Workshop: Projectile Motion

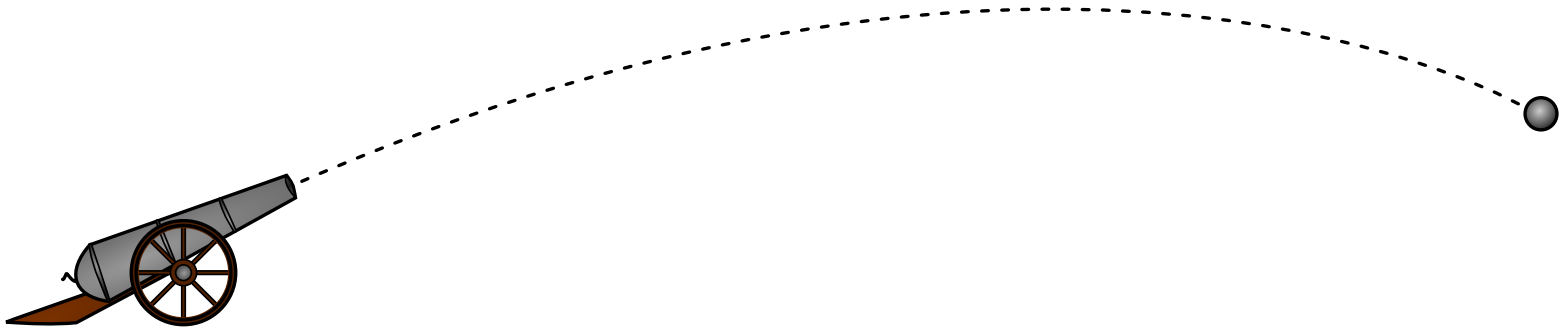
An introduction to computing trajectories

Jonas van den Brink
j.v.brink@fys.uio.no

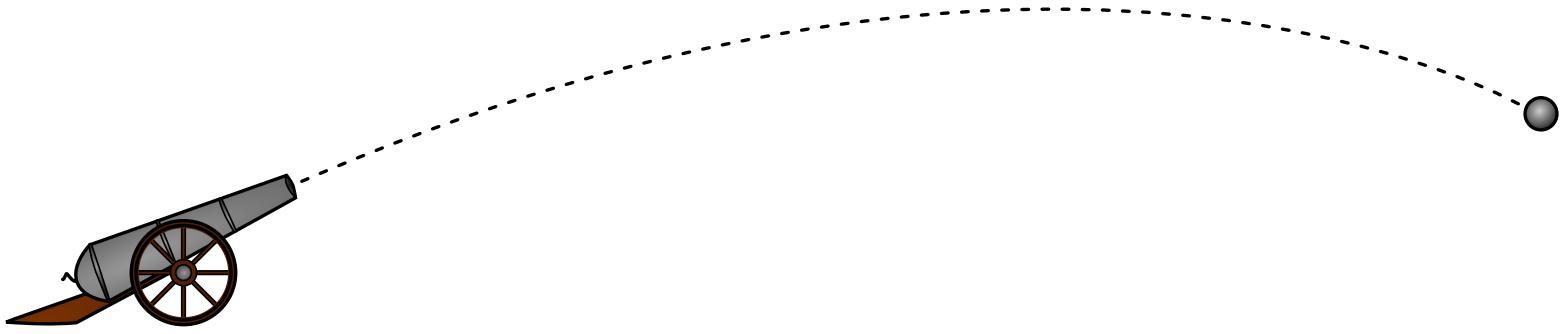
Simula Research Laboratory
Oslo, Norway

January 29, 2015

The goal is to find the velocity and position of an object as functions of time: $\vec{v}(t)$, $\vec{r}(t)$



The goal is to find the velocity and position of an object as functions of time: $\vec{v}(t)$, $\vec{r}(t)$



Equations of motion

$$\frac{\mathrm{d}\vec{r}}{\mathrm{d}t} = \vec{v}(t), \qquad \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \vec{a}(t)$$

The goal is to find the velocity and position of an object as functions of time: $\vec{v}(t)$, $\vec{r}(t)$

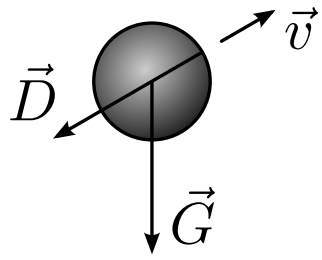
Newtons 2. law of motion

$$\vec{F} = m\vec{a}$$

The goal is to find the velocity and position of an object as functions of time: $\vec{v}(t)$, $\vec{r}(t)$

Newtons 2. law of motion

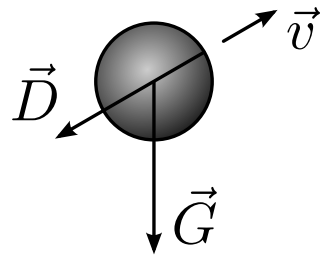
$$\vec{F} = m\vec{a}$$



The goal is to find the velocity and position of an object as functions of time: $\vec{v}(t)$, $\vec{r}(t)$

Newtons 2. law of motion

$$\vec{F} = m\vec{a}$$



$$\vec{F}(r, v, t) = m\vec{a}(r, v, t).$$

The goal is to find the velocity and position of an object as functions of time: $\vec{v}(t)$, $\vec{r}(t)$

Our algorithm is now as follows

1. Find the physical forces of the system.
2. Use Newtons 2. law to find the acceleration
3. Calculate $\vec{v}(t)$ and $\vec{r}(t)$ by solving the equations of motion

The goal is to find the velocity and position of an object as functions of time: $\vec{v}(t)$, $\vec{r}(t)$

Our algorithm is now as follows

1. Find the physical forces of the system.
2. Use Newtons 2. law to find the acceleration
3. Calculate $\vec{v}(t)$ and $\vec{r}(t)$ by solving the equations of motion

In this workshop, we will solve step number 3 numerically, using the Euler method.

The Euler Method

A method for solving ordinary differential equations (ODEs)

We can solve the equations of motion numerically using the Euler method

We can solve the equations of motion numerically using the Euler method

From the definition of the derivative

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{v(t + \Delta t) - v(t)}{\Delta t} = a(t)$$

We can solve the equations of motion numerically using the Euler method

From the definition of the derivative

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{v(t + \Delta t) - v(t)}{\Delta t} = a(t)$$

We now remove the limit, making Δt a very small constant

$$\frac{v(t + \Delta t) - v(t)}{\Delta t} \approx a(t)$$

We can solve the equations of motion numerically using the Euler method

From the definition of the derivative

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{v(t + \Delta t) - v(t)}{\Delta t} = a(t)$$

We now remove the limit, making Δt a very small constant

$$\frac{v(t + \Delta t) - v(t)}{\Delta t} \approx a(t)$$

Solving for $v(t + \Delta t)$ gives

$$v(t + \Delta t) \approx v(t) + a(t) \cdot \Delta t$$

We can solve the equations of motion by stepping forward in time

$$v(t + \Delta t) = v(t) + a(t) \cdot \Delta t$$

We can solve the equations of motion by stepping forward in time

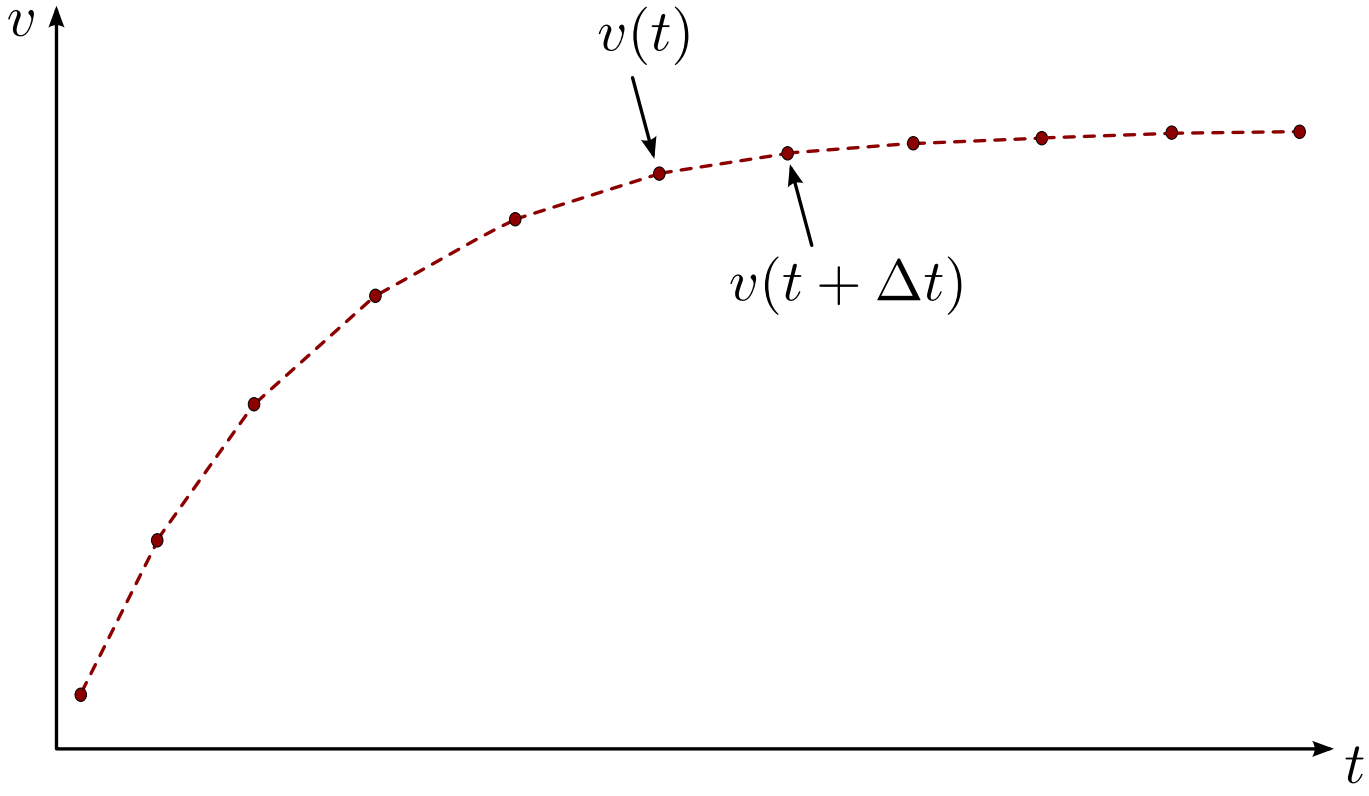
$$v(t + \Delta t) = v(t) + a(t) \cdot \Delta t$$

If $a(t)$ and $v(t)$ are known, we can calculate $v(t + \Delta t)$

We can solve the equations of motion by stepping forward in time

$$v(t + \Delta t) = v(t) + a(t) \cdot \Delta t$$

If $a(t)$ and $v(t)$ are known, we can calculate $v(t + \Delta t)$



Our functions are no longer continuous, they have become discretized

Our functions are no longer continuous, they have become discretized

We only focus on multiples of our time-step

$$t \in \{0,\ \Delta t,\ 2\Delta t,\ 3\Delta t, \ldots\}$$

$$t_i \equiv i \cdot \Delta t$$

Our functions are no longer continuous, they have become discretized

We only focus on multiples of our time-step

$$t \in \{0,\ \Delta t,\ 2\Delta t,\ 3\Delta t, \ldots\}$$

$$t_i \equiv i \cdot \Delta t$$

Introduce the shorthand

$$v(t_i) \equiv v_i$$

$$r(t_i) \equiv r_i$$

Our functions are no longer continuous, they have become discretized

We only focus on multiples of our time-step

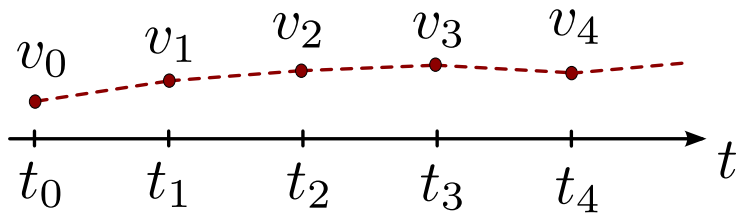
$$t \in \{0,\ \Delta t,\ 2\Delta t,\ 3\Delta t, \ldots\}$$

$$t_i \equiv i \cdot \Delta t$$

Introduce the shorthand

$$v(t_i) \equiv v_i$$

$$r(t_i) \equiv r_i$$



We solve the equations of motion iteratively

$$v_{i+1} = v_i + a_i \cdot \Delta t$$

We solve the equations of motion iteratively

$$v_{i+1} = v_i + a_i \cdot \Delta t$$

$$r_{i+1} = r_i + v_i \cdot \Delta t$$

We solve the equations of motion iteratively

$$v_{i+1} = v_i + a_i \cdot \Delta t$$

$$r_{i+1} = r_i + v_i \cdot \Delta t$$

For each time step, we must calculate the acceleration

$$a_i = a(r_i, v_i, t_i).$$

We solve the equations of motion iteratively

$$v_{i+1} = v_i + a_i \cdot \Delta t$$

$$r_{i+1} = r_i + v_i \cdot \Delta t$$

For each time step, we must calculate the acceleration

$$a_i = a(r_i, v_i, t_i).$$

We repeat these steps, starting at our initial conditions v_0 and r_0, until we have reached our end-time t_N

$$i = 0, 1, 2, 3, \ldots, N.$$

Algorithm for the Euler method

for $i = 0, 1, 2, 3, \ldots, N-1$:

1. Use the previous results x_i and v_i to compute the acceleration: $a_i = F(x_i, v_i, t_i)/m$.
2. Compute the new velocity: $v_{i+1} = v_i + a_i \Delta t$.
3. Compute the new position: $r_{i+1} = r_i + v_i \Delta t$.

Implementation

Moving from physics and math to actual computer code

for $i = 0, 1, 2, 3, \ldots, N-1$:

1. Use the previous results x_i and v_i to compute the acceleration: $a_i = F(x_i, v_i, t_i)/m$.
2. Compute the new velocity: $v_{i+1} = v_i + a_i \Delta t$.
3. Compute the new position: $r_{i+1} = r_i + v_i \Delta t$.

for $i = 0, 1, 2, 3, \ldots, N-1$:

1. Use the previous results x_i and v_i to compute the acceleration: $a_i = F(x_i, v_i, t_i)/m$.
2. Compute the new velocity: $v_{i+1} = v_i + a_i \Delta t$.
3. Compute the new position: $r_{i+1} = r_i + v_i \Delta t$.

```
for i in range(N):
    a[i] = F(x[i], v[i], t[i])/m
    v[i+1] = v[i] + a[i]*dt
    r[i+1] = r[i] + v[i]*dt
```

for $i = 0, 1, 2, 3, \ldots, N-1$:

1. Use the previous results x_i and v_i to compute the acceleration: $a_i = F(x_i, v_i, t_i)/m$.
2. Compute the new velocity: $v_{i+1} = v_i + a_i \Delta t$.
3. Compute the new position: $r_{i+1} = r_i + v_i \Delta t$.

$\Downarrow$

```
for i in range(N):
    a[i] = F(x[i], v[i], t[i])/m
    v[i+1] = v[i] + a[i]*dt
    r[i+1] = r[i] + v[i]*dt
```

We want the code to look as much as possible like the physics and math we write on paper

$t_i \Rightarrow$ `t[i]` $\quad$ $v_i \Rightarrow$ `v[i]` $\quad$ $r_i \Rightarrow$ `r[i]`

We also need various bookeeping code

Here we define the arrays we will be using

```
# Import various functions meant for numerical science
from pylab import *

t_0 = 0 # Start time, s
t_end = 10 # End time, s
N = 1000 # Number of time steps

# Create a uniformly spaced time-array
t = linspace(t_0, t_end, N+1)

# Calculate the size of a time step
dt = t[1] - t[0]

# Create empty acceleration, velocity and position arrays
a = zeros((2, N+1))
v = zeros((2, N+1))
r = zeros((2, N+1))

# Set initial conditions
v[0] = (10,0) # inital velocity, m/s
r[0] = (0,1)  # initial position, m
```

We also need various bookeeping code

Here we define physical constants for our system and define the function that describes the forces

```
m = 5.5 # mass, kg
g = 9.81 # acceleration of gravity, m/s^2
rho = 1.3 # air density, kg/m^3
C_D = 0.45 # drag coefficient
d = 0.11 # diameter of cannonball, m
A = pi*d**2 # cross-sectional area, m^2

def F(x, v, t):
    return -(0, m*g) - 0.5*rho*C_D*A*abs(v)*v
```

This example show the forces acting on the cannonball as it sails through the air

$$F(x, v, t) = F_g + F_d(\vec{v}) = -mg\vec{k} - \frac{1}{2}\rho C_D A|\vec{v}|\vec{v}$$

Plotting

Example

Results

- Can plot numerical result vs known solution
- Can plot with and without drag
- Can plot for various inital angles

Examples of possible projects

- Catapults, cannons, throwing of objects. Can include analytical and experimental data.
- Skydiver and plot the g-forces, can illustrate why a parachute needs to open slowly!
- Bungee-jumping! Great for teaching spring forces and programming technicalities.
- Block and spring, harmonic Oscillator
- Orbits of planets in the solar system with real data from NASA
- Pendulum (also with large angles!)