



Handwritten physics equations and a diagram of a rocket:

Diagram: A rocket pointing upwards with velocity \vec{v} and acceleration \vec{G} . Below it is a vector \vec{u} .

$$\Sigma F = ma = \dot{p}(t) = -G \Delta t$$
$$m \Delta v + u \Delta m + \Delta v \Delta m = -G \Delta t$$
$$m \frac{\Delta v}{\Delta t} + u \frac{\Delta m}{\Delta t} + \Delta v \frac{\Delta m}{\Delta t} = -G$$
$$\Delta t \rightarrow 0$$
$$m v'(t) + u m'(t) = -mg$$
$$v'(t) + g = -u \frac{m'(t)}{m} \quad || \int dt$$
$$v(t) + u \ln(m) = -gt + C$$
$$v(0) = 0 \Rightarrow C = u \ln(m_0)$$
$$v(t) = -gt + u \ln \frac{m_0}{m}$$

Free-Fall in 10 different Languages

me 😊



C4dynamics

BACKGROUND

$$\dot{y} = g \cdot t$$

y : height

\dot{y} : falling rate

g : gravity acceleration, about 9.8

t : time

SOLUTION

$$y = y_0$$
$$y = y + g \cdot t \cdot dt$$

- A solution in the form of a numerical integration
- Not exact as analytical solution, but more flexible
- Let's go with it...

give me the gift!



1. PYTHON



```
while t <= t_max:
```

```
    y += g * t * dt
```

```
    t += dt
```

(simple Euler integration)

gravity (acceleration, 9.8)

init with:

y = 1000, t = 0, dt = 0.01, t_max = 15

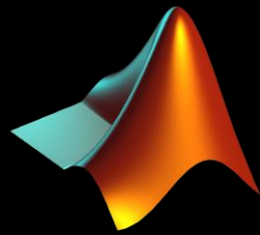
g = -9.8

give me the gift!



C4dynamics

2. MATLAB



```
while t <= t_max
```

```
    y = y + g * t * dt;
```

```
    t = t + dt;
```

```
end
```

(dt is just the time step)

gravity (acceleration, 9.8)

init with:

y = 1000, t = 0, dt = 0.01, t_max = 15

g = -9.8

give me the gift!



C4dynamics

3. C



```
for (int i = 1; t < t_max; i++) {  
    y += g * t * dt;  
    t += dt;  
}
```

(remember that y is the height above ground)

gravity (acceleration, 9.8)

init with:

y = 1000, t = 0, dt = 0.01, t_max = 15

g = -9.8

give me the gift!



C4dynamics

4. C++



```
for (int i = 1; t < t_max; i++) {  
    y += g * t * dt;  
    t += dt;  
}
```

gravity (acceleration, 9.8)

init with:

y = 1000, t = 0, dt = 0.01, t_max = 15

g = -9.8

give me the gift!



C4dynamics

5. JAVA



```
for (int i = 1; t < t_max; i++) {  
    y += g * t * dt;  
    t += dt;  
}
```

gravity (acceleration, 9.8)

init with:

y = 1000, t = 0, dt = 0.01, t_max = 15

g = -9.8

give me the gift!



C4dynamics

6. C#



```
for (int i = 1; i <= t_max / dt; i++) {  
    y += g * t * dt;  
    t += dt;  
}
```

gravity (acceleration, 9.8)

init with:

y = 1000, t = 0, dt = 0.01, t_max = 15

g = -9.8

give me the gift!



C4dynamics

7. JAVASCRIPT



```
for (let i = 1; t < t_max; i++) {  
    y += g * t * dt;  
    t += dt;  
}
```

gravity (acceleration, 9.8)

init with:

y = 1000, t = 0, dt = 0.01, t_max = 15

g = -9.8

give me the gift!



C4dynamics

8. R



```
for (i in 1 : round(t_max / dt)) {  
  y <- y + g * t * dt  
  t <- t + dt  
}
```

gravity (acceleration, 9.8)

init with:

y = 1000, t = 0, dt = 0.01, t_max = 15
g = -9.8

give me the gift!

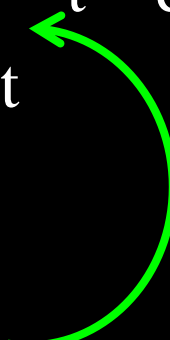


C4dynamics

9. JULIA



```
while t <= t_max
    y += g * t * dt
    t += dt
end
```




gravity (acceleration, 9.8)

init with:

y = 1000, t = 0, dt = 0.01, t_max = 15
g = -9.8

give me the gift!

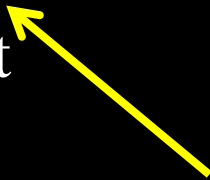


C4dynamics

10. FORTRAN



```
do i = 1, int(t_max / dt)
  y = y + g * t * dt
  t = t + dt
end do
```



gravity (acceleration, 9.8)

init with:
y = 1000, t = 0, dt = 0.01, t_max = 15
g = -9.8

give me the gift!



C4dynamics

SPECIAL BONUS!

An artistic python code doing a complete free fall model in 11 simple lines:

```
import C4dynamics as c4d

h0 = 1000
dt = 0.001
t = 0
g = -9.8

ball = c4d.datapoint(z = h0)

while ball.z >= 0:
    ball.run(dt, np.array([0, 0, g] * ball.m))
    ball.store(t)
    t += dt

ball.draw('z')

#
# for developers:
# 1) try to change the initial altitude, h0, and re-run the program.
# 2) draw also the trajectory in x axis: ball.draw('x')
#     what do you see? why there is no motion in x?
# 3) now, set initial velocity in x, change line 7 to: ball = c4d.datapoint(vx = 100,
#     z = -h0)
#     run again and draw the x trajectory.
#     what do you see now?
#     what does it do to the motion in z axis? why?
# 4) import C4dynamics as c4d in your project and use the object c4d.datapoint to
#     model your physics.
# FAQ? contact zivmeri @ linkedin \ gmail, or C4dynamics at github.
##
```



C4dynamics

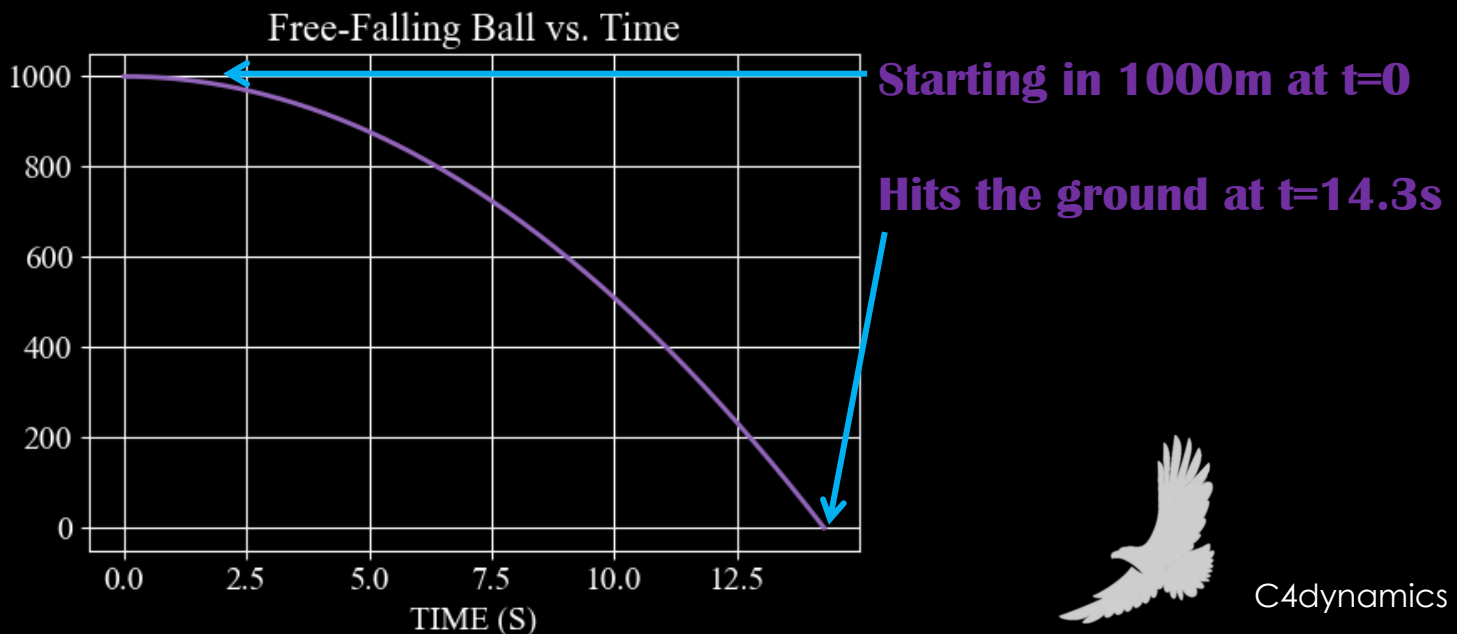
SPECIAL BONUS!

Want to make fun while developing cool algorithms for physical systems too?

Download **now** C4dynamics and run freefall.py. Follow the instructions there.

<https://github.com/C4dynamics/C4dynamics/blob/main/examples/freefall.py>

C4dynamics is a cutting-edge, high-standard algorithms development framework.





I SEE DEAD PEOPLE!

$$c = m \cdot c^2$$

$$\begin{aligned} \dot{\rho} &= c_T \cdot \cos \delta - v_m \cdot \cos \delta \\ \dot{\lambda} &= \omega \\ \dot{\omega} &= -2 \cdot \omega (c_T \cdot \cos \delta - v_m \cdot \cos \delta) / \rho \\ &\quad + a_T \cdot \cos \delta / \rho - g_m \cdot \cos \delta / \rho \end{aligned}$$

$$\delta_T = a_T / a_T$$

$$\delta_m = g_m / \sigma_m$$

$$\begin{aligned} \sqrt{a_c} &= N \cdot v_m \cdot \dot{\lambda} \\ \sqrt{y} &= C \cdot x \\ \sqrt{f} &= W \cdot x \end{aligned}$$

$$\dot{x} = A \cdot x + b \cdot u$$

$$y = C \cdot x$$

The diagram illustrates the geometry of missile guidance. A missile is shown at the origin, with its velocity vector v_m and acceleration vector a_c . The target is at a distance ρ along the line of sight. The line of sight is at an angle δ from the horizontal. The target's velocity vector v_T is shown at an angle γ from the horizontal. The target's acceleration vector a_T is shown at an angle δ from the horizontal. The target's position is given by x and y . The target's velocity is given by \dot{x} and \dot{y} . The target's acceleration is given by \ddot{x} and \ddot{y} . The target's position is also given by ρ and δ . The target's velocity is also given by v_T and γ . The target's acceleration is also given by a_T and δ . The target's position is also given by x and y . The target's velocity is also given by \dot{x} and \dot{y} . The target's acceleration is also given by \ddot{x} and \ddot{y} .

Necessary Condition for Successful Hit in pure pursuit in tail-chase situation

$$k = \frac{v_T}{\sigma_m} < 1!$$

Pure Pursuit

$y = C \cdot x$
↑
output equation

Function $q_c = P_N(\delta)$

Must be directed towards the true position of the target

Near Collision Course (NCC)

$$\lambda = 0 \rightarrow q_c = 0$$

$$\sin^{-1}$$



C4dynamics