# CIT2213 Game Engine Architecture

## Lecture: Numerical Integration

Dr Minsi Chen

Informatics

# Overview

- Numerical methods for integration
- Euler's method
- Midpoint method
- Runge-Kutta's method

# Basic Physics Calculation

- We want to know the position of objects at time $t$, i.e.
  - A function of time $r(t) = ?$
  - This function is often unknown.
- Instead, we are commonly given
  - Force ($F$)
  - Mass ($m$)
  - Acceleration ($a = \frac{F}{m}$)
  - Initial velocity and position

# Differential Equations

- Acceleration is the second derivative of $r(t)$ or the first derivative of $v(t)$, i.e.
  - $a(t) = \frac{d^2r}{dt^2} = \frac{F}{m}$
  - $a(t) = \frac{dv}{dt}$
- Likewise velocity is the first derivative of $r(t)$, i.e.
  - $v(t) = \frac{dr}{dt}$
- The function $r(t)$ is embedded in acceleration and velocity but in its derivative form; these are known as differential equations.
- They are also frequently encountered outside of physics.

# Solving Differential Equations

- The solution to a differential equation is obtained by integration.
- In physics, our objective is to recover the function $r(t)$
- It can be done either:

Analytically (rare): An exact solution can be found if the anti-derivative can be found. Used when an object travels at a constant velocity or the velocity function is trivial.

Numerically (common): In general, we do not know the function describing the change of force and velocity.

# Taylor Series

- Suppose $r(t)$ is known, i.e. we known the position at time $t$; the position at time $t + \Delta t$ can be written using Taylor expansion

$$r(t + \Delta t) = r(t) + \frac{dr}{dt}\Delta t + \frac{d^2 r}{dt^2}\frac{\Delta t^2}{2!} + ... + \frac{d^n r}{dt^n}\frac{\Delta t^n}{n!}$$

- The more derivatives we know the closer we can approximate the original function.
- For a typical physics simulation, we usually only known the initial position $r(t)$, velocity and acceleration, i.e. the first and second derivative

# Euler's Method - I

- A.K.A first order approximation as it only relies on the first derivative
- Use the first two terms from the Taylor series
  - $r(t + \Delta t) \approx r(t) + \frac{dr}{dt}\Delta t = r(t) + v(t)\Delta t$
  - $v(t + \Delta t) \approx v(t) + \frac{d^2 r}{dt^2}\Delta t = v(t) + a(t)\Delta t$

```
void update_euler_method ( dt )
{
    acceleration = force*inverted_mass;

    new_velocity = old_velocity + acceleration*dt;

    new_position = old_position + new_velocity*dt;

    old_velocity = new_velocity;
    old_position = new_position;
}
```

# Euler's Method - II

- To numerically solve a differential equation we need to prescribe
  - Initial values, e.g. initial position and velocity
  - Step size, e.g. $\Delta t$
  - The differential equation, e.g. velocity
- Evaluate the function using a discrete step $\Delta t$
- Assume nothing changes within $\Delta t$
- Ignoring the remaining terms in Taylor expansion leads to numerical error in the order of $O(\Delta t)$

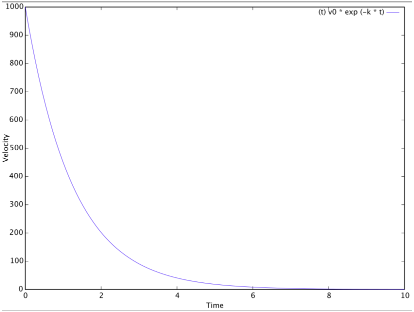
# Drag Force Example

- Consider a form of drag force affecting the velocity $v$

$$F_{drag} = -k_d v$$

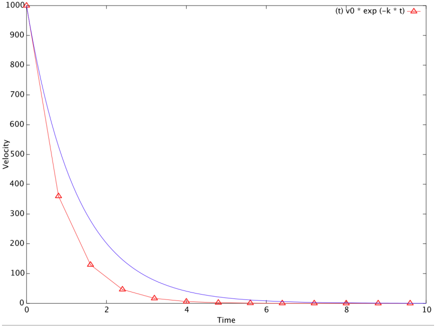
- The analytical solution is

$$v = exp(-k_d t)$$

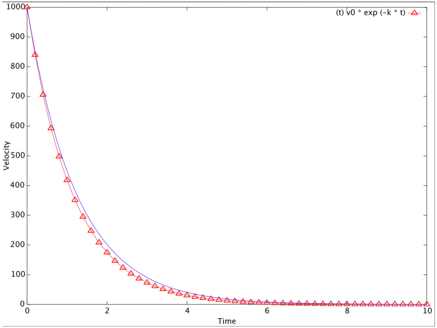



The plot of $v = exp(-k_d t)$, $k_d = 0.8$

Using Euler's method we can approximate the exact solution



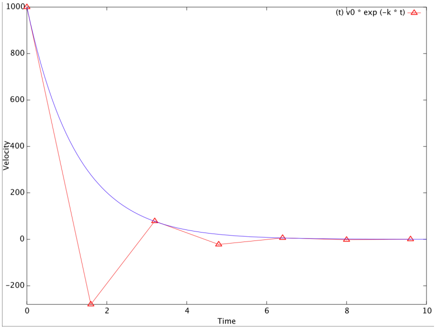

$\Delta t = 0.8$



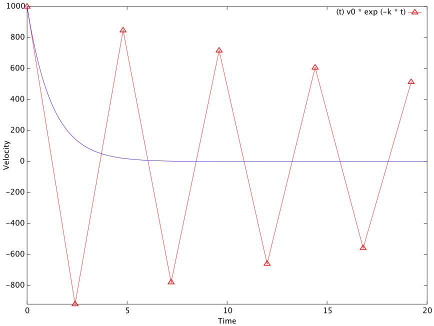

$\Delta t = 0.2$

# Drag Force Example

Things can go wrong if the step is not well chosen




$\Delta t = 1.6$ the bad




$\Delta t = 2.4$ the ugly

# Time Step Consideration

- The finer the time step $\Delta t$ is, the closer we can approximate the function
- The classic solution to reduce $\Delta t$ as much as we can, e.g.
  - $\Delta t$ can be frame rate dependent, a constant 60fps gives us around 16 ms
  - What happens when the frame rate is low or varies?
    - More integrations are needed for each frame
    - Each integration use a time step much smaller than the frame time
- We can also increase the number of terms used in the approximation

# Using Frame Independent $\Delta t$

- A common technique for increasing physics stability is to use a fixed $\Delta t$ independent from the frame time
- We can use a finer step at the cost of increasing the number of iterations

```
//Suppose we want to do 1000hz physics
fixed_dt = 0.001;

void render_frame ( frame_dt )
{
    for ( dt = 0.0; dt < frame_dt; dt += fixed_dt )
        update_physics ( fixed_dt );

    update_frame( frame_dt );
}
```

# Midpoint Method

- First evaluate the velocity at $\Delta t/2$

$$v(t + \frac{\Delta t}{2}) = v(t) + a\frac{\Delta t}{2}$$

- Then $r(t + \Delta t)$ becomes

$$r(t + \Delta t) = r(t) + v(t + \frac{\Delta t}{2})\Delta t$$

- Error becomes $O(\Delta t^2)$ since the first derivative is a quadratic function
- $O(\Delta t^2) < O(\Delta t)$ for $\Delta t < 0$

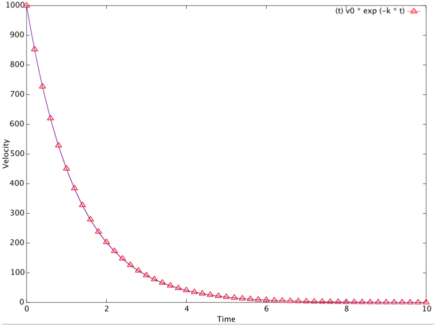
# Midpoint Method

For updating position using the Midpoint Method

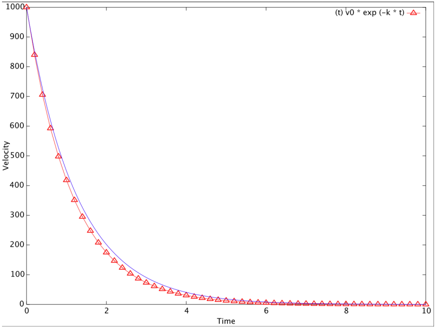
```
void update_midpoint_method ( dt )
{
    acceleration = force*inverted_mass;

    mid_velocity = old_velocity + acceleration*0.5*dt;

    new_velocity = mid_velocity*dt;

    new_position = old_position + new_velocity*dt;

    old_velocity = new_velocity;
    old_position = new_position;
}
```

# Drag Force Example - Midpoint

Using the Midpoint method:



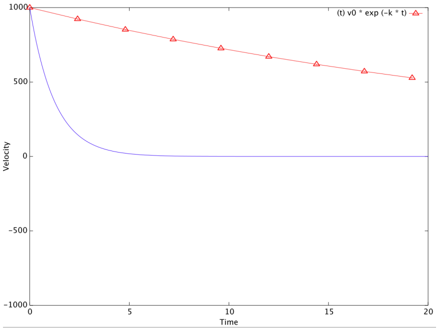

$\Delta t = 0.2$ using the Midpoint method



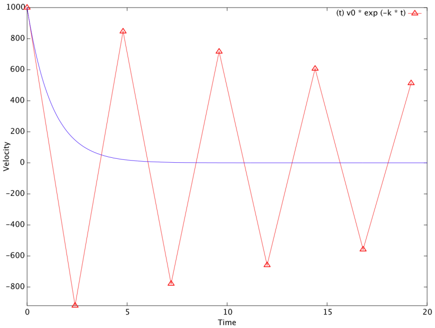

$\Delta t = 0.2$ using Euler's method

# Drag Force Example - Midpoint

For large step, the Midpoint method is inaccurate but more stable




$\Delta t = 2.4$ using the Midpoint method




$\Delta t = 2.4$ using Euler's method

We can do better than this.

# Runge-Kutta (RK) Method

- The Midpoint method is also known as the RK2 method
- The 4th order RK is a popular choice for numerical integration

$$\begin{aligned} k_1 &= v(t) + a\Delta t \\ k_2 &= v(t) + k_1 \frac{\Delta t}{2} \\ k_3 &= v(t) + k_2 \frac{\Delta t}{2} \\ k_4 &= v(t) + k_3 \Delta t \\ r(t+\Delta t) &= r(t) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\Delta t \end{aligned}$$

- The result is a weighted sum of a function evaluated at $\Delta t$, $\frac{\Delta t}{2}$ and $\frac{\Delta t}{4}$
- Error is $O(\Delta t^4)$

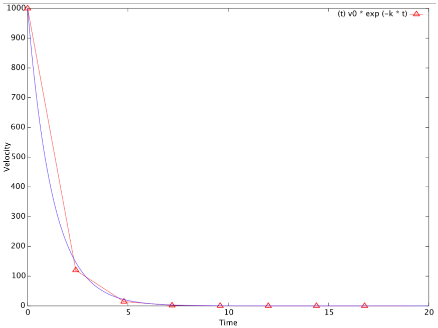
# RK4 Method

For updating position using the RK4 Method

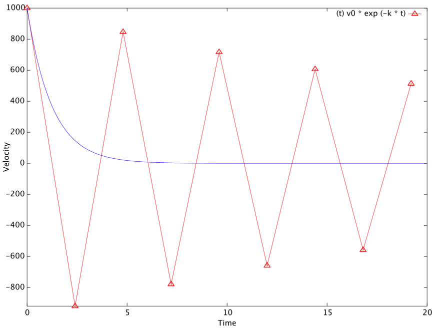
```
void update_rk4_method ( dt )
{
    acceleration = force*inverted_mass;

    k1 = old_velocity + acceleration*dt;

    k2 = old_velocity + k1*0.5*dt

    k3 = old_velocity + k2*0.5*dt;

    k4 = old_velocity + k3*dt;

    new_position = old_position + 1.0/6.0*(k1 + 2.0*k2 + 2.0*k3 + k4)*dt;

    old_position = new_position;
}
```

# Drag Force Example - RK4

Using RK4:




$\Delta t = 2.4$ using the RK4 method




$\Delta t = 2.4$ using Euler's method

Significantly better!

# Summary

- The choice of integration methods is dependent on:
  - Step size $\Delta t$
  - Computational demand
  - The behaviour of the underlying function
- For 2D and 3D simulations, each dimension can be integrated separately
- Other methods to explore
  - Verlet's method
  - Velocity Verlet
  - Improved Euler's method
- Chapter 13, Eberly, *Game Physics*