



Forces in One and Two Dimensions



#5

Serdar ARITAN

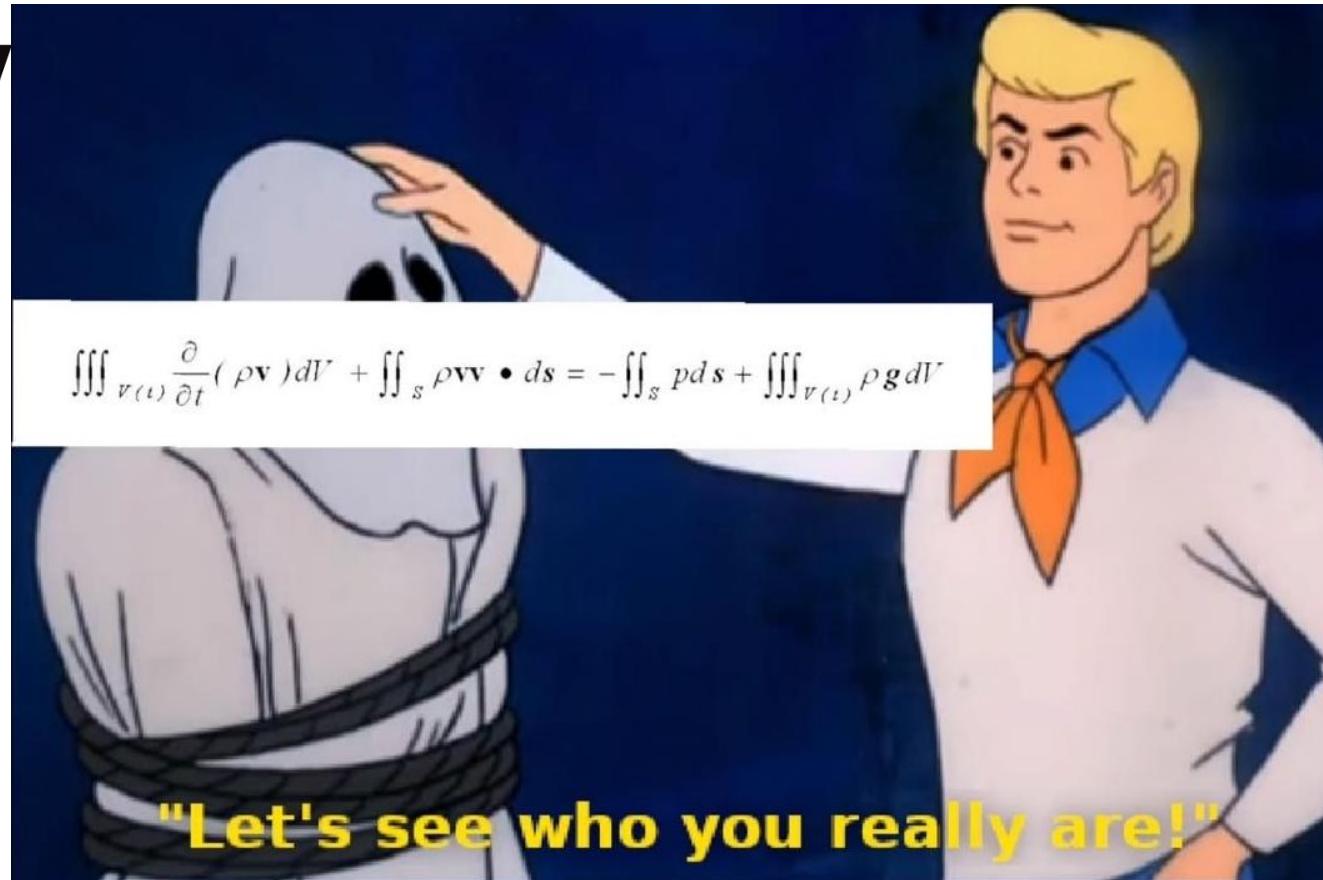
Biomechanics Research Group,
Faculty of Sports Sciences, and
Department of Computer Graphics
Hacettepe University, Ankara, Turkey



PHYSICS in COMPUTER ANIMATIONS and GAMES



UNREAL
ENGINE





PHYSICS in COMPUTER ANIMATIONS and GAMES





PHYSICS in COMPUTER ANIMATIONS and GAMES

The structured problem solving approach

Identify

What object is moving?

How is the position, $x(t)$, measured? (Origin and axes of coordinate system).

Find initial conditions, $x(t_0)$ and $v(t_0)$.

Model

Find the forces acting on the object.

Introduce models for the forces.

Apply Newton's second law of motion to find the acceleration, $a = a(x, v, t)$.

Solve

Solve the equation:

$$\frac{d^2x}{dt^2} = a(x, v, t),$$

with the initial conditions $x(t_0) = x_0$ and $v(t_0) = v_0$ using analytical or numerical techniques.

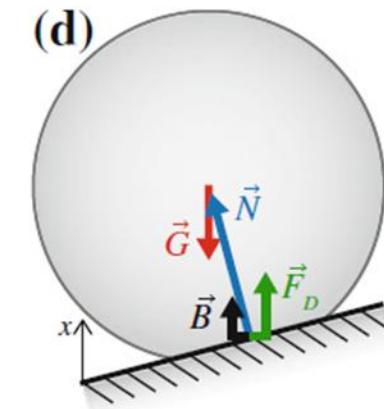
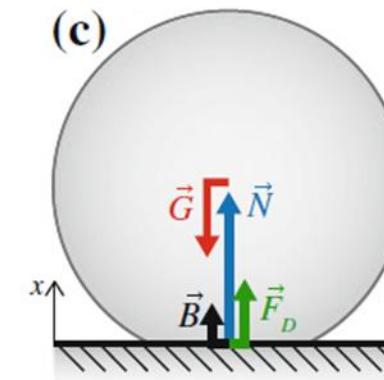
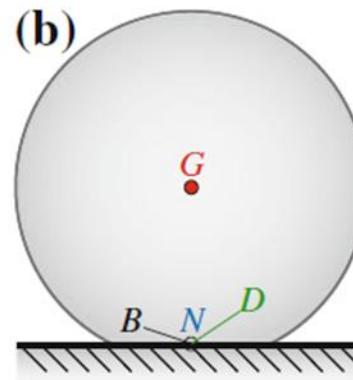
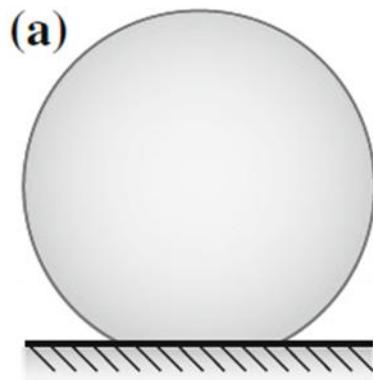
The solution gives the position and velocity as a function of time, $x(t)$, and $v(t)$.



PHYSICS in COMPUTER ANIMATIONS and GAMES

Identifying Forces

First, we need to discern between the object, also called the system, and the environment, which is everything else. In this case, the system is the ball, and the environment is everything else, such as the floor, the air surrounding the ball, and the Earth.

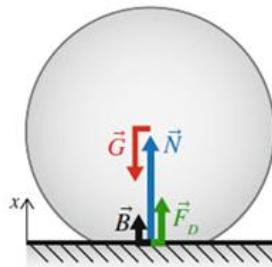




PHYSICS in COMPUTER ANIMATIONS and GAMES

Newton's Second Law of Motion

The force, F , in Newton's second law is the net external force acting on the object. By external we mean that the force has a cause outside the system, as we insisted when we drew a free-body diagram of an object. By net force we mean that if there are several forces acting on an object, it is the sum of all the external forces that causes the acceleration. We call this sum the net force:



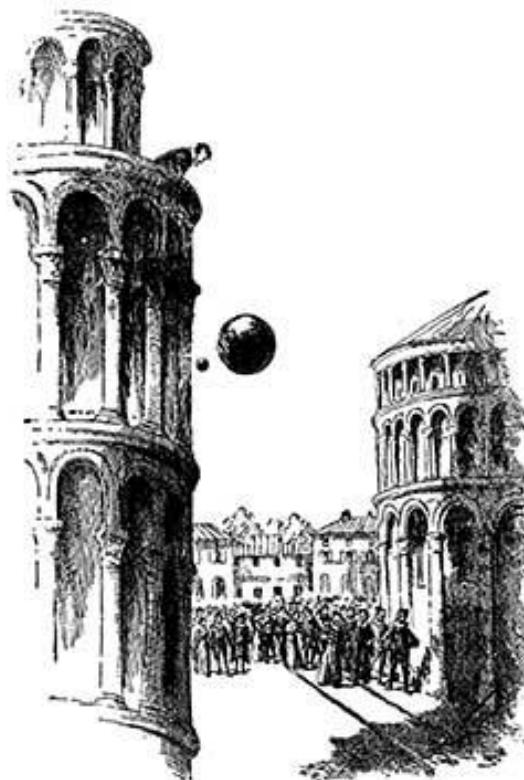
$$\vec{F}_{net} = \sum_j \vec{F}_j = m\vec{a}$$

$$\vec{F}_{net} = \sum_j \vec{F}_j = \vec{G} + \vec{N} + \vec{F}_D + \vec{B} = m\vec{a}$$



PHYSICS in COMPUTER ANIMATIONS and GAMES

Free Fall of an Object: An Experiment by Galileo



Galileo didn't know calculus (because Newton and Leibniz hadn't discovered it yet) so he couldn't derive the equation mathematically. Since we do know calculus we know that acceleration is the variation of velocity with time.

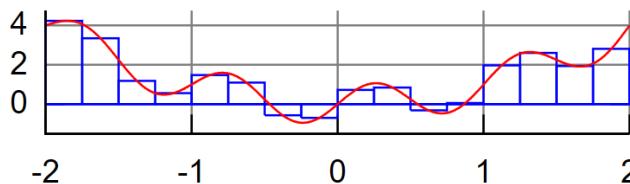
$$\vec{F}_{air\ drag} = \frac{1}{2} \rho A C |\vec{v}|^2 \hat{v}$$
$$\vec{F}_{net} = \sum_j \vec{F}_j = m \vec{a}.$$
$$\vec{F}_{gravity} = m \vec{g}.$$



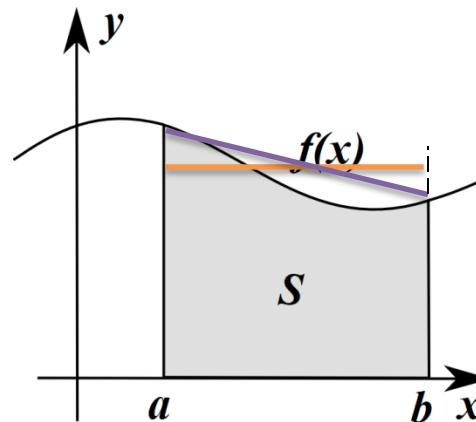
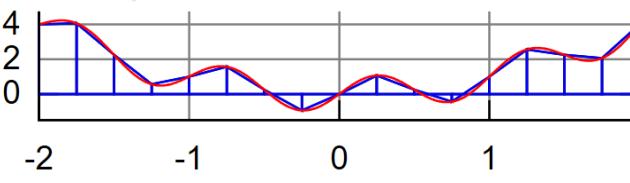
PHYSICS in COMPUTER ANIMATIONS and GAMES

Numerical integration

Rectangle rule



Trapezoidal rule



$$\int_a^b f(x) dx$$

$$\int_a^b f(x) dx \approx (b - a) f\left(\frac{a + b}{2}\right)$$

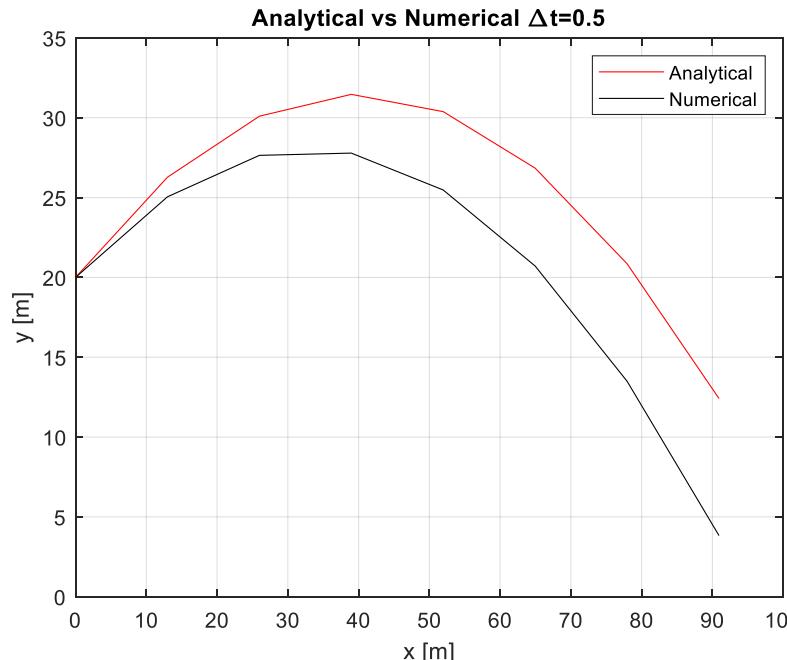
$$\int_a^b f(x) dx \approx (b - a) \frac{f(a) + f(b)}{2}$$



PHYSICS in COMPUTER ANIMATIONS and GAMES

Comparison of Analytical vs Numerical Methods

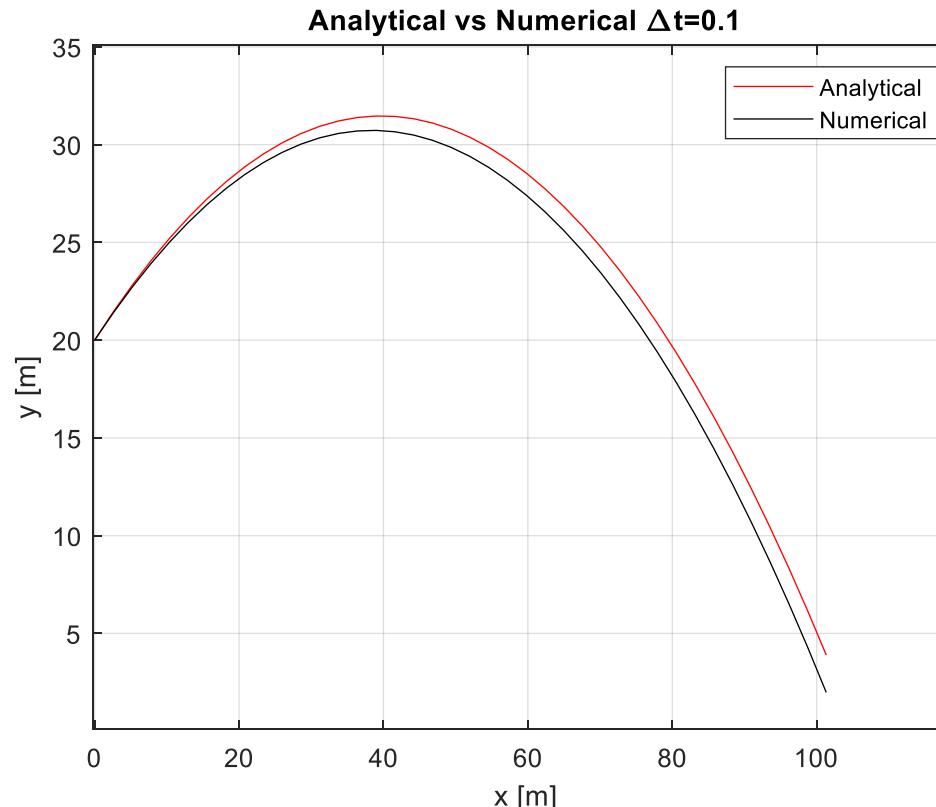
$$y_t = y_0 + v_0 \sin(\alpha)t - \frac{1}{2} \vec{g} t^2$$
$$x_t = x_0 + v_0 \cos(\alpha)t$$





PHYSICS in COMPUTER ANIMATIONS and GAMES

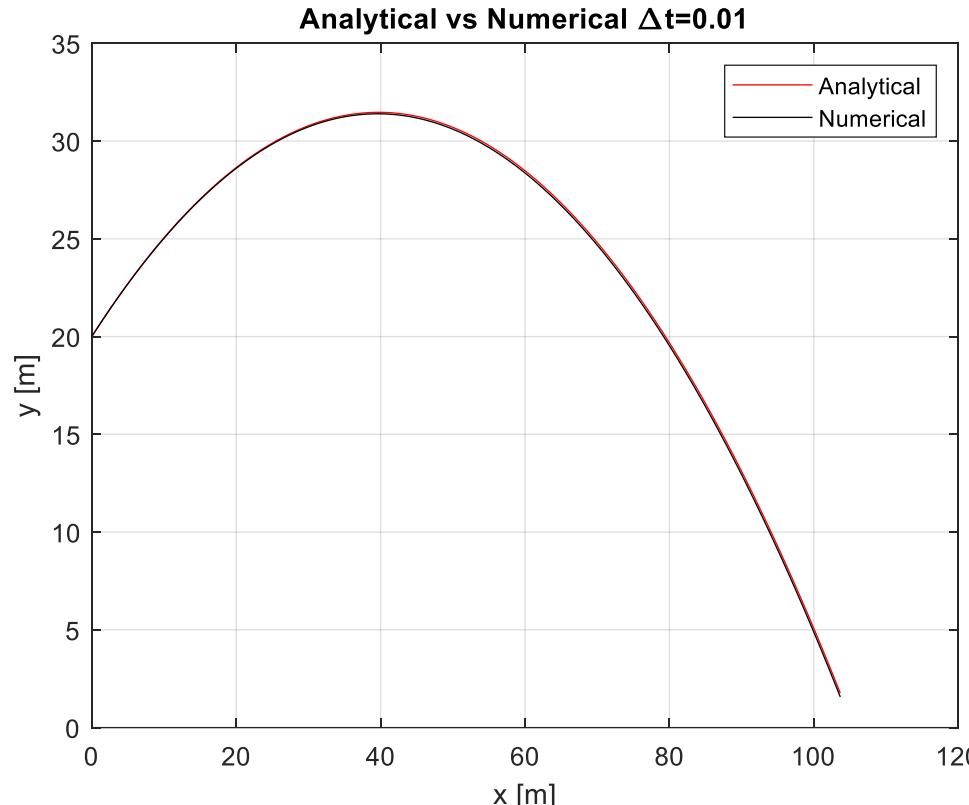
Comparison of Analytical vs Numerical Methods





PHYSICS in COMPUTER ANIMATIONS and GAMES

Comparison of Analytical vs Numerical Methods





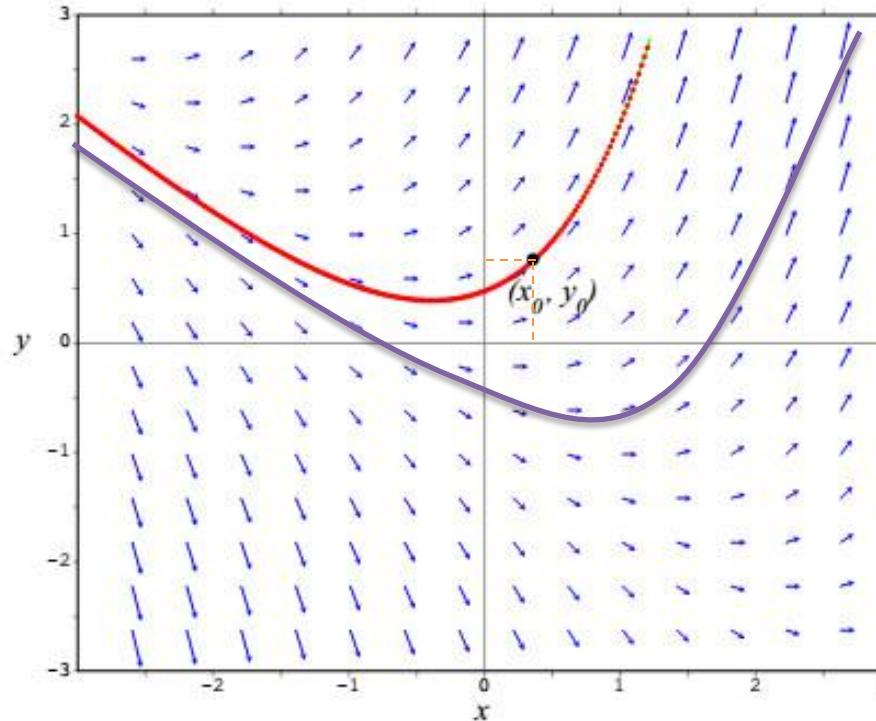
PHYSICS in COMPUTER ANIMATIONS and GAMES

The vector field plot for the differential equation

$$\frac{dy}{dx} = y + x$$

$$x_0 = 0.4$$

$$y_0 = 0.8$$

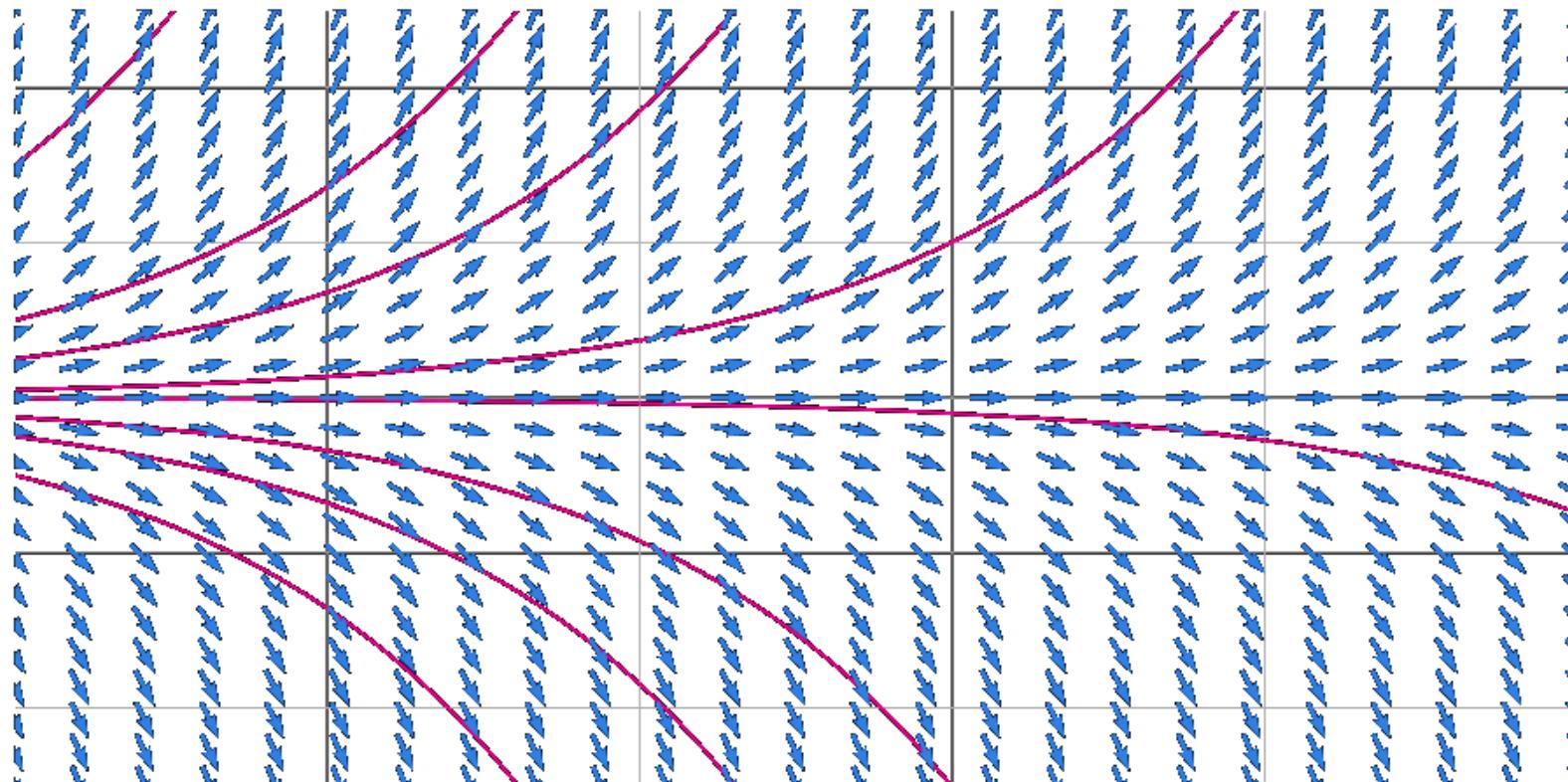


Initial-Value Problem :
An IVP is a differential equation together with a place for a solution to start.



PHYSICS in COMPUTER ANIMATIONS and GAMES

There are infinitely many integral curves, each corresponding to an integral curve.



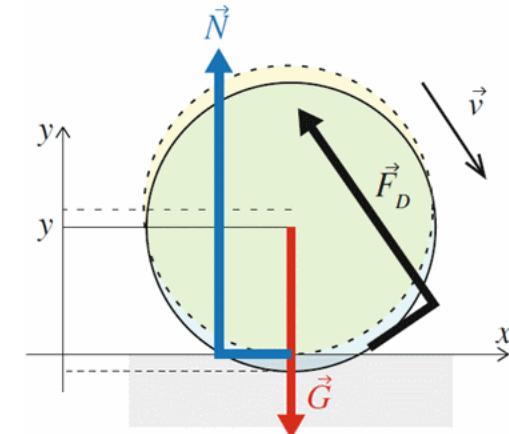


PHYSICS in COMPUTER ANIMATIONS and GAMES

Motion of a Bouncing Ball with Air Resistance

A ball is thrown from a height h above the ground with an initial velocity \vec{v}_0 . Find the velocity and position of the ball as a function of time t . Include the normal force from the floor while the ball is in contact with the floor. We describe the position of the ball by $\vec{r}(t)$, measured in a coordinate system with origin at the floor. The initial position and velocity of the projectile is $\vec{r}(t_0) = h \mathbf{j}$ and $\vec{v}(t_0) = v_{x_0} \mathbf{i} + v_{y_0} \mathbf{j}$.

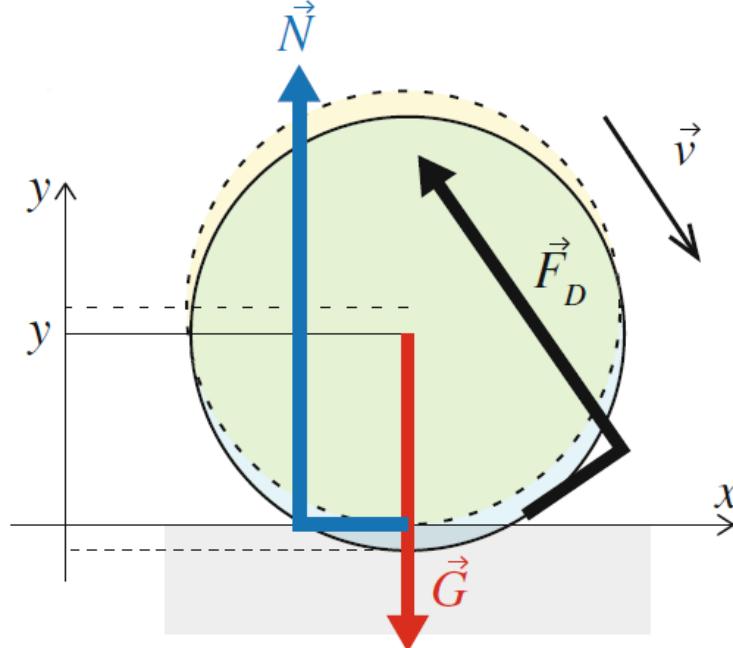
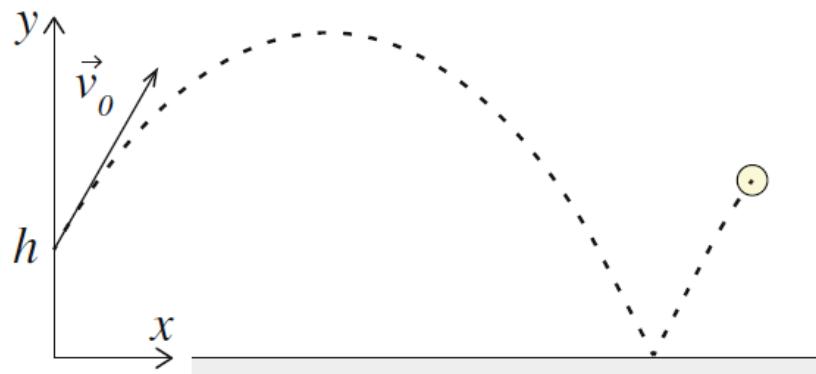
The motion of the ball is determined by the forces acting: air resistance, \vec{F}_D , the normal force \vec{N} from the floor, and gravity, $\vec{G} = -mg\mathbf{j}$, as illustrated in the free-body diagram. We use a square law for air resistance: $\vec{F}_D = -Dv^2$





PHYSICS in COMPUTER ANIMATIONS and GAMES

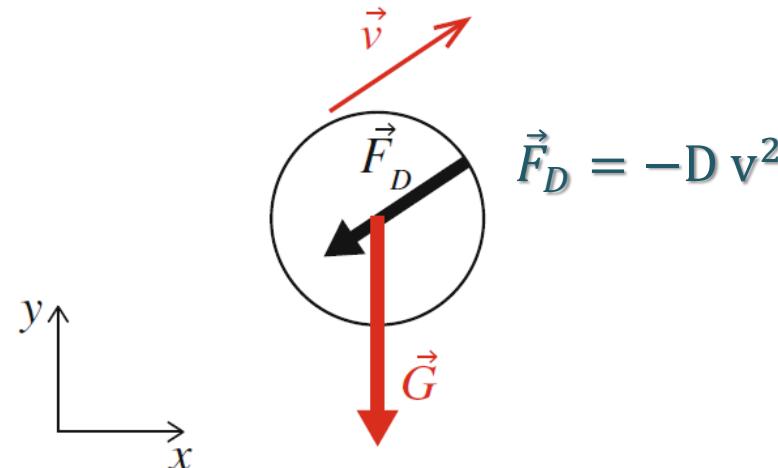
The normal force from the floor on the ball is represented by a spring force. This is a strong simplification of the actual deformation process occurring at the contact between the ball and the floor due to the deformation of both the ball and the floor.





PHYSICS in COMPUTER ANIMATIONS and GAMES

Since the projectile will be moving fast, we use the square-law force model for the air resistance. For a **spherical object** we have that the **pre-factor** is $D \cong 3.0\rho d^2$, where d is the diameter of the sphere and ρ is the density of the surrounding air. At sea level and at 15 °C air has a density of approximately 1.225 kg/m³. Here, we will assume that the density of the surrounding air does not change significantly. We will use $\rho = 1.225 \text{ kg/m}^3$. Let us also assume that the projectile has a diameter of $d = 0.02 \text{ m}$, and that its mass is $m = 0.2 \text{ kg}$.





PHYSICS in COMPUTER ANIMATIONS and GAMES

Values of Air Density As a Function of Altitude

Altitude (m)	Altitude (ft)	Density (kg/m ³)
0.0	0.0	1.225
305	1000	1.189
610	2000	1.154
914	3000	1.121
1219	4000	1.088
1524	5000	1.055
2134	7000	0.992
3048	10,000	0.905

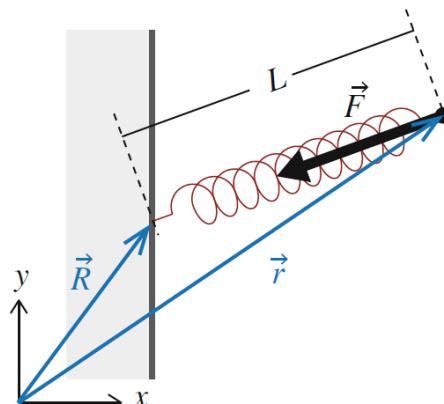


PHYSICS in COMPUTER ANIMATIONS and GAMES

From one-dimensional experiments, we expect the force from a spring on the object attached to the spring to depend on the elongation of the spring and act in the direction of the spring. A spring is characterized by its equilibrium length, L_0 , and its spring constant, k . The force from the spring on the object is:

$$\vec{F} = -k(L - L_0)\hat{u}_r$$

where **L** is the length of the spring, and the unit vector \hat{u}_r points from the spring towards the object.



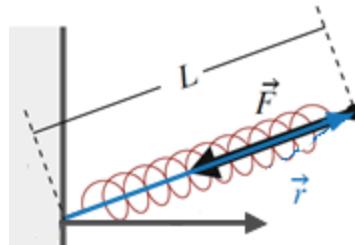
$$L = |\vec{r} - \vec{R}|$$
$$\vec{r} - \vec{R}$$
$$\hat{u}_r = \frac{\vec{r} - \vec{R}}{|\vec{r} - \vec{R}|}$$

A diagram showing a 2D coordinate system with x and y axes. A vector \vec{r} originates from the origin and points to a point r . A vector \vec{R} originates from the origin and points to a point R . The vector difference $\vec{r} - \vec{R}$ is shown. A unit vector \hat{u}_r is drawn from the tip of \vec{R} in the direction of \vec{r} . The angle between \vec{r} and \hat{u}_r is labeled \hat{u}_r .



PHYSICS in COMPUTER ANIMATIONS and GAMES

Often, we will place the origin at the attachment point of the spring, so that $R = 0$, and the full model simplifies to:



$$\vec{F} = -k(r - L_0) \frac{\vec{r}}{r}$$

where the length of the spring, $L = r = |\vec{r}|$, is the distance from the origin to the particle. We have named this model the “full model” because it most closely represents the behavior of a real, physical spring. This force model is versatile and general and can be widely applied. For example, it can be used to model the deformation of an elastic body. This model will be our preferred model for contact forces such as forces due to deformation in two- and three-dimensional systems.



PHYSICS in COMPUTER ANIMATIONS and GAMES

Notice that the force model has a spherical symmetry: When we choose the origin at the attachment point ($R = 0$), the force from the spring on the attached object always acts along a line through the origin, and the magnitude of the force depends on the distance r to the origin. This means that force on the particle from the spring in the x-direction is:

$$\vec{F}_x = -k(r - L_0) \frac{x}{r} = -k\left(\sqrt{x^2 + y^2} - L_0\right) \frac{x}{\sqrt{x^2 + y^2}}$$

That is, the force in the x-direction, depends not only on the x-position, but also on the y coordinate. If we apply Newton's second law of motion to such a system, the acceleration of the object in the x-direction, will depend on the x, y, and z coordinates of the object: We call such a system **coupled**.



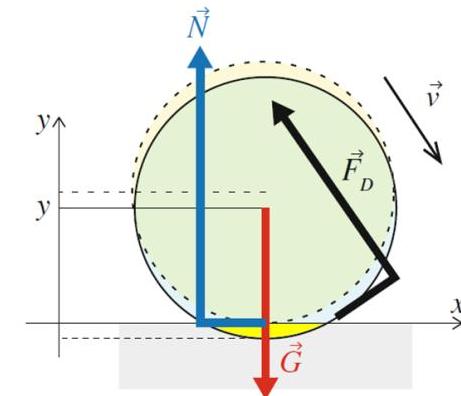
PHYSICS in COMPUTER ANIMATIONS and GAMES

The deformed region corresponds roughly to the region of “*overlap*” between the ball and the floor. The depth of this region is $\Delta y = R - y(t)$, where R is the radius of the ball, which corresponds to the compression ΔL of the spring:

$$\vec{N} = -k(R - y(t)) \mathbf{j}$$

we must also ensure that the normal force only acts when the ball is in contact with the floor, otherwise the normal force is zero.

$$\vec{N} = \begin{cases} -k(R - y(t)) \mathbf{j} & \text{when } y(t) < R \\ 0 & \text{when } y(t) \geq R \end{cases}$$





PHYSICS in COMPUTER ANIMATIONS and GAMES

Newton's second law: Newton's second law is now

$$\sum_j \vec{F}_j = \vec{G} + \vec{F}_D + \vec{N} = \vec{G} = -mg\mathbf{j} - D v^2 + \vec{N} = m\vec{a}$$

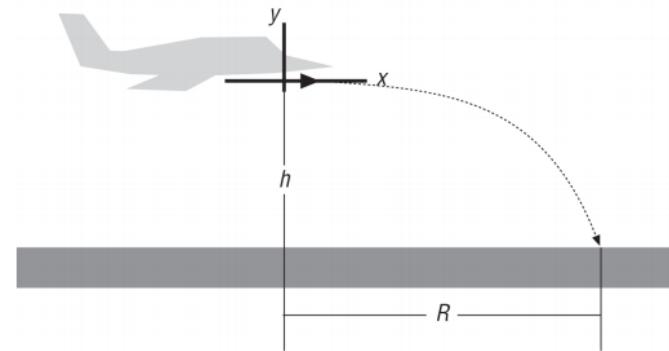
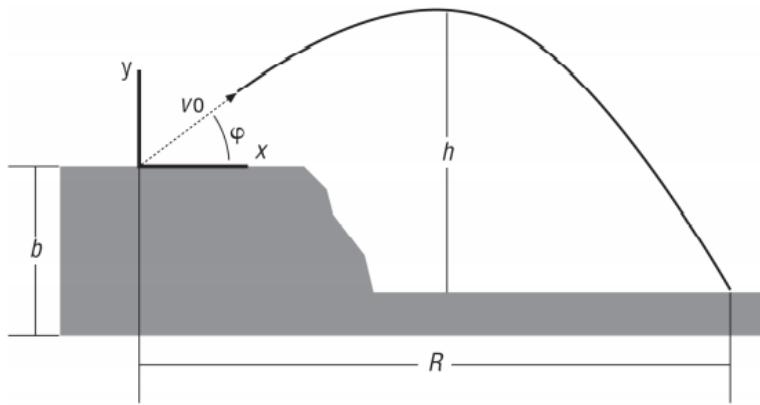
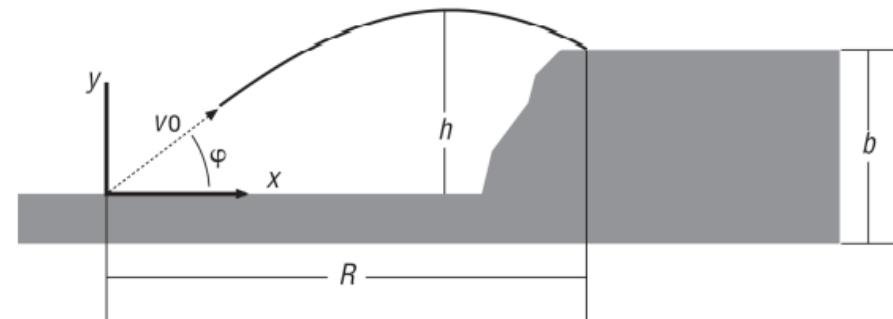
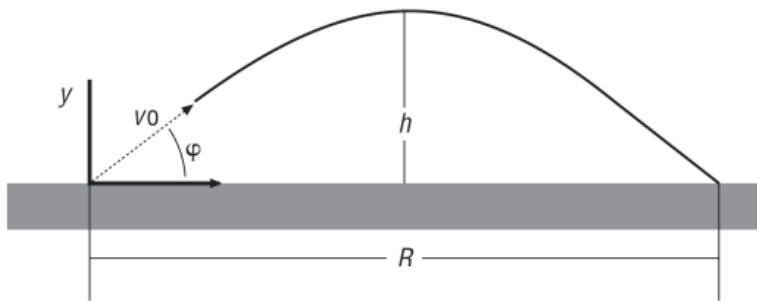
which gives

$$\vec{a} = -\left(\frac{D}{m}\right)v^2 - g\mathbf{j} + \vec{N}/m$$

with the initial conditions: $r(t_0) = r(0) = r_0$ and $v(t_0) = v(0) = v_0$. While it is difficult to determine the motion analytically, we may be able to find analytical solutions for parts of the motion. However, we can determine the motion numerically by integrating using Euler's method.



PHYSICS in COMPUTER ANIMATIONS and GAMES





PHYSICS in COMPUTER ANIMATIONS and GAMES

$$\mathbf{v}(t_i + \Delta t) \cong \mathbf{v}(t_i) + \Delta t \mathbf{a}(t_i, \mathbf{r}(t_i), \mathbf{v}(t_i))$$

$$\mathbf{r}(t_i + \Delta t) \cong \mathbf{r}(t_i) + \Delta t \mathbf{v}(t_i + \Delta t)$$

The implementation is straight-forward:

may be able to find analytical solutions for parts of the motion. However, we can determine the motion numerically by integrating using Euler's method.



PHYSICS in COMPUTER ANIMATIONS and GAMES

Normalization using NumPy norm

Normalization of a vector or a matrix is a common operation performed in a variety of scientific, mathematical, and programming applications. NumPy has a dedicated submodule called `linalg` for functions related to Linear Algebra.

```
import numpy as np  
  
a = np.array([1, 2, 3, 4, 5])  
  
a_norm = np.linalg.norm(a)  
  
print(a_norm)
```

$$\|a\|_2 = \sqrt{1^2 + 2^2 + 3^2 + 4^2 + 5^2} = 7.4162$$



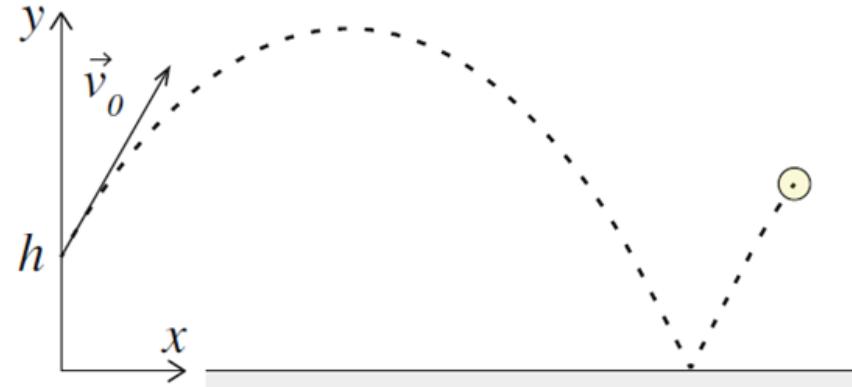
PHYSICS in COMPUTER ANIMATIONS and GAMES

```
import matplotlib.pyplot as plt
import numpy as np
# Physical variables
m = 0.2                      # mass of the ball in kg
g = 9.81                       # gravitational acceleration of the Earth in m/s^2
alpha = np.radians(75)           # Initial angle in radian
v0 = 30                         # Initial velocity in m/s
y0 = 0                           # Initial position in meters
diam = 0.02                      # diameter of the ball in meters
rho = 1.225                      # air density in kg/m^3
D = 3.0*rho*diam**2             # drag coefficient
R = (diam/2)                     # ball contact starts - equilibrium length of Spring
k = 1000.0                        # stiffness of the spring N/m
time = 10.0                       # simulation time
dt = 0.01                         # time step
```



PHYSICS in COMPUTER ANIMATIONS and GAMES

```
# Numerical initialization
n = int(np.ceil(time/dt))
a = np.zeros((n, 2),float)
v = np.zeros((n, 2),float)
r = np.zeros((n, 2),float)
t = np.zeros((n, 2),float)
# Set initial values
r[0,1] = y0                                     # initial position of the ball
v[0,:] = v0*np.cos(alpha), v0*np.sin(alpha)      # initial velocity in i, j
```

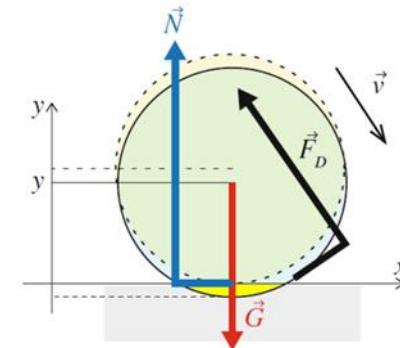




PHYSICS in COMPUTER ANIMATIONS and GAMES

```
# Integration loop
for i in range(n-1):
    if (r[i,1] < R):
        N = k*(R-r[i,1])*np.array([0,1])
    else:
        N = np.array([0,0])
FD = - D*np.linalg.norm(v[i,:])*v[i,:]
G = -m*g*np.array([0,1])
Fnet = N + FD + G
a = Fnet/m
v[i+1,:] = v[i] + a*dt
r[i+1,:] = r[i] + v[i+1]*dt
t[i+1] = t[i] + dt
```

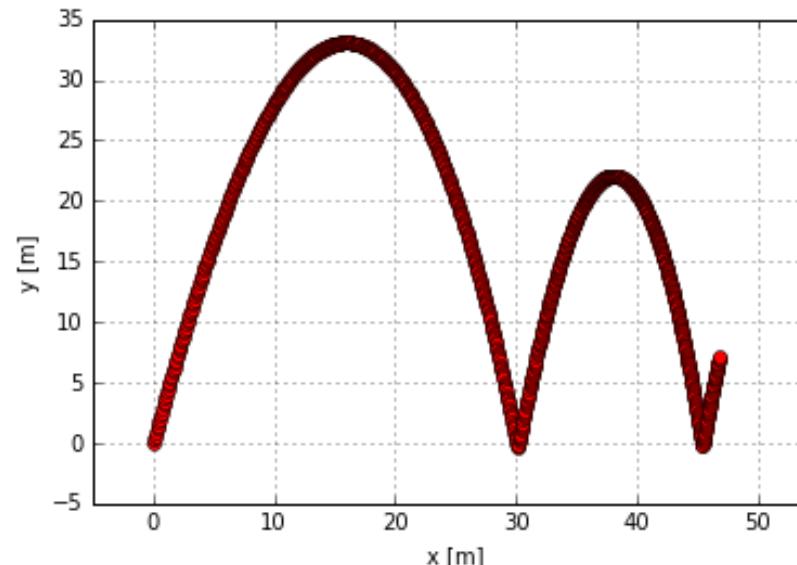
$$\vec{N} = \begin{cases} -k(R - y(t)) \mathbf{j} & \text{when } y(t) < R \\ 0 & \text{when } y(t) \geq R \end{cases}$$





PHYSICS in COMPUTER ANIMATIONS and GAMES

```
# Plotting the results,  
fig, ax = plt.subplots()  
ax.plot(r[:,0],r[:,1], 'ro')  
ax.axis([0, 50, 0, 50]), ax.axis('equal')  
ax.set_xlabel('x [m]'), ax.set_ylabel('y [m]'), ax.grid(True)
```





PHYSICS in COMPUTER ANIMATIONS and GAMES

Classworks

Use different values for;

mass – m [kg],

diameter – $diam$ [m],

air density – ρ [kg/m³],

gravity – g [m/s²]

and plot the results.



PHYSICS in COMPUTER ANIMATIONS and GAMES

Flock of Birds with the mass and the diameters



TERENCE



MATILDA



BOMB



HAL



CHUCK



RED



STELLA



THE
BLUES



BUBBLES

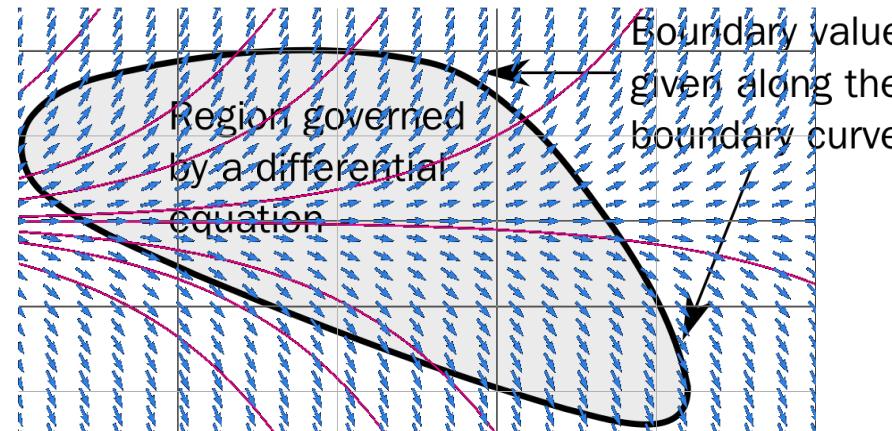
mass	12	8	7	5	5	4	4	2	1
size	Large	Large Medium	Large Medium	Medium	Medium	Medium	Medium Small	Small	Small



PHYSICS in COMPUTER ANIMATIONS and GAMES

Boundary Value Problems

In mathematics, in the field of differential equations, a boundary value problem is a differential equation together with a set of additional constraints, called the boundary conditions. A solution to a boundary value problem is a solution to the differential equation which also satisfies the boundary conditions.

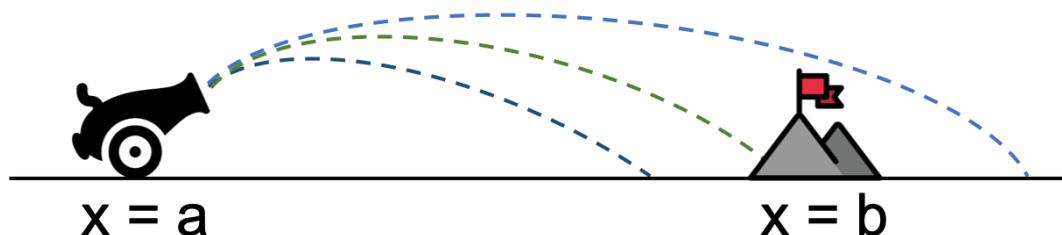




PHYSICS in COMPUTER ANIMATIONS and GAMES

Boundary Value Problems - The Shooting Methods

The shooting methods are developed with the goal of transforming the ODE boundary value problems to an equivalent **initial value problems (IVP)**, then we can solve it using the methods that we learned. In the IVP, we can start at **the initial value** and **march forward to get the solution**. But this method is not working for the boundary value problems, because there are not enough initial value conditions to solve the ODE to get a unique solution. Therefore, the shooting methods was developed to overcome this difficulty.





PHYSICS in COMPUTER ANIMATIONS and GAMES

The Shooting Methods

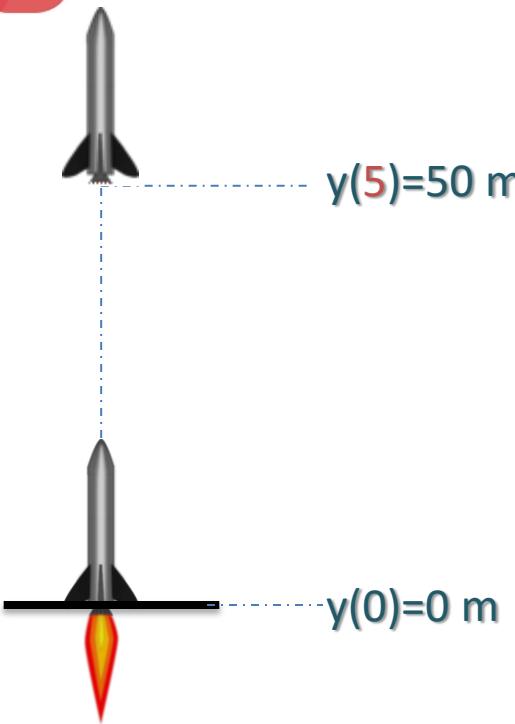
We are going out to launch a rocket, and let $y(t)$ is the altitude (meters from the surface) of the rocket at time t . We know the gravity $g = 9.8 \text{ m/s}^2$. If we want to have the rocket at 50 m off the ground after 5 seconds after launching, what should be the velocity at launching? (we ignore the drag of the air resistance). The problem is a boundary value problem for a second-order ODE. The ODE is:

$$\frac{d^2y}{dt^2} = -g$$

with the two boundary conditions are $y(0)=0$ and $y(5)=50$

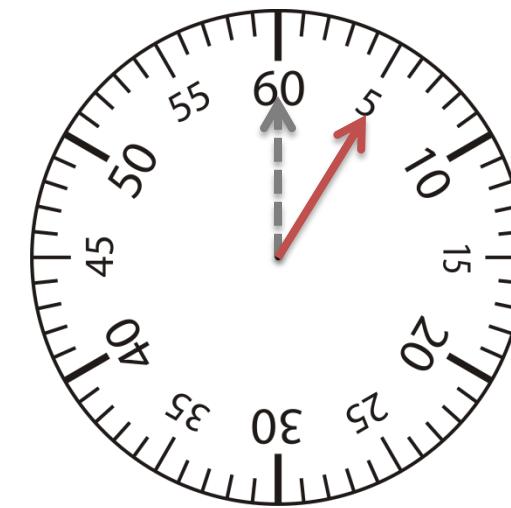


PHYSICS in COMPUTER ANIMATIONS and GAMES



Equation of Motion

$$\frac{d^2y}{dt^2} = -g$$

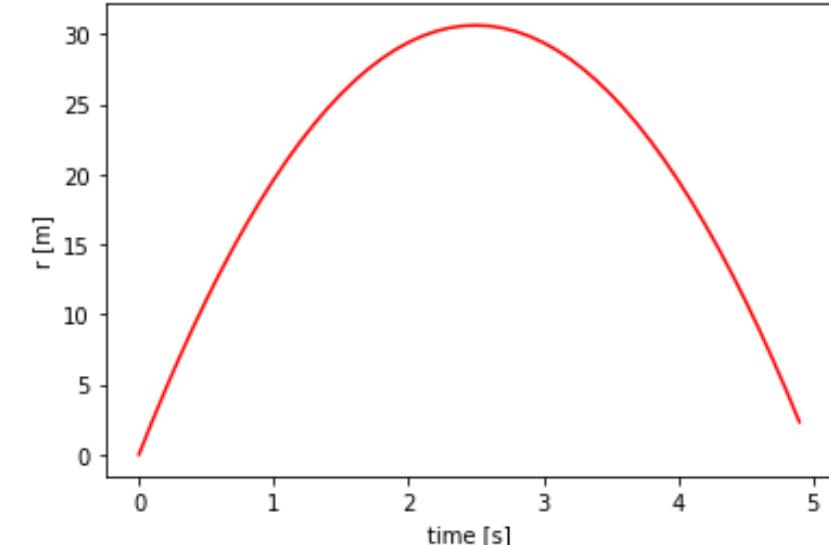


boundary conditions are: $y(0)=0$ and $y(5)=50$



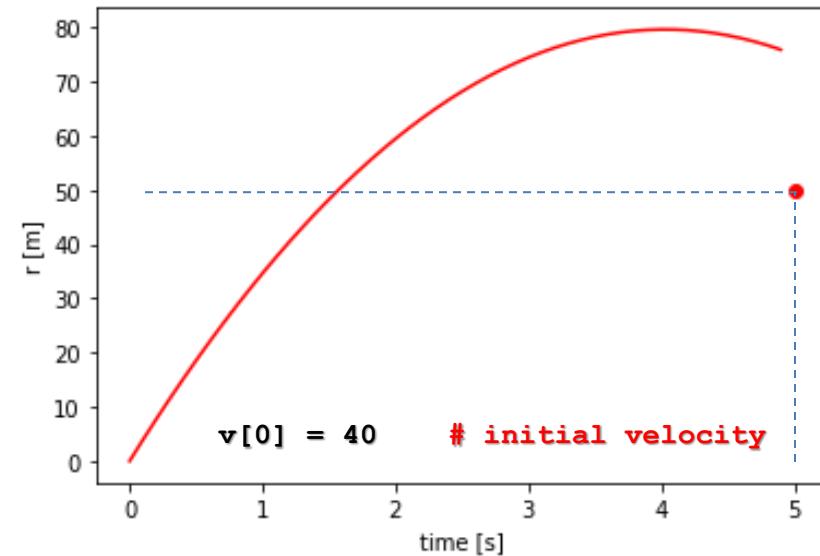
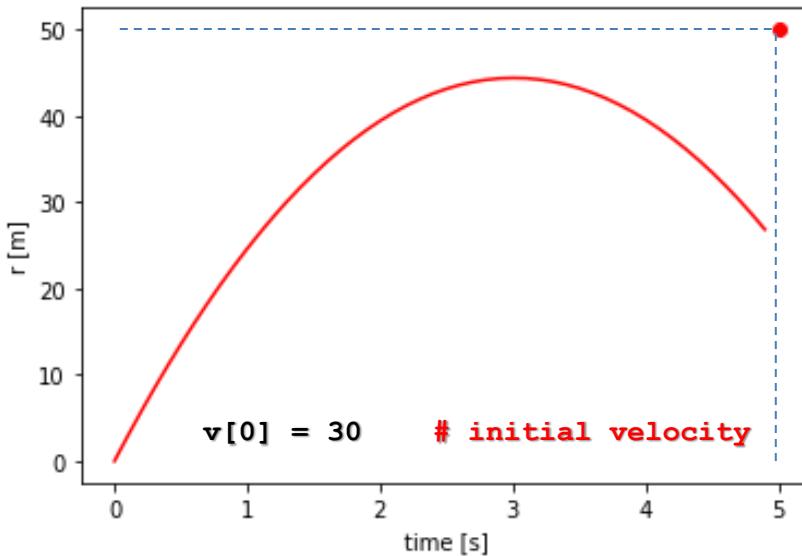
PHYSICS in COMPUTER ANIMATIONS and GAMES

```
import numpy as np
import matplotlib.pyplot as plt
# Physical variables
g = 9.81          # gravity
time = 5.0         # Simulation Time
dt = 0.1           # (h) time step
# Numerical initialization
n = int(np.ceil(time/dt))
a = np.zeros(n, float)
v = np.zeros(n, float)
r = np.zeros(n, float)
t = np.zeros(n, float)
# Set initial values
r[0] = 0           # meters
v[0] = 25          # initial velocity
a[:] = -g          # constant acceleration
# Integration loop
for i in range(n-1):
    v[i+1] = v[i] + a[i]*dt
    r[i+1] = r[i] + v[i+1]*dt
    t[i+1] = t[i] + dt
# Position Plotting
fig, ax = plt.subplots()
ax.plot(t, r, '-r')
ax.set_xlabel('time [s]')
ax.set_ylabel('r [m]')
plt.show()
```





PHYSICS in COMPUTER ANIMATIONS and GAMES





PHYSICS in COMPUTER ANIMATIONS and GAMES

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

F = lambda t, s: np.dot(np.array([[0,1],[0,-9.8/s[1]]]), s)

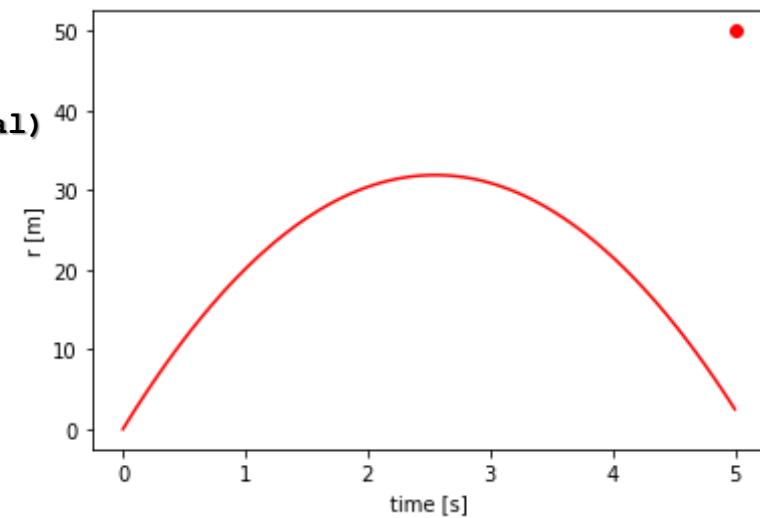
t_span = np.linspace(0, 5, 100)
y0 = 0
v0 = 25
t_eval = np.linspace(0, 5, 100)
sol = solve_ivp(F, [0, 5], [y0, v0], t_eval = t_eval)

# Position Plotting
fig, ax = plt.subplots()
ax.plot(sol.t, sol.y[0], '-r')
ax.plot(5, 50, 'ro')
ax.set_xlabel('time [s]')
ax.set_ylabel('r [m]')
plt.show()
```

scipy.integrate.solve_ivp

Solve an initial value problem for a system of ODEs.

This function numerically integrates a system of ordinary differential equations given an initial value





PHYSICS in COMPUTER ANIMATIONS and GAMES

```
import numpy as np
from scipy.integrate import solve_ivp
from scipy.optimize import fsolve

f = lambda t, s: np.dot(np.array([[0,1],[0,-9.8/s[1]]]),s)

y0 = 0
v0 = 40
t_eval = np.linspace(0, 5, 100)

def objective(v0):
    sol = solve_ivp(f, [0, 5], [y0, v0], t_eval = t_eval)
    y = sol.y[0]
    return y[-1] - 50

v0, = fsolve(objective, 10)
print(v0)

>>> 34.49999999999999
```



PHYSICS in COMPUTER ANIMATIONS and GAMES

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

f = lambda t, s: np.dot(np.array([[0,1],[0,-9.8/s[1]]]), s)

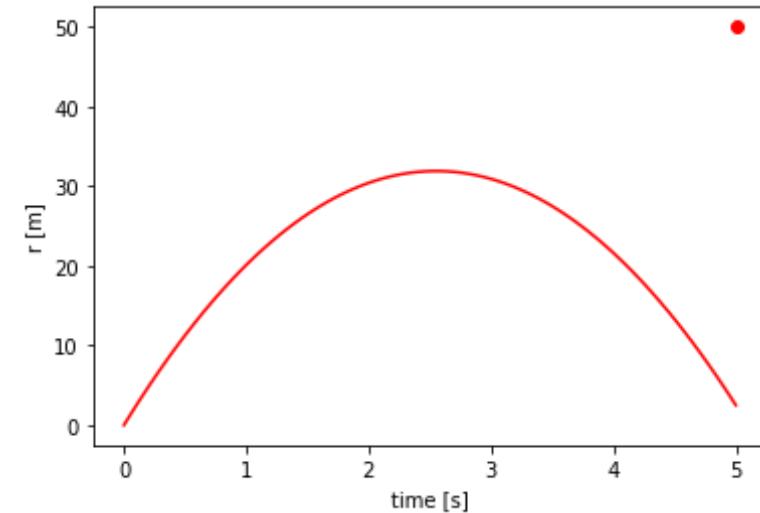
t_span = np.linspace(0, 5, 100)
y0 = 0
v0 = 25
t_eval = np.linspace(0, 5, 100)
sol = solve_ivp(f, [0, 5], \
                 [y0, v0], t_eval = t_eval)

# Position Plotting
fig, ax = plt.subplots()
ax.plot(sol.t, sol.y[0], '-r')
ax.plot(5, 50, 'ro')
ax.set_xlabel('time [s]')
ax.set_ylabel('r [m]')
plt.show()
```

`scipy.integrate.solve_ivp`

Solve an initial value problem for a system of ODEs.

This function numerically integrates a system of ordinary differential equations given an initial value





PHYSICS in COMPUTER ANIMATIONS and GAMES

Find a solution to the system of equations:

$$\begin{aligned}x_0 \cdot \cos(x_1) &= 4 \\x_1 \cdot x_0 - x_1 &= 5\end{aligned}\quad \rightarrow \quad \begin{aligned}x_0 \cdot \cos(x_1) - 4 &= 0 \\x_1 \cdot x_0 - x_1 - 5 &= 0\end{aligned}$$

```
import numpy as np
from scipy.optimize import fsolve
```

```
def func(x):
    return [x[0] * np.cos(x[1]) - 4,
            x[1] * x[0] - x[1] - 5]
```

```
# The starting estimate [1, 1] for the roots of func(x) = 0.
root = fsolve(func, [1, 1])
print(root)
print(np.isclose(func(root), [0.0, 0.0])) # func(root) should be 0.0
```

```
[6.50409711 0.90841421]
[ True  True]
```

```
numpy.isclose(a, b, rtol=1.000000000000001e-05, atol=1e-08, equal_nan=False)
```

Returns a boolean array where two arrays are element-wise equal within a tolerance.



PHYSICS in COMPUTER ANIMATIONS and GAMES

```
import numpy as np
from scipy.integrate import solve_ivp
from scipy.optimize import fsolve

f = lambda t, s: np.dot(np.array([[0,1],[0,-9.8/s[1]]]),s)

y0 = 0
v0 = 40
t_eval = np.linspace(0, 5, 100)

def objective(v0):
    sol = solve_ivp(f, [0, 5], [y0, v0], t_eval = t_eval)
    y = sol.y[0]
    return y[-1] - 50

v0, = fsolve(objective, 10)
print(v0)

>>> 34.49999999999999
```



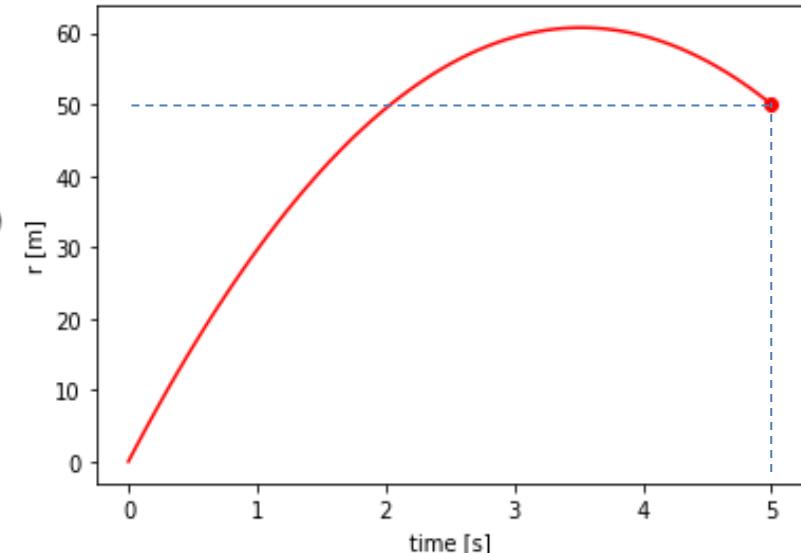
PHYSICS in COMPUTER ANIMATIONS and GAMES

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

f = lambda t, s: np.dot(np.array([[0,1],[0,-9.8/s[1]]]), s)

t_span = np.linspace(0, 5, 100)
y0 = 0
v0 = 34.49999999999999
t_eval = np.linspace(0, 5, 100)
sol = solve_ivp(f, [0, 5], \
                 [y0, v0], t_eval = t_eval)

# Position Plotting
fig, ax = plt.subplots()
ax.plot(sol.t, sol.y[0], '-r')
ax.plot(5, 50, 'ro')
ax.set_xlabel('time [s]')
ax.set_ylabel('r [m]')
plt.show()
```





PHYSICS in COMPUTER ANIMATIONS and GAMES

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

f = lambda t, s: np.dot(np.array([[0,1],[0,-9.8/s[1]]]), s)

y0 = 0
t_eval = np.linspace(0, 5, 100)

def objective(v0):
    sol = solve_ivp(f, [0, 5], \
                    [y0, v0], t_eval = t_eval)
    y = sol.y[0]
    return y[-1] - 50

for v0_guess in range(1, 100, 10):
    v0, = fsolve(objective, v0_guess)
    print(f"Init: {v0_guess}, Result: {v0}")
```

```
Init: 1, Result: 34.499999999999986
Init: 11, Result: 34.499999999999986
Init: 21, Result: 34.49999999999999
Init: 31, Result: 34.49999999999999
Init: 41, Result: 34.49999999999999
Init: 51, Result: 34.499999999999986
Init: 61, Result: 34.499999999999986
Init: 71, Result: 34.499999999999986
Init: 81, Result: 34.499999999999986
Init: 91, Result: 34.499999999999986
```

Note that changing the initial guesses does not change the result, which means that this method is **stable**



PHYSICS in COMPUTER ANIMATIONS and GAMES

Non-Linear Equations

A Non-linear equation is a type of equation. The degree in non-linear equations is two or more than two. The general equation of a linear equation is $Ax + By + C = 0$ is a linear equation. Other than that are a non-linear equation. The general equation is :

$$Ax^2 + By^2 = C \rightarrow Ax^2 + By^2 - C = 0$$

Where A, B, and C are constants, x and y are variables. It forms a curve when it is plotted on a graph.



PHYSICS in COMPUTER ANIMATIONS and GAMES

What is fsolve?

It is a function in a scipy module that returns the roots of non-linear equations:

```
scipy.optimize.fsolve (func, x0, args=(), fprime=None,  
full_output=0, col_deriv=0, xtol=1.49012e-08, maxfev=0, band=None,  
epsfcn=None, factor=100, diag=None)
```

Parameters

func: It is a function that takes an argument and returns the value.

x0: ndarray, It is a starting estimate for the root of $\text{fun}(x)=0$.

args: Tuple, it is an extra argument to the function, optional.

Returns

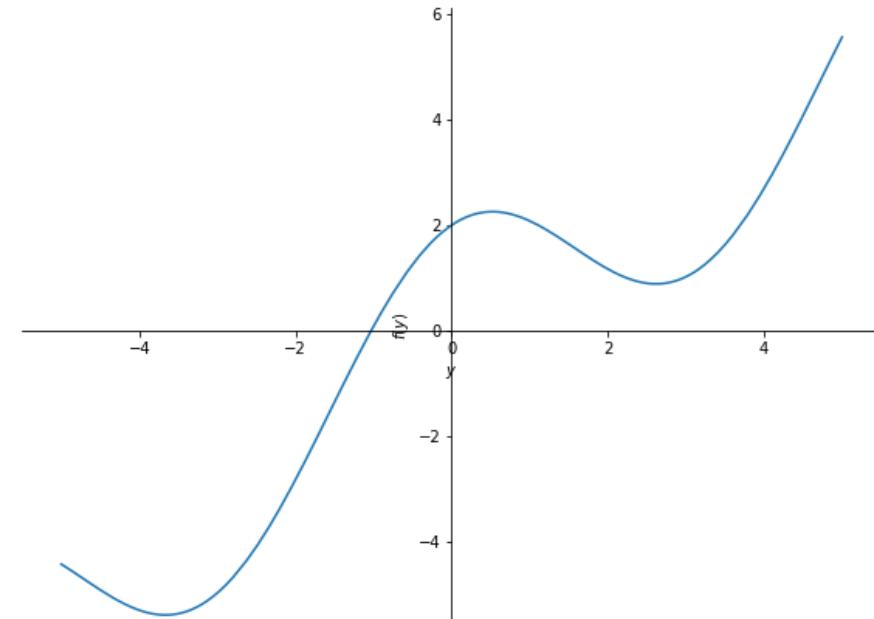
x: ndarray, It is a solution.



PHYSICS in COMPUTER ANIMATIONS and GAMES

To find the roots of an equation $y+2\cos(y)$ starting point - 0.2?

```
from sympy import *
init_printing(use_unicode=True)
y = symbols('y')
plot(y+2*cos(y), (y, -5, 5))
nsolve(y+2*cos(y), y, 0.2)
-1.0298665293226
```



```
sympy.solvers.solvers.nsolve(*args, dict=False, **kwargs) [source]
```

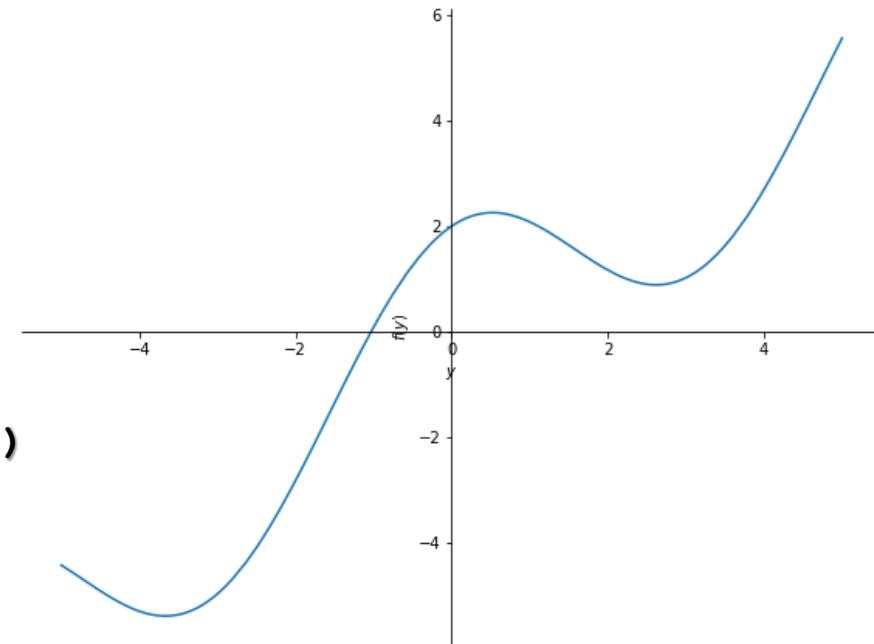
Solve a **nonlinear** equation system **numerically**:



PHYSICS in COMPUTER ANIMATIONS and GAMES

To find the roots of an equation $y+2\cos(y)$ starting point – 0.2?

```
from math import cos  
import scipy.optimize  
  
def fun(y):  
    x = y + 2*cos(y)  
    return x  
  
x = scipy.optimize.fsolve(fun, 0.2)  
  
print (x)  
[-1.02986653]
```





PHYSICS in COMPUTER ANIMATIONS and GAMES

To solve an equations for x^2+y-4 and $x+ y^2+3$?

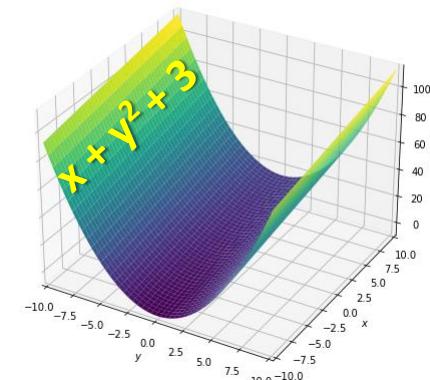
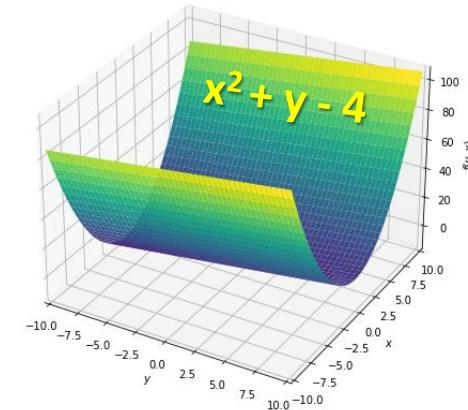
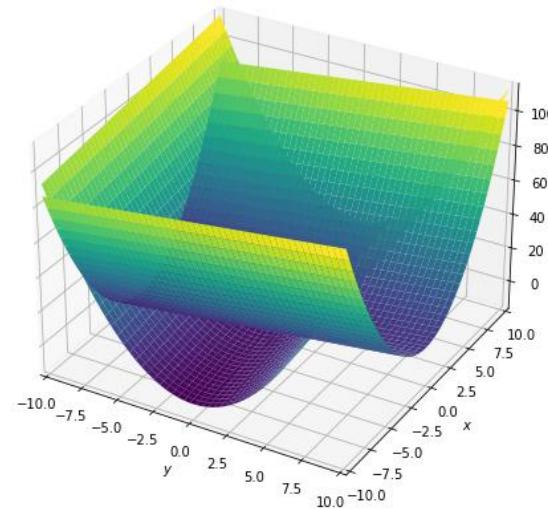
First plot the equation $x^2+y-4 = 0$ then $x+ y^2+3$

```
from sympy import symbols, nsolve  
from sympy.plotting import plot3d
```

```
x, y = symbols('x y')
```

```
e1 = x**2+y-4  
e2 = x+y**2+3
```

```
plot3d(e1)  
plot3d(e2)  
plot3d(e1, e2)
```





PHYSICS in COMPUTER ANIMATIONS and GAMES

To solve an equations for x^2+y-4 and $x+y^2+3$?

```
from sympy import symbols, nsolve
from sympy.plotting import plot3d

x, y = symbols('x y')

e1 = x**2+y-4
e2 = x+y**2+3

nsolve((e1, e2), (x, y), (0.1, 1))

ValueError: Could not find root within given tolerance.
(0.953441320283561482907 > 2.16840434497100886801e-19)
Try another starting point or tweak arguments.
```



PHYSICS in COMPUTER ANIMATIONS and GAMES

To solve an equations for x^2+y-4 and $x+ y^2+3$?

```
import scipy.optimize

def fun(variables) :
    (x,y)= variables
    eqn_1 = x**2+y-4
    eqn_2 = x+y**2+3
    return [eqn_1,eqn_2]

result = scipy.optimize.fsolve(fun, (0.1, 1))
print(result)

[-2.08470396 -0.12127194]
```



PHYSICS in COMPUTER ANIMATIONS and GAMES

To solve an equations for $x^2+y^2+z^2-1$ and $x-2y+3z-0.5$ and $x+y+z$?

```
import numpy as np
import scipy.optimize as opt

def fun(var):
    x = var[0]
    y = var[1]
    z = var[2]
    Func= np.empty((3))
    Func[0] = x**2 + y**2 + z**2 - 1
    Func[1] = x - 2*y + 3*z - 0.5
    Func[2] = x + y + z
    return Func

a= np.array([2,1,3])
b= opt.fsolve(fun, a)
print(b)
>>>[-0.78990497  0.21596199  0.57394298]
```