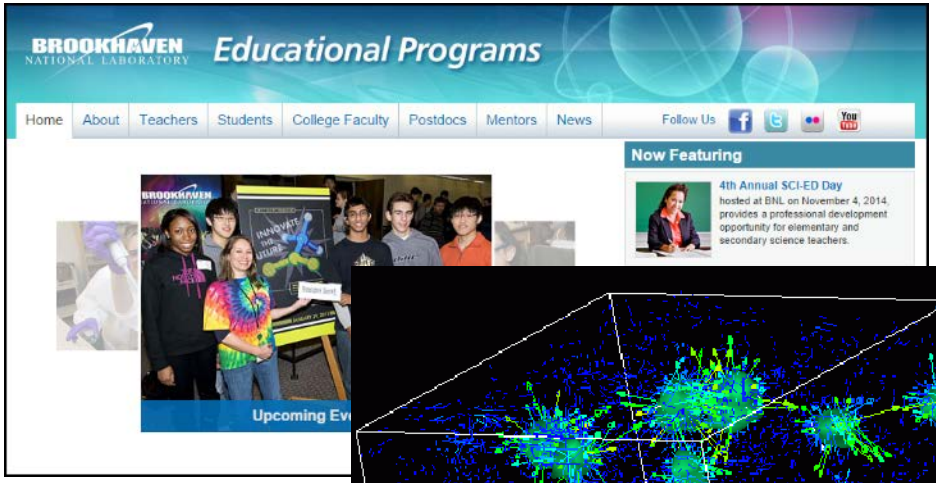
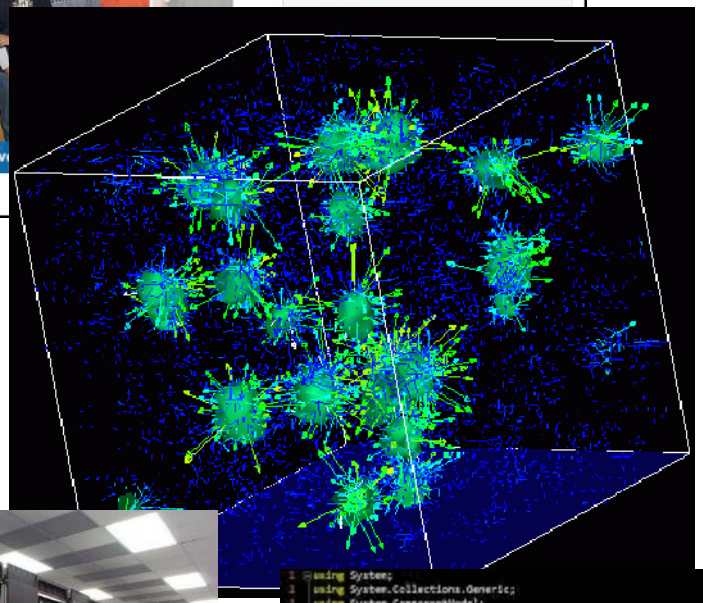
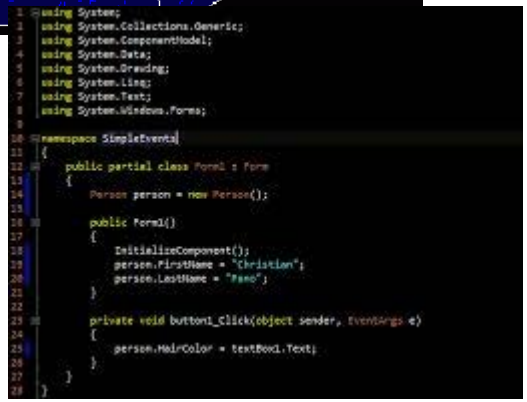






Survey of Scientific Computing (SciComp 301)

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

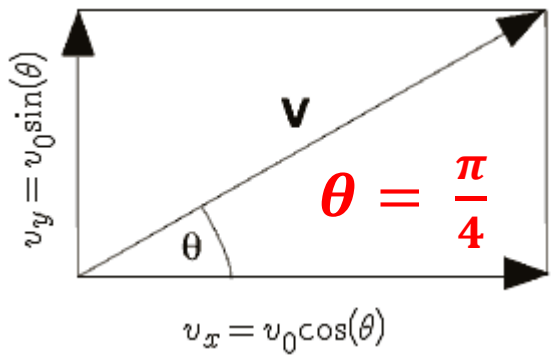
Session 19
Computational Physics

Session Goals

- How to simulate the **trajectory** of a circus cannon performer
- Implement **Euler's Method** for finding numerical solutions to physical laws represented as differential equations
  - Derive Euler's Method directly from the **first derivative**
  - Model the radioactive decay of Fluorine-18 and Carbon-14
- Appreciate the importance of **stability** in numerical solutions
  - Use Euler's Method to model the simple harmonic motion of a single (unforced, undamped) pendulum
  - Use the **Euler-Cromer** method to eliminate artificial energy gain in the long-term modeling of a system

Projectile Motion

Projectile Motion



$$x = v_0 * t * \cos(\theta)$$

$$y = v_0 * t * \sin(\theta) - \frac{1}{2} g t^2$$

$$t = \frac{x}{v_0 * \cos(\theta)}$$

$$y = \tan(\theta) * x - \frac{g}{2 * v_0^2 * cos^2(\theta)} * x^2$$

Given Range = 400m,
what does v_0 need to be?

$$v_0 = \sqrt{\frac{Range * g}{\sin 2\theta}}$$

This is the equation of motion that allows us to **plot y as x increases** from launch point to trampoline

Open Lab 1 – Projectile Motion

```
void draw(SimpleScreen& ss)
{
    ss.Clear();
    ss.DrawAxes();
    ss.DrawRectangle("red", 390, 0, 20, 5);

    if (mode == drawMode::DRAW)
    {
        PointSet psTrajectory;

        // Set fixed angle of elevation (45 degrees converted to radians)
        double theta = 45.0 * M_PI / 180.0;

        // Set accerlation due to gravity in SI units
        double gravity = 9.81;

        // Calculate height and range of trajectory
        double trajectoryHeight = pow(initialVelocity, 2)
            * pow(sin(theta), 2) / (2 * gravity);
        double trajectoryRange = 4 * trajectoryHeight / tan(theta);
```

The trampoline is 20m long x 5 m high centered 400m from cannon

View Lab 1 – Projectile Motion

```
// Set accerlation due to gravity in SI units
double gravity = 9.81;

// Calculate height and range of trajectory
double trajectoryHeight = pow(initialVelocity, 2)
    * pow(sin(theta), 2) / (2 * gravity);
double trajectoryRange = 4 * trajectoryHeight / tan(theta);

// Set the number of intervals to draw across the domain
int intervals = 97;

// Calculate rate to increment x with each new interval step
double deltaX = trajectoryRange / intervals;

// Calculate the trajectory of the performer
for (int i = 0; i <= intervals; i++)
{
    // Calculate  WORLD coordinates for current x and f(x)
    double x = deltaX * i;
    double y = x * tan(theta) - pow(x, 2) *
        (gravity /
        (2 * pow(initialVelocity, 2)
            * pow(cos(theta), 2)));
    psTrajectory.add(x, y);
}

// Draw the trajectory
ss.DrawLines(&psTrajectory, "blue", 3, false, false, 10);

// Show results [dead center = sqrt(3924) = 62.64]
string msg = ((initialVelocity >= 62.34) && (initialVelocity <= 63.64))
    ? "Safe Landing!" : "** Splat! **";
cout << msg << endl;
```

$$y = \tan(\theta) * x - \frac{g}{2 * v_0^2 * cos^2(\theta)} * x^2$$

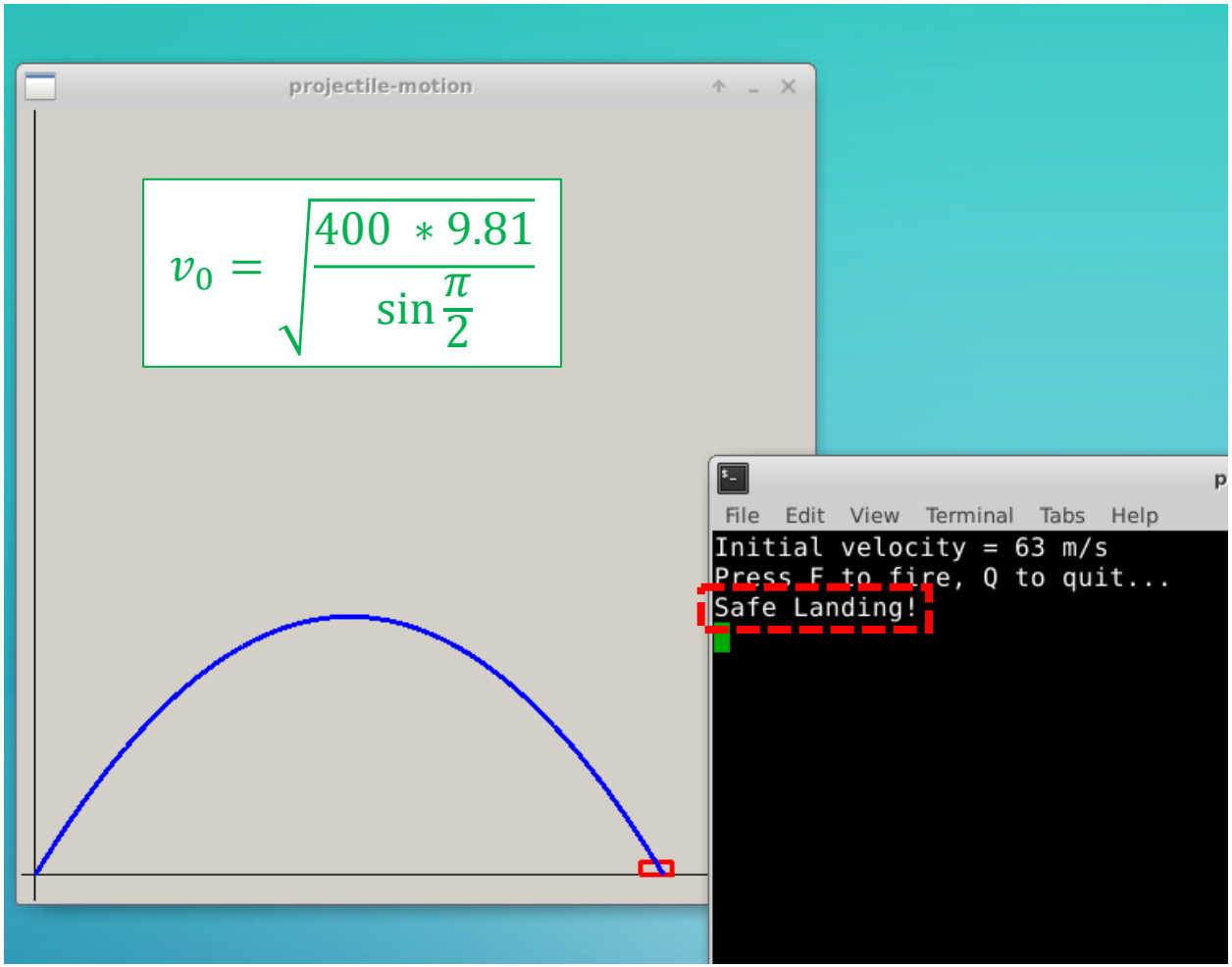
Run Lab 1 – Projectile Motion

Edit Lab 1 – Projectile Motion

```
int main()
{
    SimpleScreen ss(draw, eventHandler);
    ss.SetZoomFrame("white", 3);

    ss.SetWorldRect(-10, -10, 500, 300);

    initialVelocity = 45.0;

    cout << "Initial velocity = "
        << initialVelocity << " m/s" << endl
        << "Press F to fire, Q to quit..." << endl;

    ss.HandleEvents();

    return 0;
}
```

Change this initial velocity v_0 !

Run Lab 1 – Projectile Motion



$$v_0 = \sqrt{\frac{400 * 9.81}{\sin\frac{\pi}{2}}}$$

Modelling Nuclear Decay

$N(t) \equiv$ number of nuclei at time t

$\tau \equiv$ mean lifetime (half life)

$$\frac{dN}{dt} = -\frac{N(t)}{\tau}$$

$$\frac{dN}{dt} = \frac{N(t+\Delta t) - N(t)}{\Delta t}$$

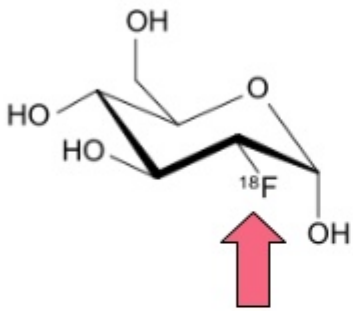
$$-\frac{N}{\tau} = \frac{N(t+\Delta t) - N(t)}{\Delta t}$$

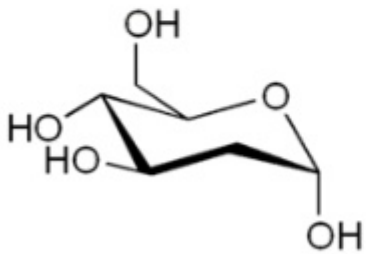
This is Euler's Method

$$N(t+\Delta t) = N(t) - \frac{N(t)}{\tau}\Delta t$$

Fluorine-18

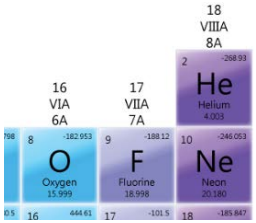
Example: FDG



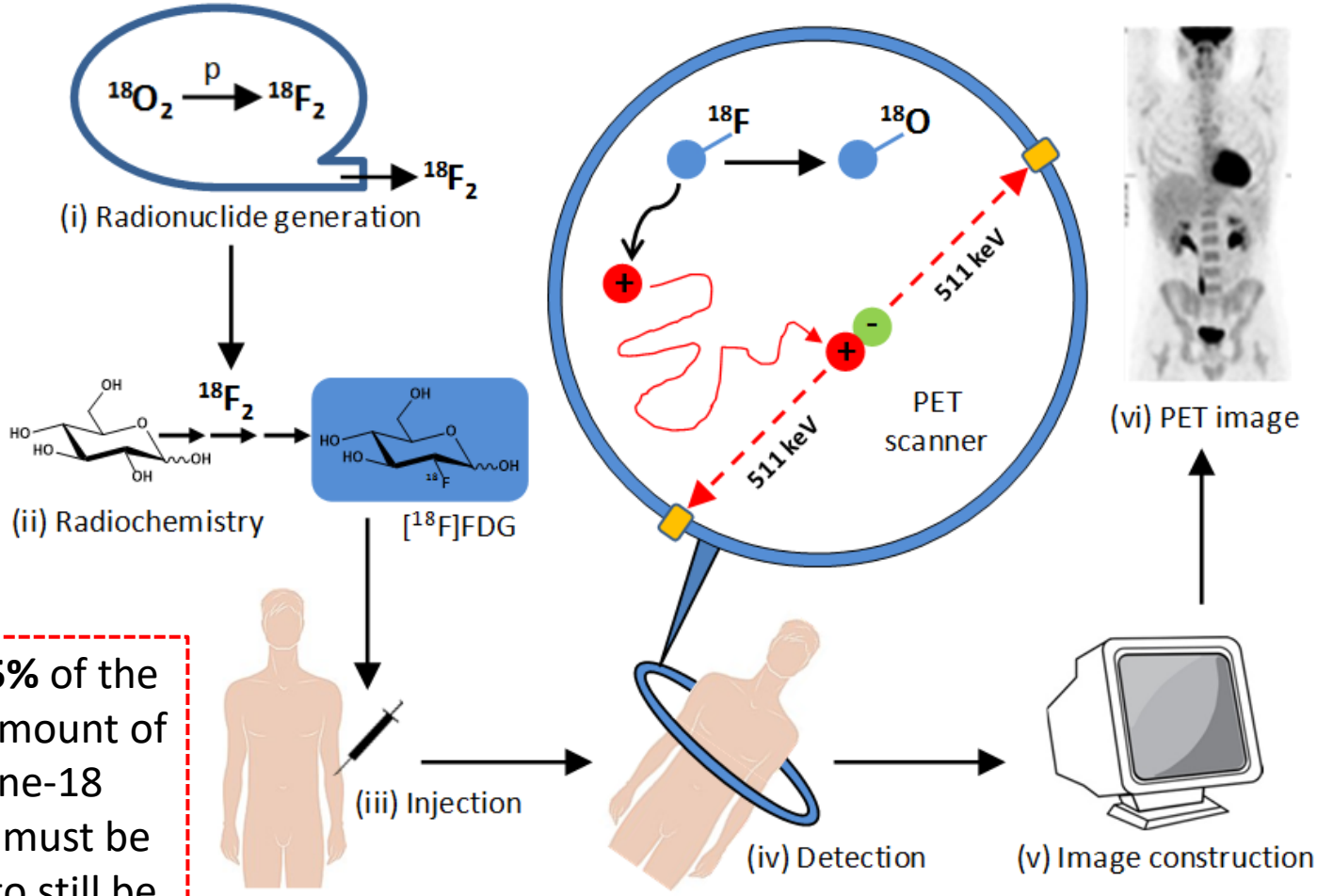


2-Deoxy-D-Glucose (2DG)

- Fluorodeoxyglucose is a radiopharmaceutical is a glucose analog with the radioactive isotope Fluorine-18 in place of OH
- ^{18}F has a half life of 110 minutes
- FDG is taken up by high glucose using cells such as brain, kidney, and cancer cells.
- Once absorbed, it undergoes a biochemical reaction whose products cannot be further metabolized, and are retained in cells.
- After decay, the ^{18}F atom becomes a harmless non-radioactive heavy oxygen $^{18}O^-$ that joins up with a hydrogen atom, and forms glucose phosphate that is eliminated via carbon dioxide and water



Fluorine-18



At least **5%** of the original amount of Flourine-18 injected must be present to still be detectable

Open Lab 2 – Fluorine-18 Decay

```
7   int main()
8   {
9       // Set number of time steps in simulation
10      const int timeSteps{ 100 };
11
12      double timeAt[timeSteps];
13      double nuclei[timeSteps];
14
15      // Set percent of nuclei initially present
16      nuclei[0] = 100;
17
18      // Half-life of Fluorine-18 (secs to hours)
19      const double halfLife{ 6586.0 / 60 / 60 };
20
21      // Duration of simulation (hours)
22      const double endTime{ 12 };
23
24      // Calculate time step (delta t)
25      const double deltaTime{ endTime / timeSteps };
26
27      // Calculate decay factor
28      const double decayFactor = deltaTime / halfLife;
29
30      // Set initial time
31      timeAt[0] = 0.0;
32
```

View Lab 2 – Fluorine-18 Decay

$$\frac{dN}{dt} = -\frac{N}{\tau}$$

$$N(t + \Delta t) = N(t) - \frac{N(t)}{\tau}\Delta t$$

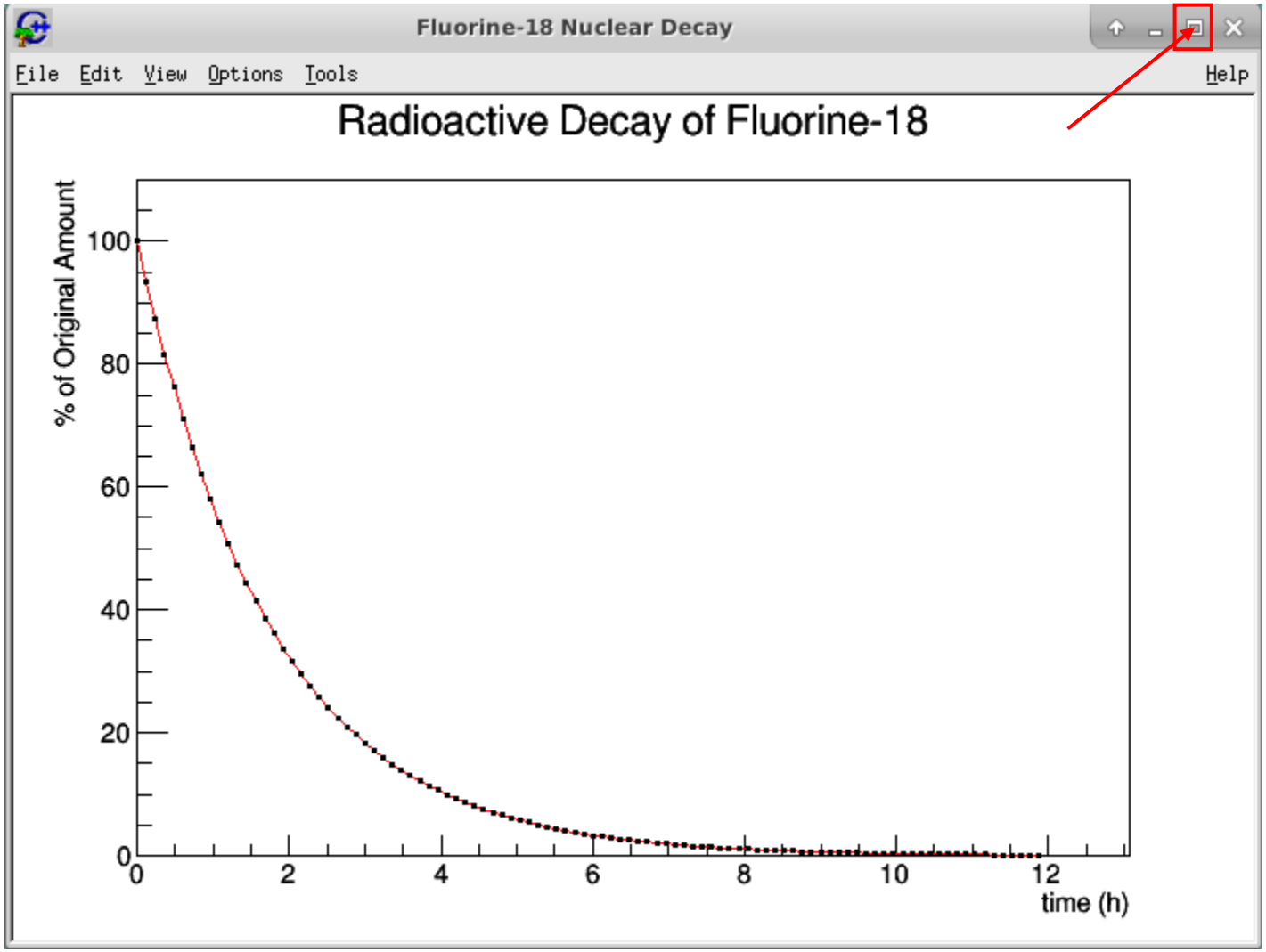
This is Euler's Method

```
// Perform Euler method to estimate differential equation
for (int step{}; step < timeSteps - 1; ++step) {
    nuclei[step + 1] = nuclei[step] - nuclei[step] * decayFactor;
    timeAt[step + 1] = timeAt[step] + deltaTime;
}
```

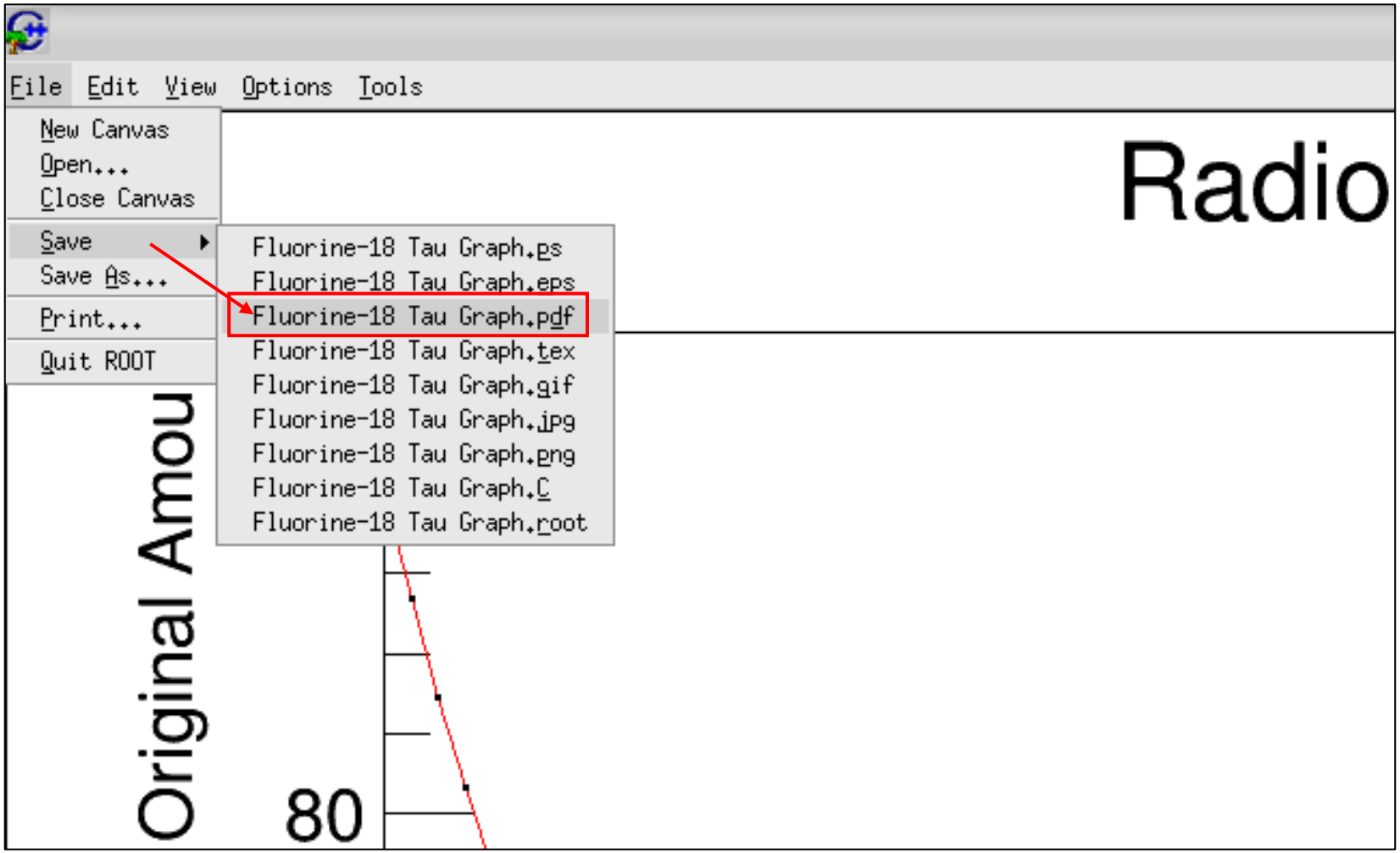
Run Lab 2 – Fluorine-18 Decay

```
    // Graph the decay curve using CERN's ROOT libraries
    TApplication* theApp =
        new TApplication("Differential Equations", nullptr, nullptr);

    TCanvas* c1 = new TCanvas("Fluorine-18 Tau Graph");
    c1->SetTitle("Lab 1 - Nuclear Decay");

    TGraph* g1 = new TGraph(timeSteps, timeAt, nuclei);

    g1->SetTitle("Radioactive Decay of Fluorine-18;time (h);% of Original Amount");
    g1->SetMarkerStyle(kFullDotMedium);
    g1->SetLineColor(2);
    g1->Draw();

    theApp->Run();

    delete g1;
    delete c1;
    delete theApp;

    return 0;
}
```

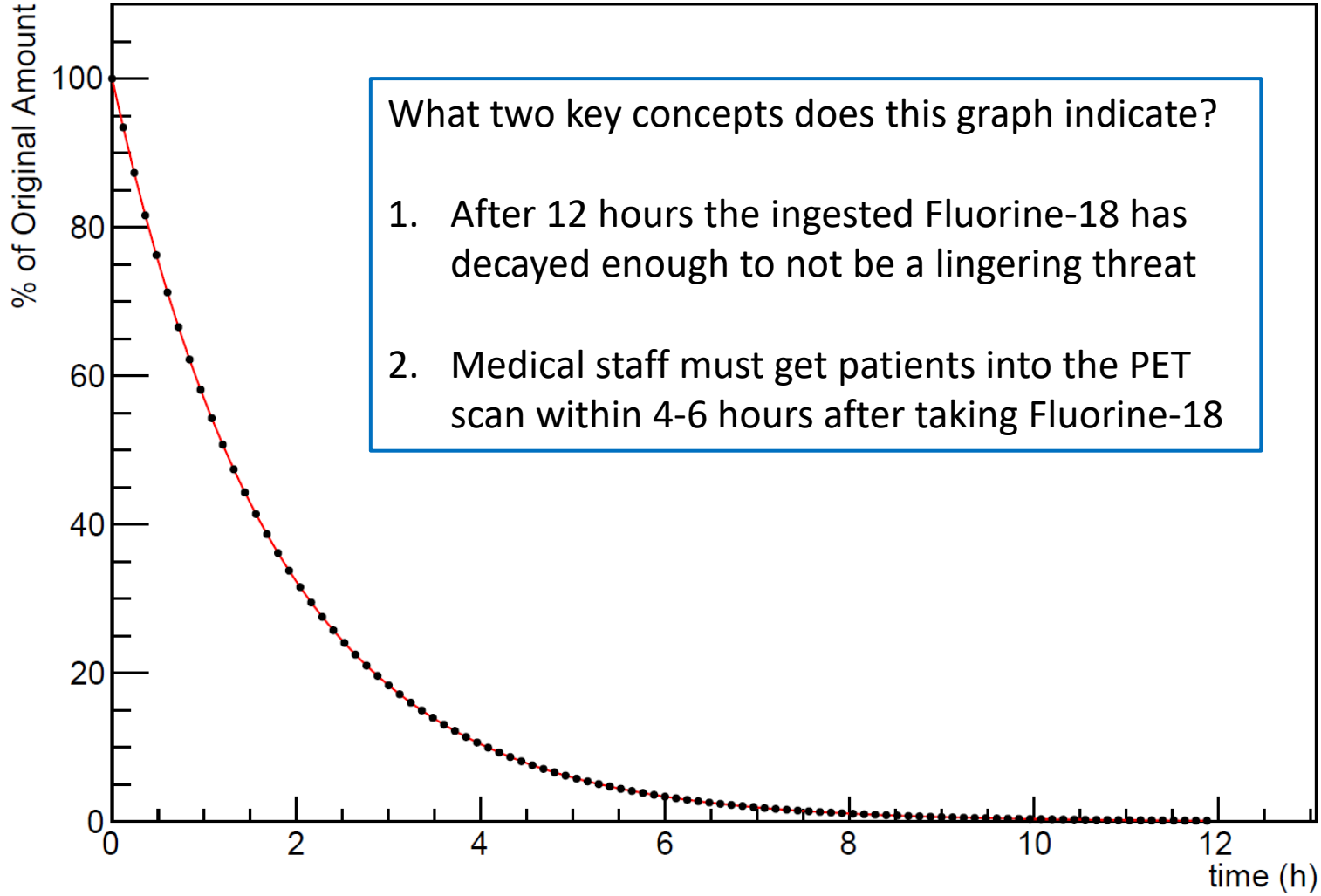
Check Lab 2 – Fluorine-18 Decay



Check Lab 2 – Fluorine-18 Decay



Radioactive Decay of Fluorine-18



What two key concepts does this graph indicate?

1. After 12 hours the ingested Fluorine-18 has decayed enough to not be a lingering threat

2. Medical staff must get patients into the PET scan within 4-6 hours after taking Fluorine-18

Modelling Carbon-14 Decay

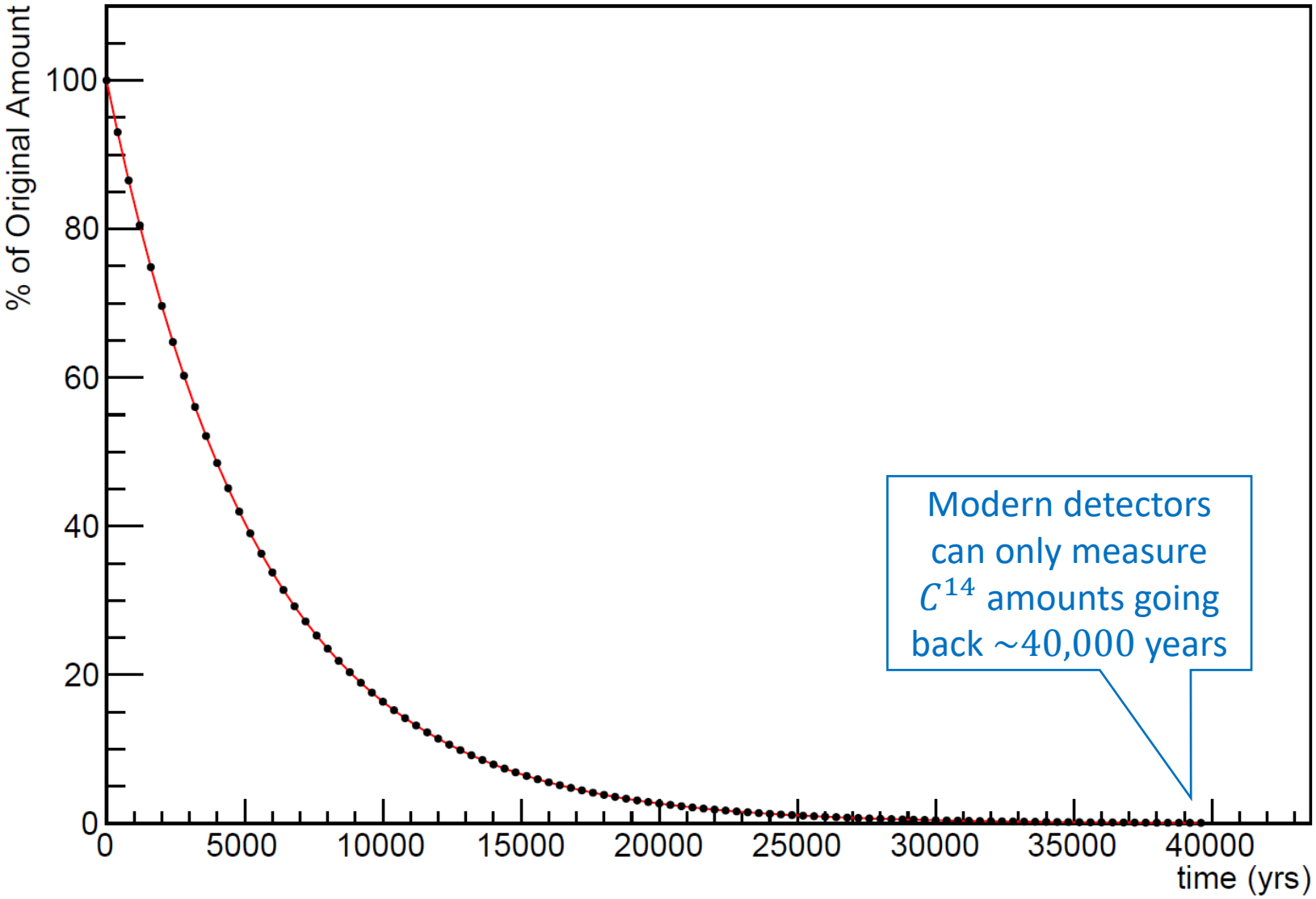
- Radio carbon dating uses $\boldsymbol{C^{14}}$ isotopes to date items
- During their lifetime, organisms absorb a certain amount of **Carbon-14** that naturally exists in their environment
- When an organism dies it **stops ingesting new Carbon-14** atoms, and the amount already present in the tissues begins to undergo radioactive decay
- It is known that C^{14} has a half-life of **5,730 years** and at least **0.1%** of the original amount of C^{14} must be present to be detectable
- Given the half-life, **how far back in time** can scientists can use radio carbon dating to determine the age of an item?

Edit Lab 3 – Modelling Carbon-14 Decay

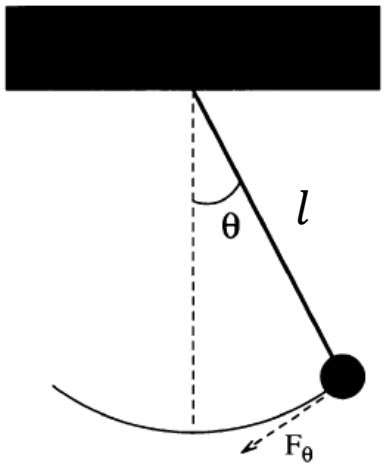
```
int main()
{
    // Set number of time steps in simulation
    const int timeSteps{ 100 };

    double timeAt[timeSteps];
    double nuclei[timeSteps];

    // Set percent of nuclei initially present
    nuclei[0] = 100;

    // Half-life of Carbon-14 (years)
    const double halfLife{ 1 };

    // Duration of simulation (years)
    const double endTime{ 1 };

    // Calculate time step (delta t)
    const double deltaTime{ endTime / timeSteps };

    // Calculate decay factor
    const double decayFactor = deltaTime / halfLife;

    // Set initial time
    timeAt[0] = 0.0;

    // Perform Euler method to estimate differential equation
    for (int step{}; step < timeSteps - 1; ++step) {
        nuclei[step + 1] = nuclei[step] - nuclei[step] * decayFactor;
        timeAt[step + 1] = timeAt[step] + deltaTime;
    }

```

Provide correct values for these two constants

Radioactive Decay of Carbon-14



Modelling a Simple Pendulum



$$F_\theta = -mg\sin\theta$$

Gravity is a restoring force

$$ml\frac{d^2\theta}{dt^2} = -mg\sin\theta$$

$$\sin\theta \approx \theta \left(for\ \theta < 22^\circ\right)$$

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

But Euler's Method works only on first order ODEs! ☹

$$s = l\theta$$

$$\frac{d^2s}{dt^2} = l\frac{d^2\theta}{dt^2}$$

$$F = ma$$

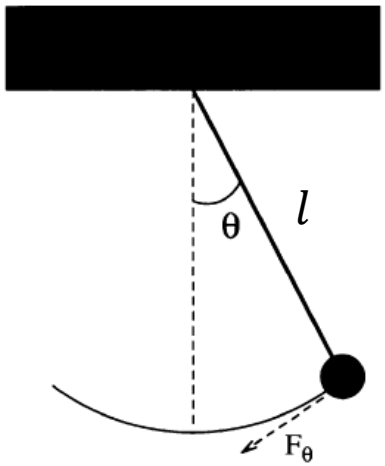
$$F = m\frac{d^2s}{dt^2}$$

$$F_\theta = ml\frac{d^2\theta}{dt^2}$$

$$\boxed{\begin{aligned}\frac{d\omega}{dt} &= -\frac{g}{l}\theta \\ \frac{d\theta}{dt} &= \omega\end{aligned}}$$

We can break the 2nd order ODE into two linked first order ODEs and use **Euler's method** on each

Modelling a Simple Pendulum



$$\frac{d\omega}{dt} = -\frac{g}{l}\theta \rightarrow \omega_{i+1} = \omega_i - \frac{g}{l}\theta_i \Delta t$$

$$\frac{d\theta}{dt} = \omega \rightarrow \theta_{i+1} = \theta_i + \omega_i \Delta t$$

Open Lab 4

Lab 4
Harmonic Motion

```
int main()
{
    // Set number of time steps in simulation
    const int timeSteps{ 250 };

    double timeAt[timeSteps];
    double omega[timeSteps];
    double theta[timeSteps];

    const double length = 1.0;  // (m)
    const double g

    const double p

    // Set initial
    omega[0] = 0.0;

    // Set initial pendulum displacement
    theta[0] = M_PI / 18.0;

    // Duration of simulation (secs)
    const double endTime{ 10 };

    // Calculate time step (delta t)
    const double deltaTime{ endTime / timeSteps };

    // Set initial time
    timeAt[0] = 0.0;

    // Perform Euler method to estimate differential equatio
    for (int step{}; step < timeSteps - 1; ++step) {
        omega[step + 1] = omega[step] - phaseConstant * theta[step] * deltaTime;
        theta[step + 1] = theta[step] + omega[step] * deltaTime;
        timeAt[step + 1] = timeAt[step] + deltaTime;
    }
```

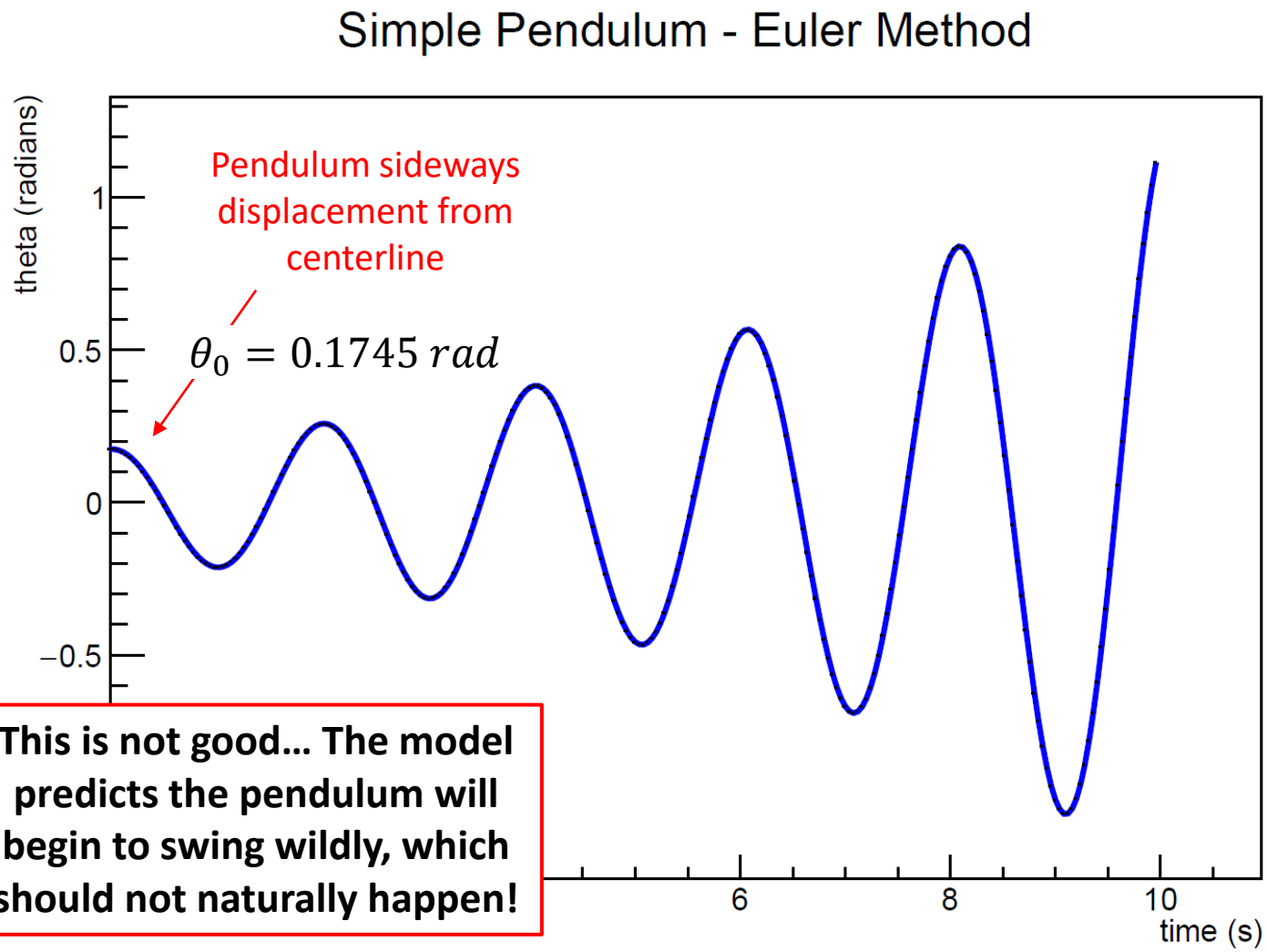
Run Lab 4

$$10^{\circ} = 10 \times \frac{2\pi}{360^{\circ}} = \frac{\pi}{18} = 0.1745\, rad$$

$$\omega_{i+1} = \omega_i - \frac{g}{l}\theta_i \Delta t$$

$$\theta_{i+1} = \theta_i + \omega_i \Delta t$$

Check Lab 4 – Harmonic Motion



This is not good… The model predicts the pendulum will begin to swing wildly, which should not naturally happen!

Instability of Euler Method For Highly Oscillatory Modes

- Increasing **`timeSteps`** **10x** does not prevent the displacement from ***growing*** after each oscillation
- This simple Euler method worked fine for modelling radioactive decay – but it is **unstable** for harmonic motion
- The **energy** in the system is *artificially* growing **over time** without any bounds

$$E = \frac{1}{2}ml^2\omega^2 + mgl(1 - \cos\theta)$$

$$\left(\theta < 22^\circ, \cos\theta \approx 1 - \frac{\theta^2}{2}\right)$$

$$E = \frac{1}{2}ml^2\left(\omega^2 + \frac{g}{l}\theta^2\right)$$

$$E_{i+1} = E_i + \frac{1}{2}mgl\left(\omega_i^2 + \frac{g}{l}\theta_i^2\right)(\Delta t)^2$$

Lab 4 – Harmonic Motion : Euler-Cromer

Euler

$$\omega_{i+1} = \omega_i - \frac{g}{l}\theta_i \Delta t$$
$$\theta_{i+1} = \theta_i + \omega_i \Delta t$$

Euler-Cromer

$$\omega_{i+1} = \omega_i - \frac{g}{l}\theta_i \Delta t$$
$$\theta_{i+1} = \theta_i + \omega_{i+1} \Delta t$$

```
// Perform Euler method to estimate differential equation
for (int step{}; step < timeSteps - 1; ++step)
{
    omega[step + 1] = omega[step] - phaseConstant * theta[step] * deltaTime;
    theta[step + 1] = theta[step] + omega[step+1] * deltaTime;
    timeAt[step + 1] = timeAt[step] + deltaTime;
}
```

Edit Lab 4 to add the +1

ife Sciences," which was one of the first textbooks to draw connections between physics and the more biological sciences. Similarly, he connected physics to its applications in industry with "Physics in Science and Industry." Those books are still widely in use, not only at Northeastern but at colleges nationwide.

"I think he was one of the first people to recognize the importance of having a text book for biological physics," Nath said.

Lab 4 – Harmonic Motion : Euler-Cromer

AAPT PHYSICS EDUCATION
AMERICAN JOURNAL of PHYSICS

Stable solutions using the Euler approximation

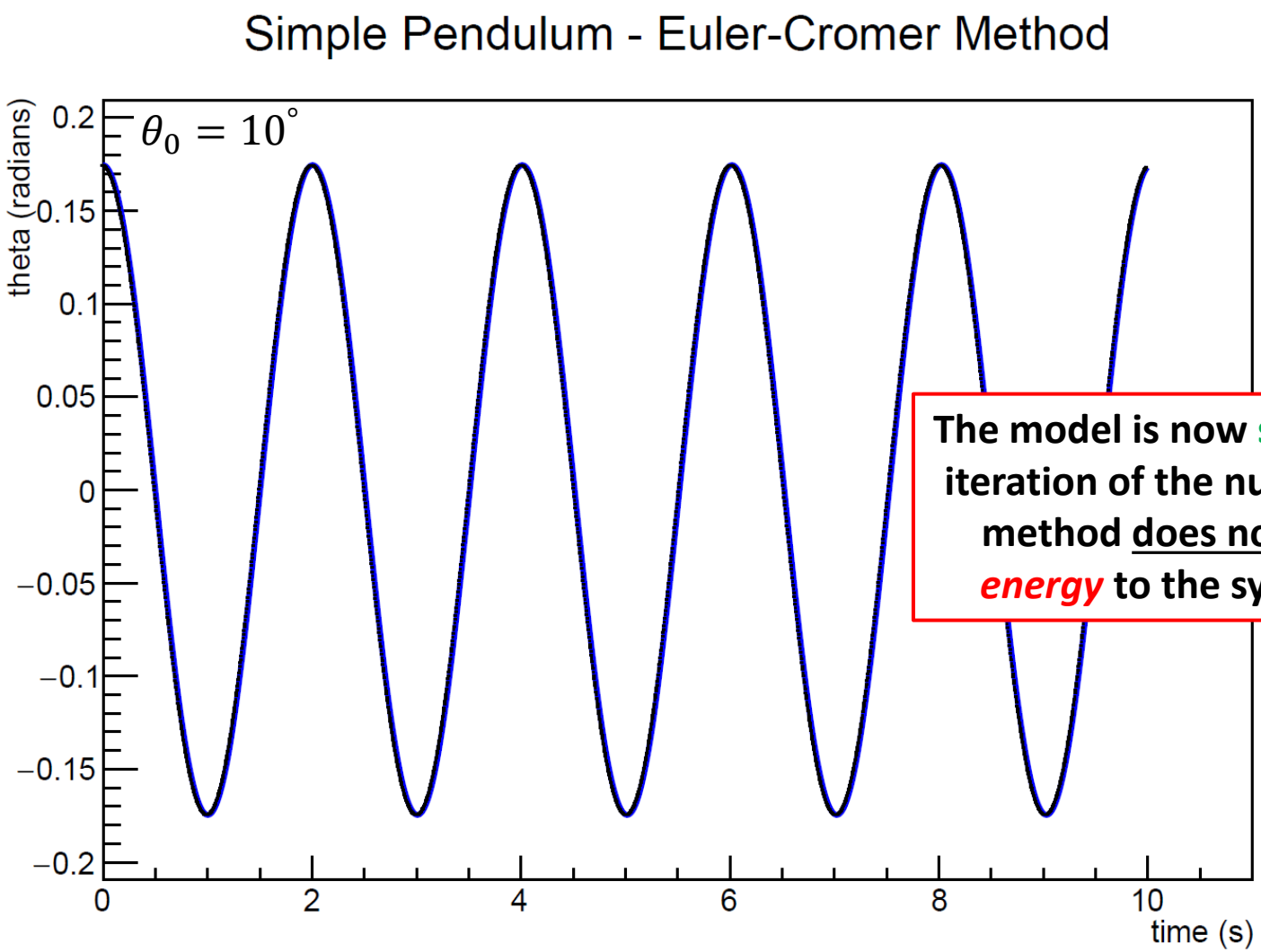
Alan Cromer[1]

+ VIEW AFFILIATIONS

Am. J. Phys. **49**, 455 (1981) http://dx.doi.org/10.1119/1.12478

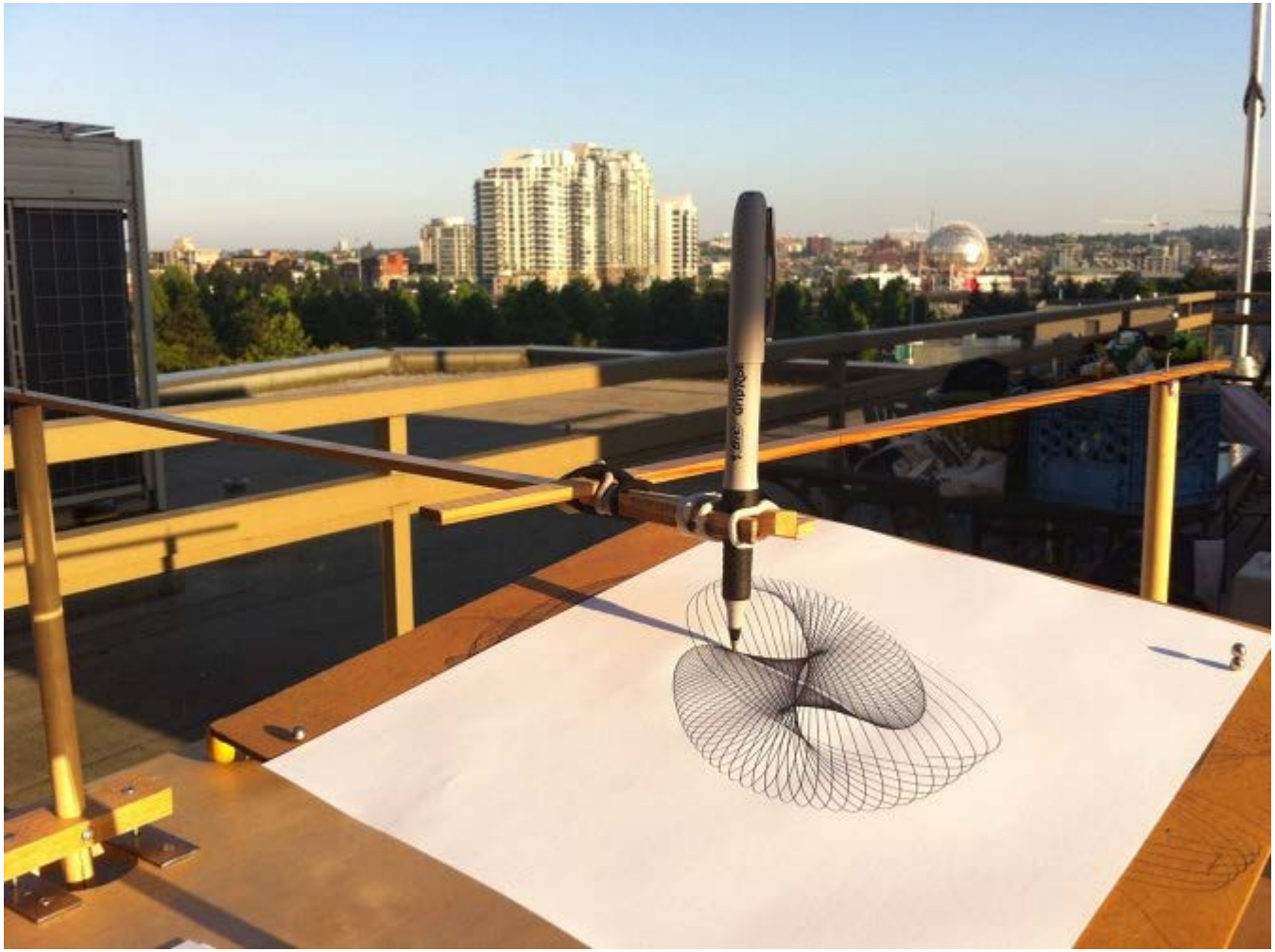
A minor modification of the standard Euler approximation for the solution of oscillatory problems in mechanics yields solutions that are stable for arbitrarily large number of iterations, regardless of the size of the iteration interval. The period of a nonlinear oscillator converges rapidly to its exact value as the size of the iteration interval is decreased. In two dimensions, closed orbits are given for the two-body Kepler problem and the restricted three-body problem can be iterated indefinitely to produce space-filling orbits. In this new approximation, the difference ΔE between the initial energy and the energy after n iterations is bounded, oscillatory, and zero when averaged over half a cycle of the motion.

Run Lab 4 – Harmonic Motion

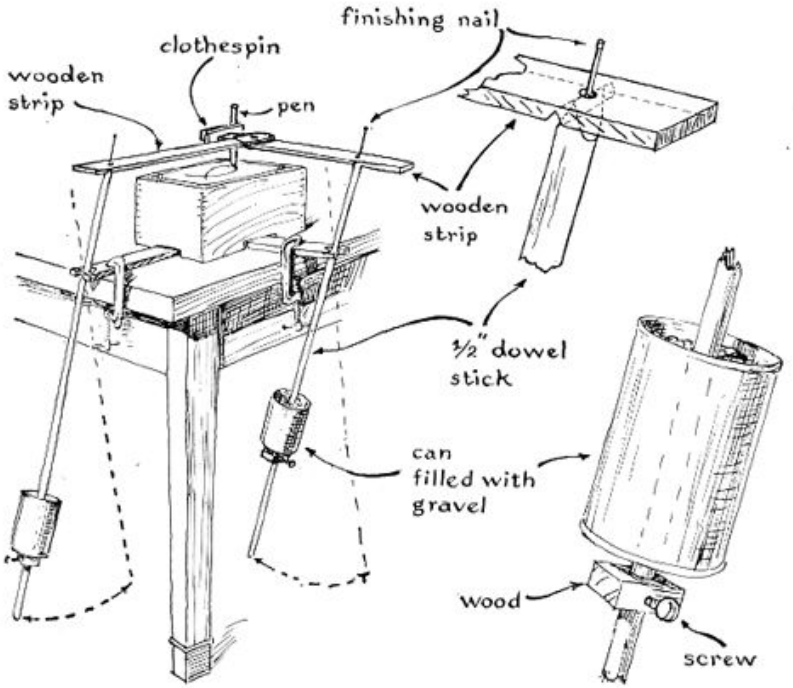


The model is now stable as iteration of the numerical method does not add *energy* to the system.

Coupled Harmonograph



Coupled Harmonograph



Open Lab 5 – Coupled Harmonograph

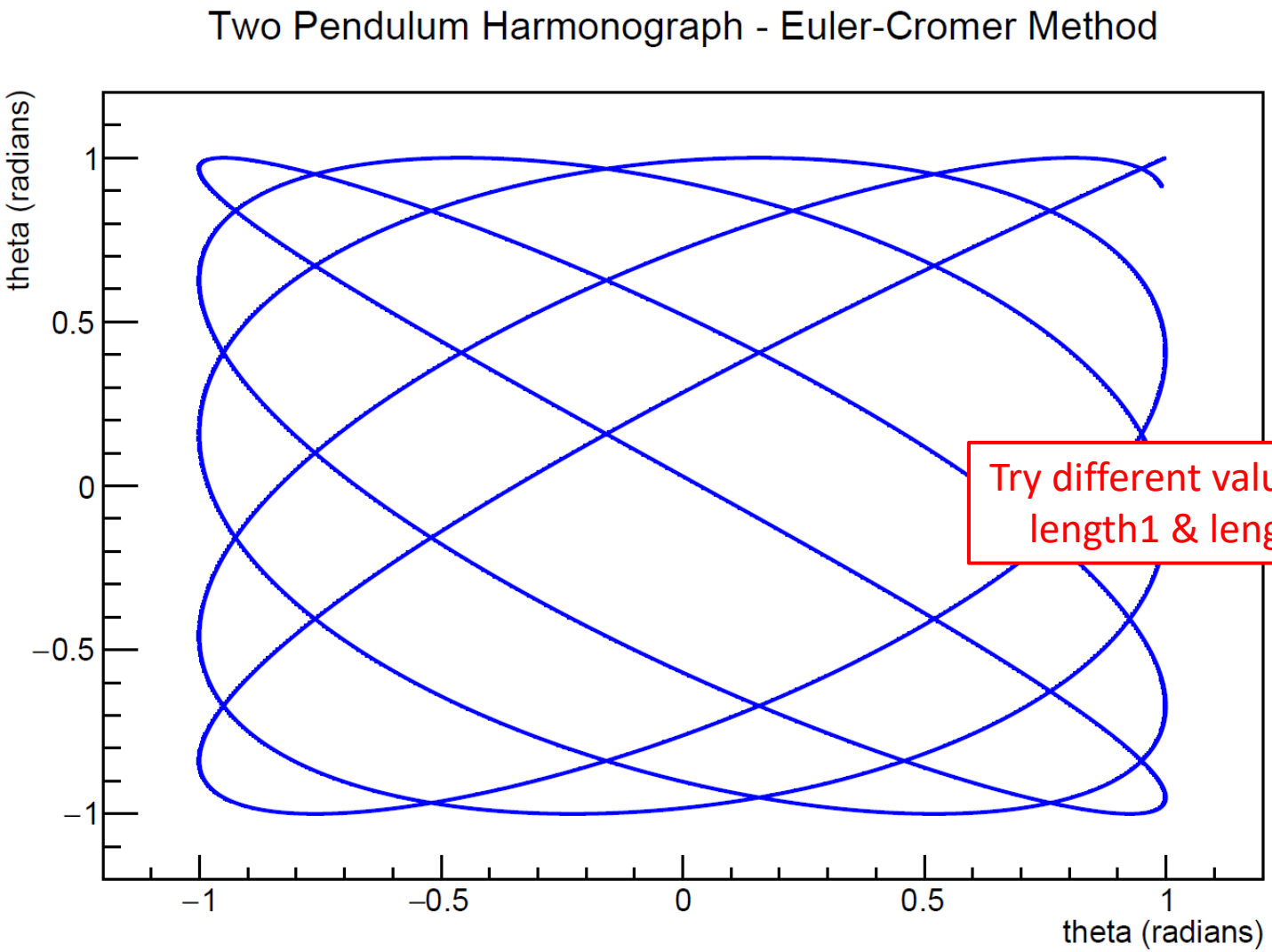
```
int main()
{
    const int timeSteps{ 2500 };
    const double endTime{ 10 };
    const double deltaTime{ endTime / timeSteps };
    double timeAt[timeSteps];
    timeAt[0] = 0.0;

    const double g = 9.8;        // (m/s^2)

    // Define first pendulum
    double omega1[timeSteps];
    double theta1[timeSteps];
    const double length1 = 1.0; // (m)
    const double phaseConstant1 = g / length1;
    theta1[0] = 1;
    omega1[0] = 0;

    // Define first pendulum
    double omega2[timeSteps];
    double theta2[timeSteps];
    const double length2 = 1.5; // (m)
    const double phaseConstant2 = g / length2;
    theta2[0] = 1;
    omega2[0] = 0;

```

View Lab 5 – Coupled Harmonograph

```
32
33      // Perform Euler-Cromer method to estimate differential equation
34      for (int step{}; step < timeSteps - 1; ++step) {
35          // First pendulum
36          omega1[step + 1] = omega1[step] - phaseConstant1 * theta1[step] * deltaTime;
37          theta1[step + 1] = theta1[step] + omega1[step + 1] * deltaTime;
38          // Second pendulum
39          omega2[step + 1] = omega2[step] - phaseConstant2 * theta2[step] * deltaTime;
40          theta2[step + 1] = theta2[step] + omega2[step + 1] * deltaTime;
41          // Update time
42          timeAt[step + 1] = timeAt[step] + deltaTime;
43      }
44
```

```
45      // Graph the decay curve using CERN's ROOT libraries
46      TApplication* theApp =
47          new TApplication("Differential Equations", nullptr, nullptr);
48
49      TCanvas* c1 = new TCanvas("Two Pendulum Harmonograph");
50      c1->SetTitle("Two Pendulum Harmonograph - Euler-Cromer Method");
51
52      TGraph* g1 = new TGraph(timeSteps, theta1, theta2);
53
```

Run Lab 5 – Coupled Harmonograph



Try different values for length1 & length2

Now you know…

- How to develop the **equations of motion** to accurately plot the 2D trajectory of a projectile moving through a uniform gravitational field
- Euler's Method (**time step analysis**) yields numeric solutions to differential equations
  - We model 2nd order differentials by representing them as a **chain of linked 1st order equations**
  - Euler-Cromer is better when modelling **harmonic oscillators**
- Increasing the number of time steps (i.e. decreasing Δt) improves the accuracy of the estimations
  - However using more time steps also causes the program to take much longer to produce an answer - scientific computing aims to balance the competing demands between greater accuracy and greater speed