward differencing for $\mathbf{q}$:

$$\frac{\mathbf{q}^{n+1} - \mathbf{q}^n}{h} = \dot{\mathbf{q}}^{n+1}$$
$$\frac{\dot{\mathbf{q}}^{n+1} - \dot{\mathbf{q}}^n}{h} = \ddot{\mathbf{q}}^{n} = M^{-1}\mathbf{F}(\mathbf{q}^{n}, \dot{\mathbf{q}}^{n})$$

Next-time-step quantity $\dot{\mathbf{q}}^{n+1}$ is still present on the right hand side of the first equation, but this term actually becomes known if we evaluate the second equation first. Therefore, no equation needs to be solved. This time integration scheme is called *Symplectic Euler*.

Symplectic Euler integrator is sometimes also called semi-implicit Euler or forward-backward Euler. We adopt the name "Symplectic Euler" as a standard. The 'update rule' is:

$$\dot{\mathbf{q}}^{n+1} = \dot{\mathbf{q}}^n + hM^{-1}\mathbf{F}(\mathbf{q}^n, \dot{\mathbf{q}}^n)$$
$$\mathbf{q}^{n+1} = \mathbf{q}^n + h\dot{\mathbf{q}}^{n+1}$$

Notice that the velocity update depends only on the position and velocity at the previous timestep, while the position update depends on the velocity at the current timestep. Contrast this with explicit Euler, where both updates depend on the previous step's position and velocity.

To qualitatively verify the behavior of symplectic Euler, compare to the behavior of explicit Euler with the test example scenes/pa1/scene01.yml. Switch back and forth the solver type in the solver section in scene01.yml: That is, use

```
solver:
  type: explicit_euler  # symplectic_euler # explicit_euler
  timestep: 0.0005   # timestep size
```

in comparision to

```
solver:
  type: symplectic_euler  # symplectic_euler # explicit_euler
  timestep: 0.0005   # timestep size
```

And watch the plot in the top-right sub-window. Your Explict Euler should produce a diverging plot curve, while your Symplectic Euler should produce an oscillating curve. See Figure 2.1.

TODO: Implement symplectic Euler using the provided source file src/nemo/solvers/symplectic_euler.py.

Figure 2.1: *Explicit vs Symplectic Euler*. Sceenshots of the plots produced by Nemo when using correct implementation of Explicit Euler (left) and Symplectic Euler (right) integrators.

2.2.3 Midpoint Method

Explicit Euler can also be viewed from Taylor expansion perspective. The Tayler expansion of a state vector $\mathbf{y}$ is as follows:

$$\mathbf{y}(t_0 + h) = \mathbf{y}(t_0) + h\frac{d\mathbf{y}(t_0)}{dt} + \frac{h^2}{2}\frac{d^2\mathbf{y}(t_0)}{dt^2} + O(h^3). \tag{2.1}$$

where $\mathbf{y} = \begin{bmatrix}\mathbf{q}\\ \dot{\mathbf{q}}\end{bmatrix}$ is the state vector. Since

$$\frac{d\mathbf{y}(t_0)}{dt} = \begin{bmatrix}\dot{\mathbf{q}}(t_0)\\ \frac{1}{m}\mathbf{f}(\mathbf{q}(t_0))\end{bmatrix}, \tag{2.2}$$

it is clear that Explict Euler has an accuracy up to the first order of the Taylor expansion (2.1) (i.e., the second-order term and above are ignored by the Explicit Euler integration scheme).

In order to design a second-order integrator (i.e., one that takes into account up to the second-order term), we need to somehow compute $\frac{d^2\mathbf{y}(t_0)}{dt^2}$. For the sake of presentation, let's use $\mathbf{F}(\mathbf{y}_0)$ to denote the right-hand side of (2.2), i.e.,

$$\mathbf{F}(\mathbf{y}_0) := \begin{bmatrix}\dot{\mathbf{q}}(t_0)\\ \frac{1}{m}\mathbf{f}(\mathbf{q}(t_0))\end{bmatrix}. \tag{2.3}$$

Staring from $\dot{\mathbf{y}}(t_0) = \mathbf{F}(\mathbf{y}_0)$, the chain rule gives

$$\ddot{\mathbf{y}}(t_0) = \frac{\mathbf{F}(\mathbf{y}_0)}{dt} = \frac{\partial \mathbf{F}}{\partial \mathbf{y}}\dot{\mathbf{y}} = \mathbf{F}'(\mathbf{y}_0)\mathbf{F}(\mathbf{y}_0). \tag{2.4}$$

Now, let's perform another Taylor expansion, this time of the function of $\mathbf{F}$:

$$\mathbf{F}(\mathbf{y}_0 + \Delta\mathbf{y}) = \mathbf{F}(\mathbf{y}_0) + \Delta\mathbf{y}\mathbf{F}'(\mathbf{y}_0) + O(\Delta\mathbf{y}^2).$$

Choose $\Delta\mathbf{y} = h/2\mathbf{F}(\mathbf{y}_0)$, and substitute it into the above Taylor expansion, we have

$$\mathbf{F}(\mathbf{y}_0 + \frac{h}{2}\mathbf{F}(\mathbf{y}_0)) = \mathbf{F}(\mathbf{y}_0) + \frac{h}{2}\mathbf{F}(\mathbf{y}_0)\mathbf{F}'(\mathbf{y}_0) + O(\Delta\mathbf{y}^2) = \mathbf{F}(\mathbf{y}_0) + \frac{h}{2}\ddot{\mathbf{y}}(t_0) + O(h^2). \tag{2.5}$$

Multiply both sides of (2.5), we have

$$h\mathbf{F}(\mathbf{y}_0 + \frac{h}{2}\mathbf{F}(\mathbf{y}_0)) = h\mathbf{F}(\mathbf{y}_0) + \frac{h^2}{2}\ddot{\mathbf{y}}(t_0) + O(h^3). \tag{2.6}$$

If we Further add both sides by $\mathbf{y}(t_0)$, we have

$$\mathbf{y}(t_0+h) = \mathbf{y}(t_0) + h\mathbf{F}(\mathbf{y}_0) + \frac{h^2}{2}\ddot{\mathbf{y}}(t_0) + O(h^3) = \mathbf{y}(t_0) + h\mathbf{F}(\mathbf{y}_0 + \frac{h}{2}\mathbf{F}(\mathbf{y}_0)). \tag{2.7}$$

This last equality suggests a 2nd-order accurate numerical integration scheme, namely, the *midpoint method*:

$$\mathbf{y}^{n+1} = \mathbf{y}^n + h\mathbf{F}(\mathbf{y}^n + \frac{h}{2}\mathbf{F}(\mathbf{y}^n)). \tag{2.8}$$

TODO: Implement Midpoint method using the provided source file src/nemo/solvers/midpoint.py.

Question: If you run your midpoint method on scene01.yml, does the simulation diverage (like Explicit Euler) or stay stable (like the Symlectic method)?

2.2.4 Constant Gravity

Recall from introductory physics that, sufficiently close to earth's surface, we can approximate gravity's effect as a constant acceleration $\mathbf{g}$ on all objects. Placing the 0 potential reference at the origin, this force corresponds to a potential energy of $U(\mathbf{x}) = -m\mathbf{g}\cdot\mathbf{x}$. Taking the gradient of this potential, the force is given by $\mathbf{F} = -\nabla U = m\mathbf{g}$.

Make sure your integrator takes into account the gravity force. In our code, the vertical direction can be configured through the constructor of ModelBuilder, so it is not necessearily the $Y-$direction. As a result, the gravity coefficient vector (e.g., $(0, 0, -9.81)$ if $Z-$ direction is set to be the vertical direction) is stored in model.gravity, and can be accessed in the step function by self.model.gravity.

2.2.5 Bonus: Creative Scenes

While we provide a set of testing scenes, we encourage students to create their own scenes or even extend the starter code with more interesting features (e.g., you can consider to extent src/nemo/sim/forces.py) to add other kinds of forces (such as wind force or magnetic force) and use them in your integrators.

This part is not *required*. But if you feel interested and motivated to do that, we will give you up to 5 bonus points (out of 100) based on the effort you need to devote to create the creative scenes and extensions.

When you submit, please include in the README.md file a list of scenes and features that you added and point to the files and commands we should run to see your effects.