Survey of Scientific Computing
(SciComp 301)

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

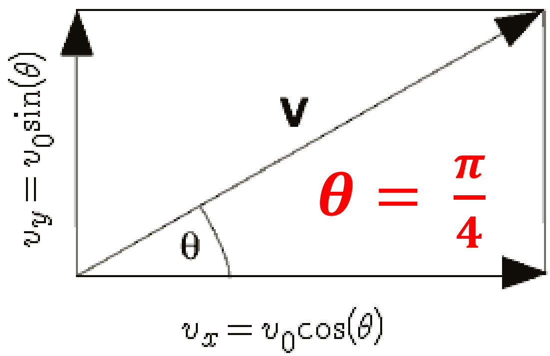
Session 19
Computational Physics

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
  - Obtain Euler's Method from **Fermat's** definition of the **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-Crome**r method to eliminate artificial energy gain in the long-term modeling of a system

Projectile Motion

Projectile Motion



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

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

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

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

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

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

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

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

```
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);
```

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);
```

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

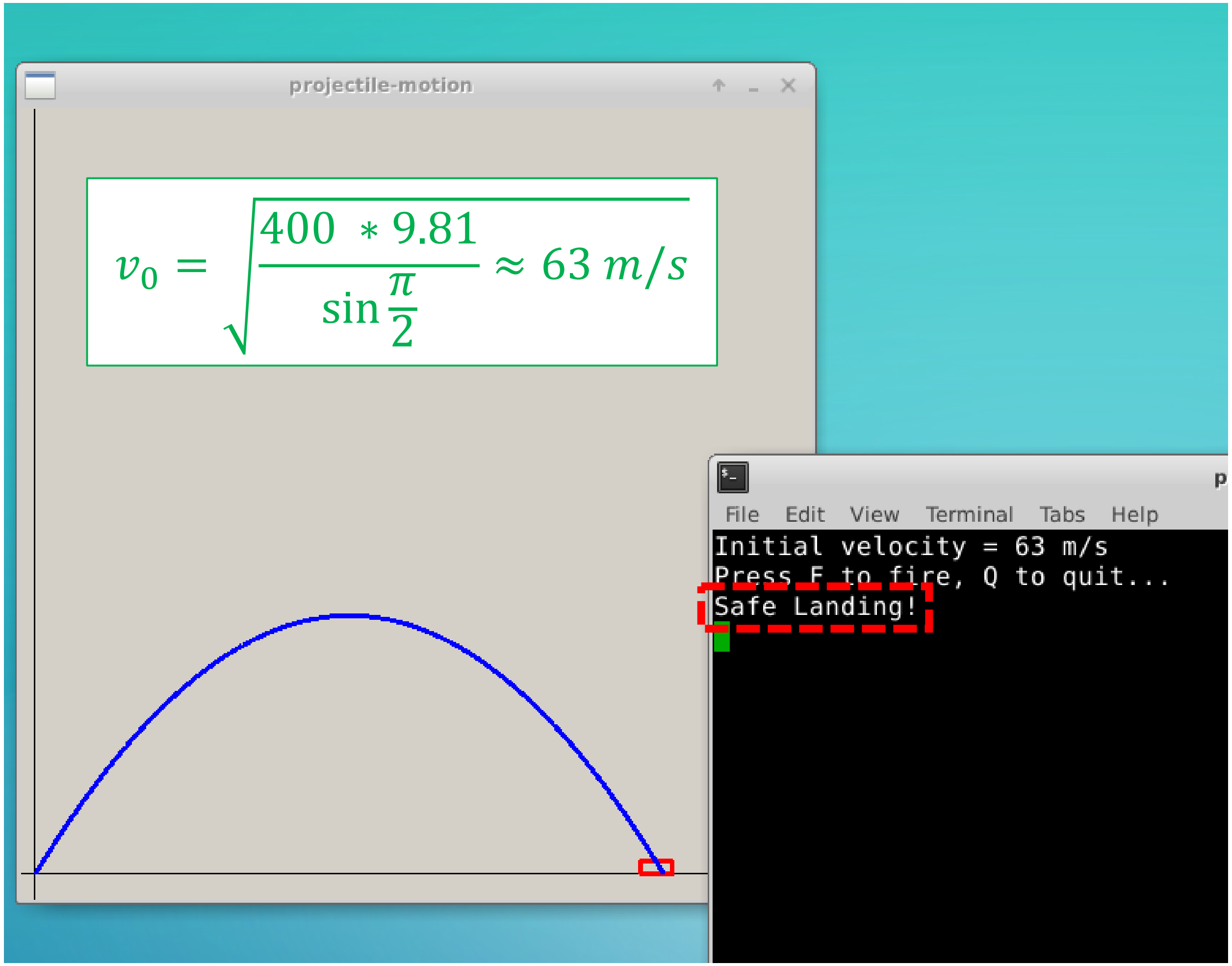
Run Lab 1 – Projectile Motion

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 $\boldsymbol{v_0}$!

Run Lab 1 – Projectile Motion



Modelling Nuclear Decay

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

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

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

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

Fermat's Difference Quotient

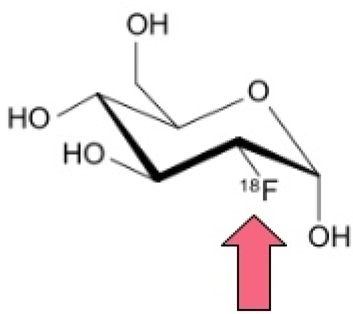
$$-\frac{N(t)}{\tau} = \frac{N(t+\Delta t) - N(t)}{\Delta t}$$

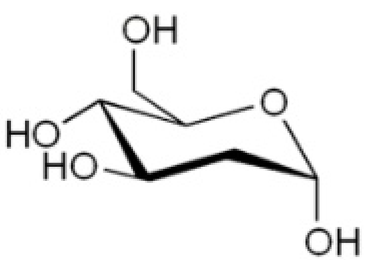
$$N(t+\Delta t) = N(t) - \frac{N(t)}{\tau}\Delta t$$

This is Euler’s Method

Fluorine-18

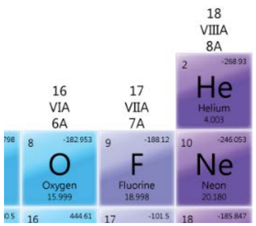
Example: FDG



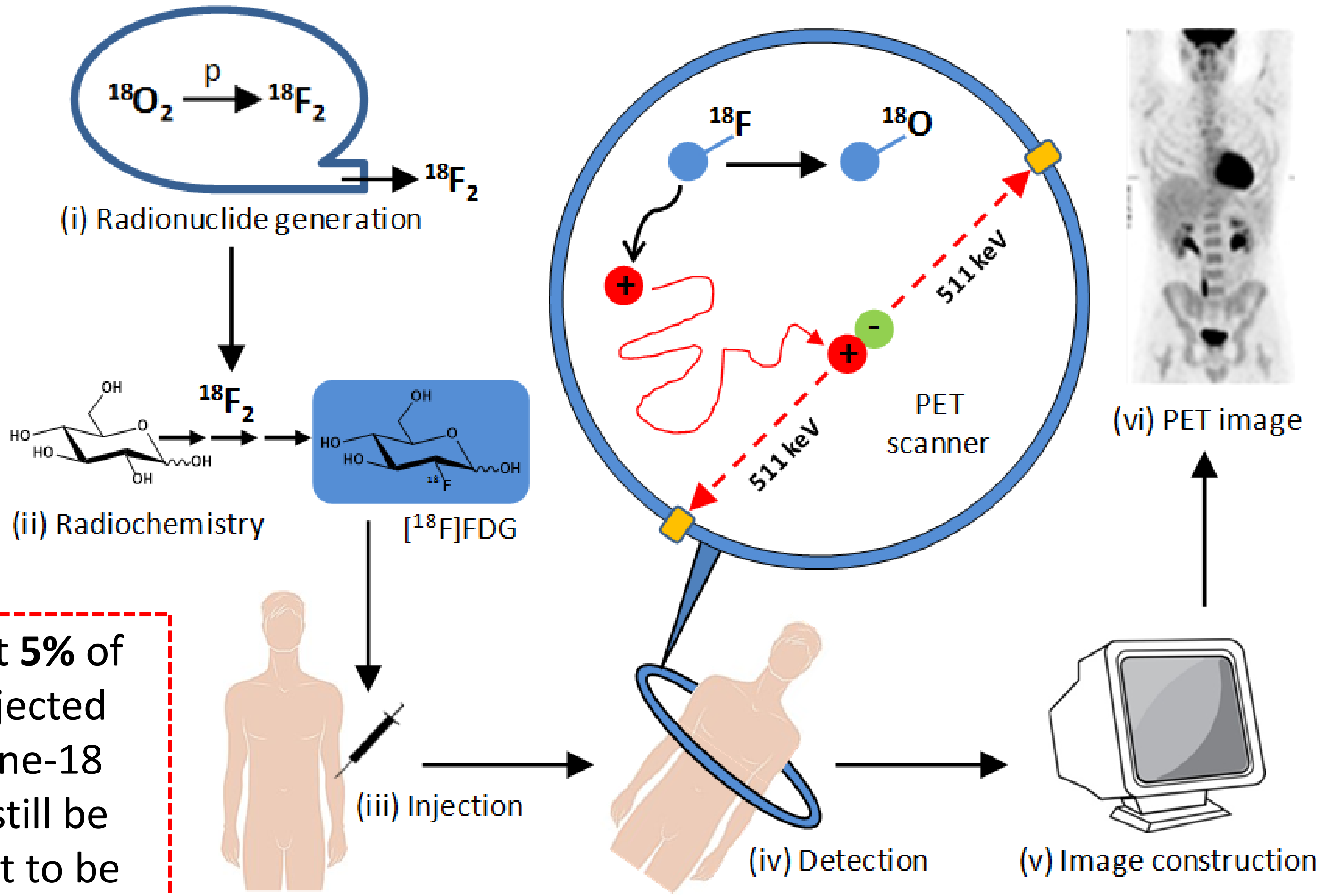


2-Deoxy-D-Glucose (2DG)

- 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



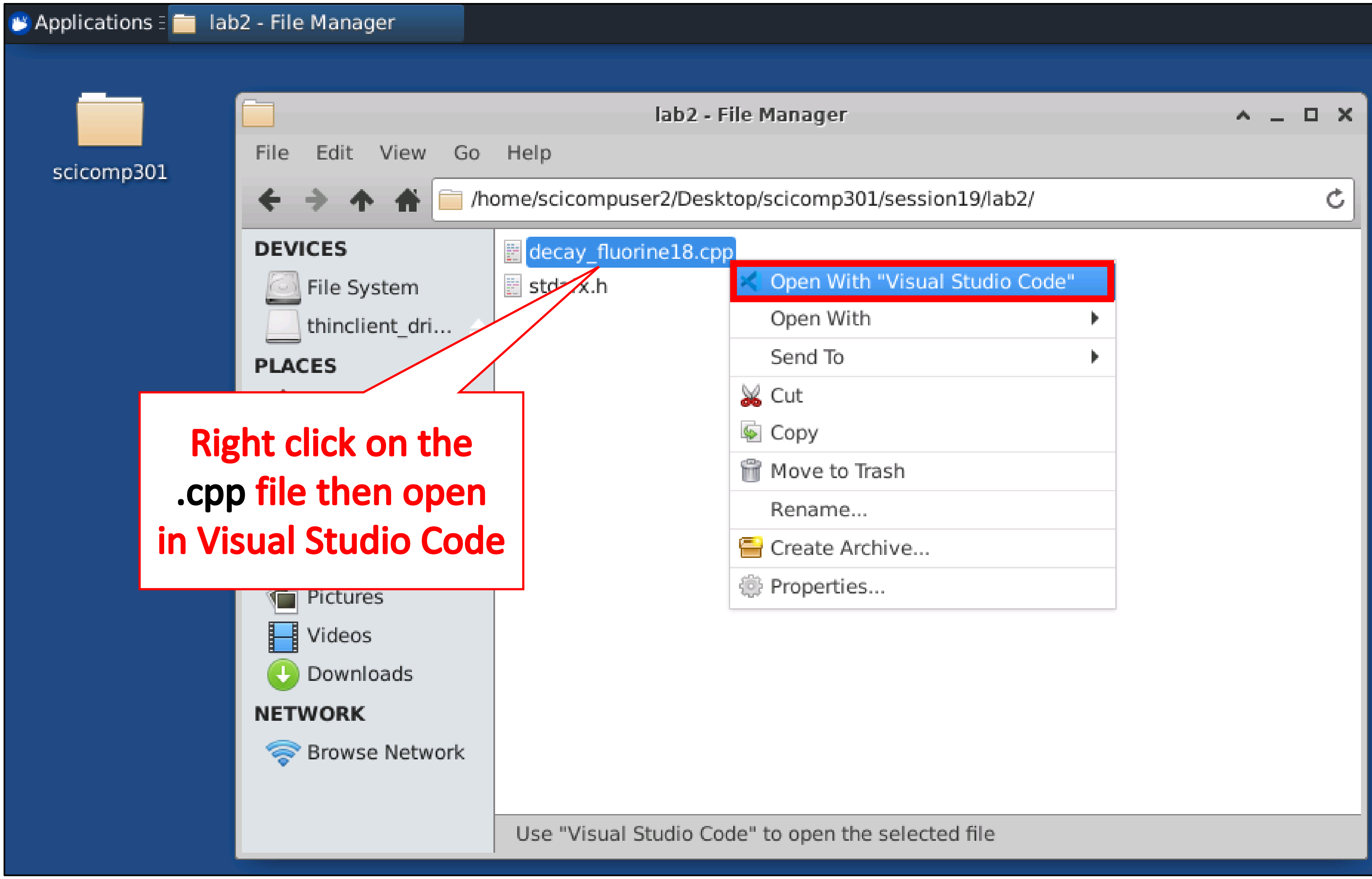
At least **5%** of the injected Flourine-18 must still be present to be detected

Lab 2 – Fluorine-18 Decay

ROOT
Data Analysis Framework

Visual Studio Code

Open Lab 2 – Fluorine-18 Decay



View Lab 2 – Fluorine-18 Decay

decay_fluorine18.cpp

```cpp
#include "stdafx.h"

using namespace std;

void decay_fluorine18()
{
    // Half-life of Fluorine-18 (secs to hours)
    const double halfLife{6586.0 / 60 / 60};

    // Set number of time steps in simulation
    const int timeSteps{100};

    // Duration of simulation (hours)
    const double endTime{12};

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

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

```

ROOT uses underscores as word breakers in file names and the "main" function

View Lab 2 – Fluorine-18 Decay

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

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

```cpp

    // Initialize domain and range vectors
    vector<double> time(timeSteps, 0);
    vector<double> nuclei(timeSteps, 0);

    // Set percent of nuclei initially present
    nuclei.at(0) = 100;

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

```

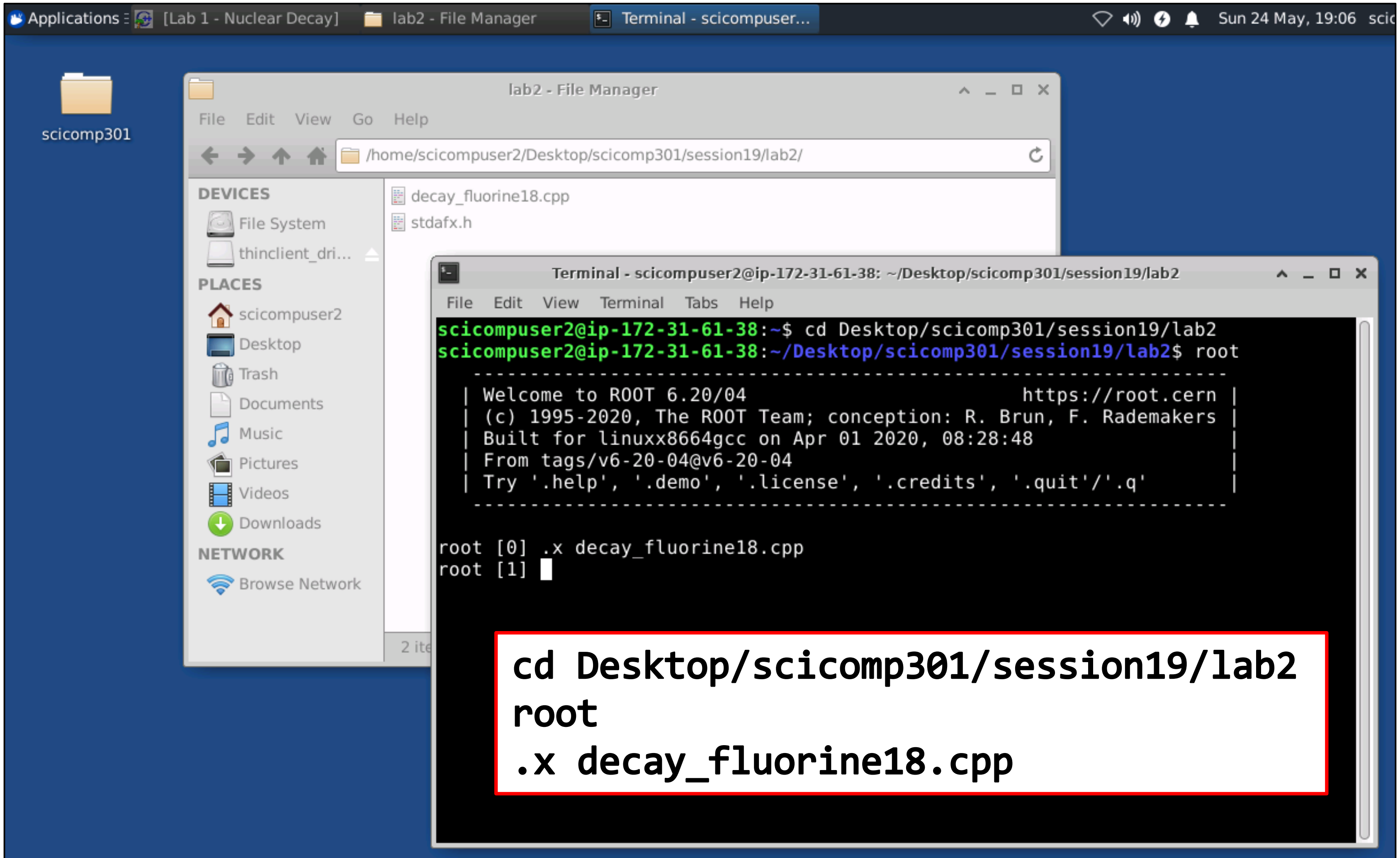
This is Euler's Method

View Lab 2 – Fluorine-18 Decay

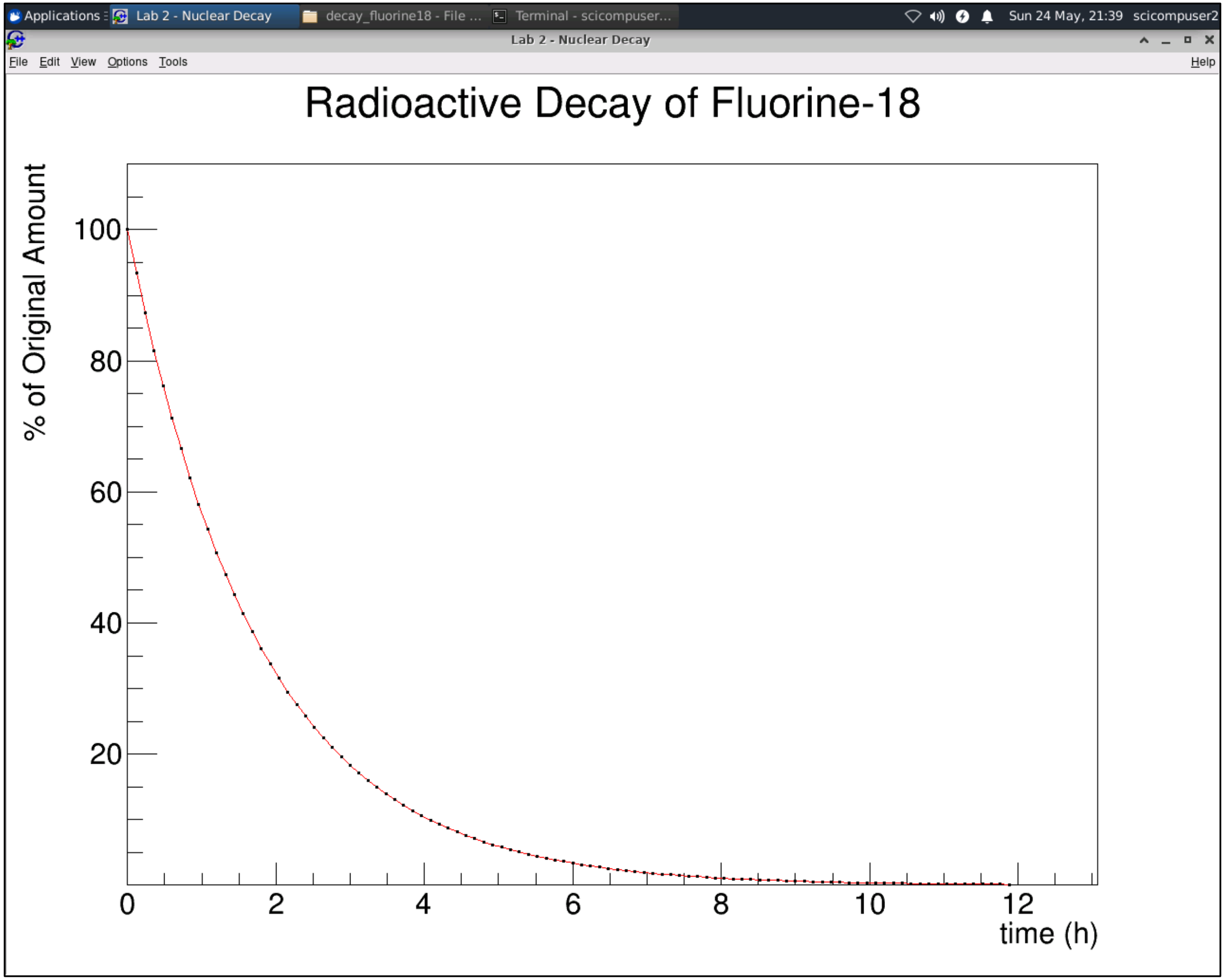
independent *dependent*

```
    // Graph the decay curve using CERN's ROOT libraries
    TCanvas *c1 = new TCanvas("Fluorine-18 Tau Graph");
    c1->SetTitle("Lab 2 - Nuclear Decay");

    TGraph *g1 = new TGraph(timeSteps, time.data(), nuclei.data());

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

```

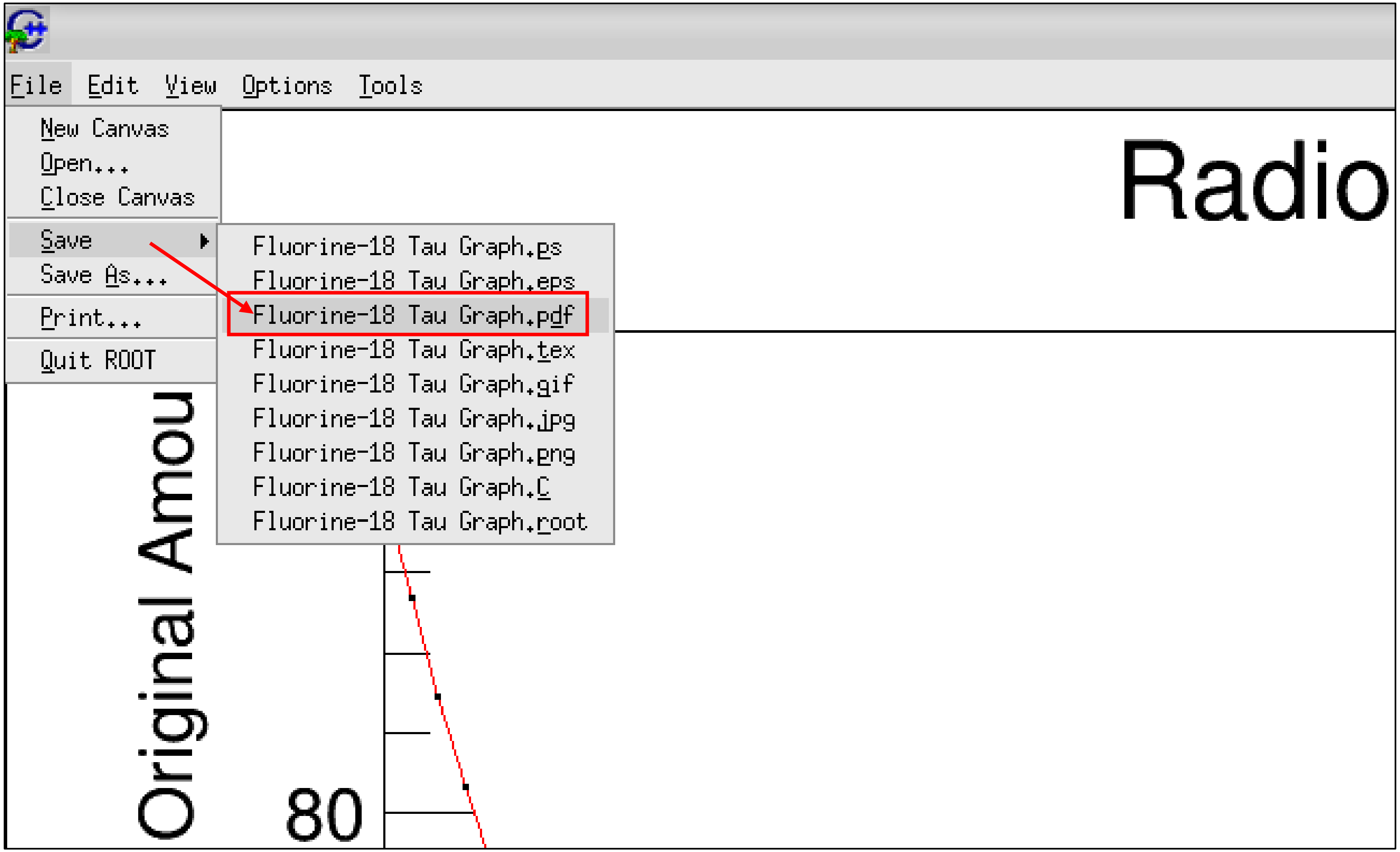
Run Lab 2 – Fluorine-18 Decay



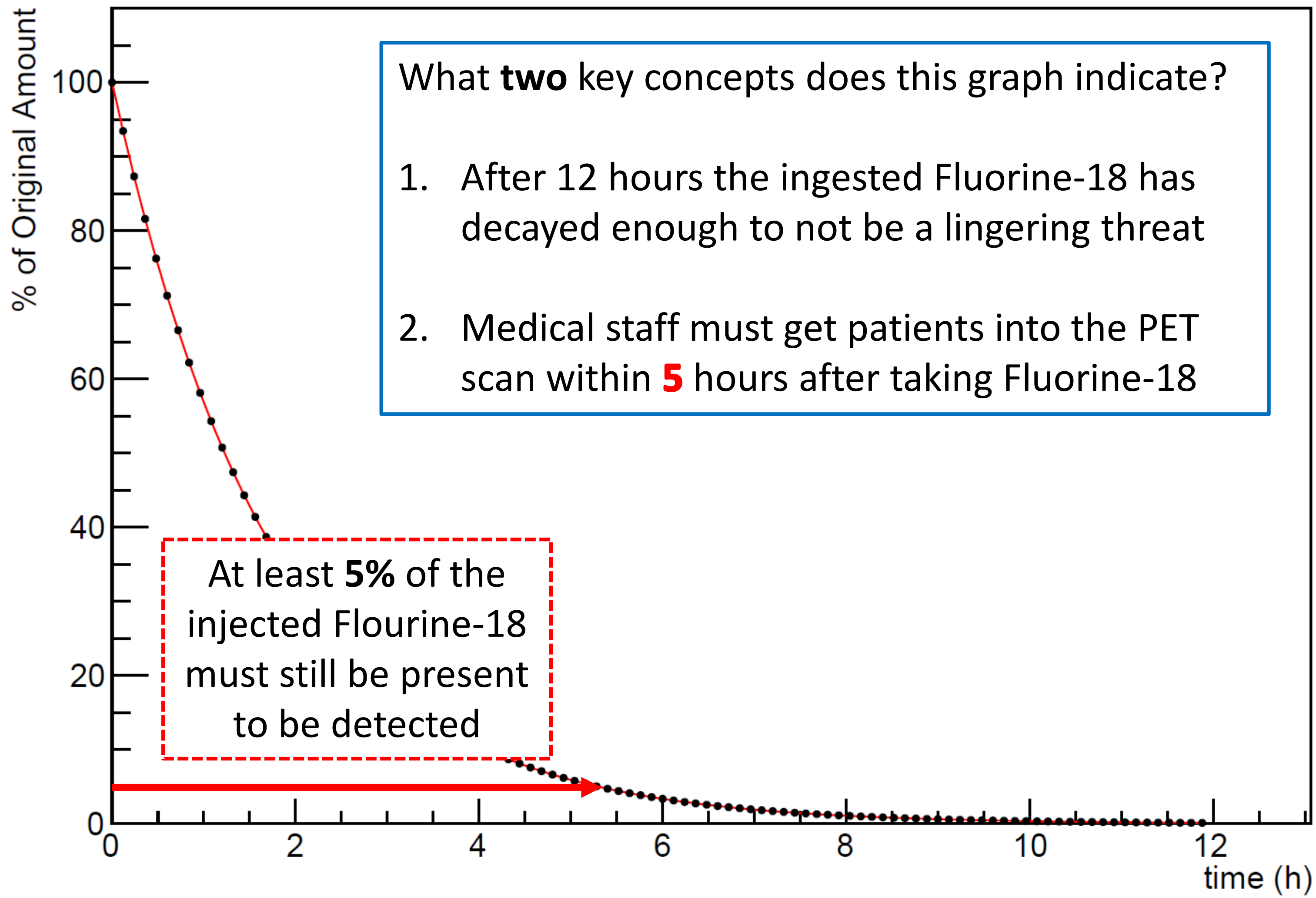
Run Lab 2 – Fluorine-18 Decay



Check Lab 2 – Fluorine-18 Decay



Radioactive Decay of Fluorine-18

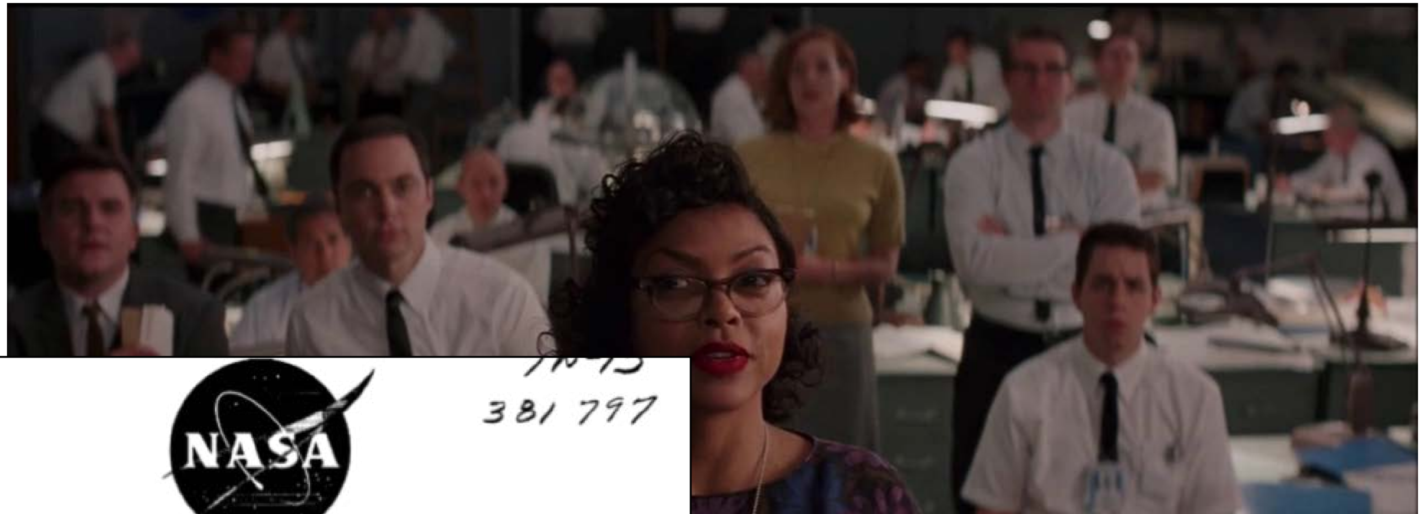


What **two** key concepts does this graph indicate?

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 **5** hours after taking Fluorine-18

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

Hidden Figures, Fox 2000 Pictures, 2016



381 797

TECHNICAL NO

D - 233

DETERMINATION OF AZIMUTH ANGLE AT BURN

SATELLITE OVER A SELECTED EART

By T. H. Skopinski and Katherine G.

Langley Research Center
Langley Field, Va.

Time from perigee is expressed as

$$t(\theta) = \frac{T}{2\pi}(E - e \sin E) \qquad (8)$$

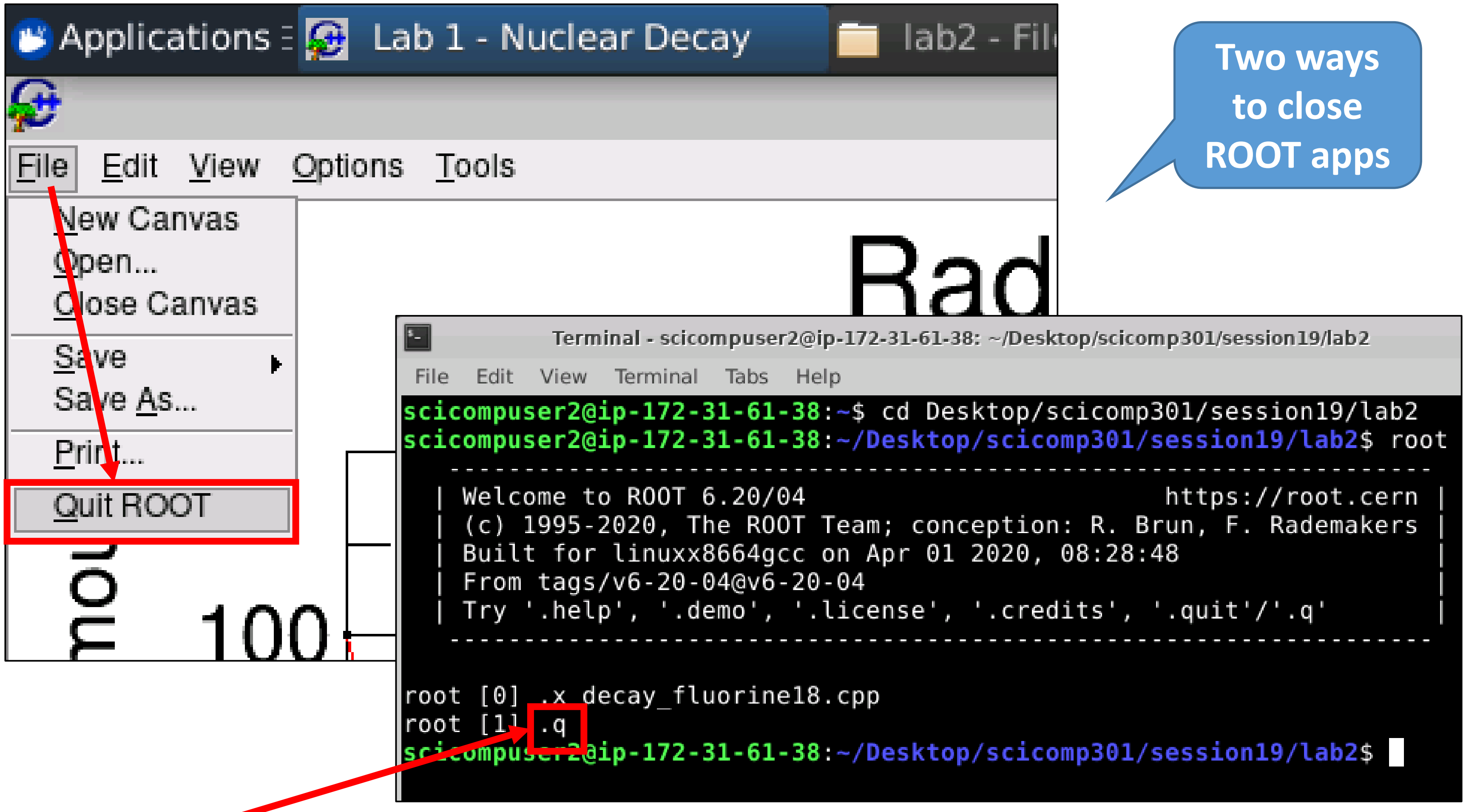
Eccentric anomaly (fig. 1(b)) is given by

$$E = 2 \tan^{-1}\left(\sqrt{\frac{1 - e}{1 + e}} \tan \frac{\theta}{2}\right)$$

Euler's Method

In the use of equations (19) and (20) an iterative procedure is required, since the time $t(\theta_{2e})$ from perigee to the equivalent position is not known initially. A satisfactory first approximation is to assume that

Quit Lab 2 – Fluorine-18 Decay



Two ways to close ROOT apps

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?

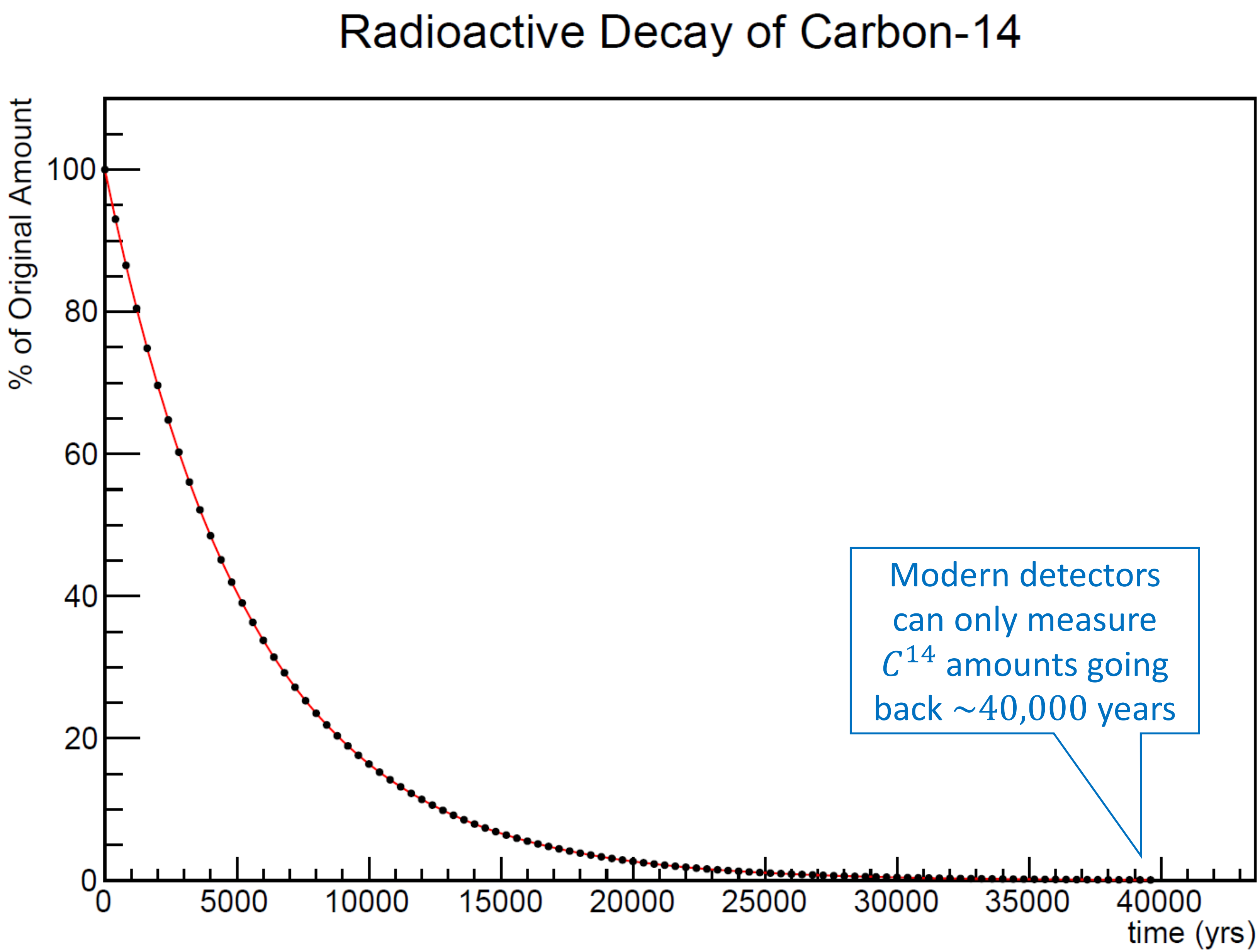
Edit Lab 3 – Modelling Carbon-14 Decay

decay_carbon14.cpp ×

E: > DaveB > Repos > scicomp-labs-cpp > decay_carbon14 > decay_carbon14.cpp > ...

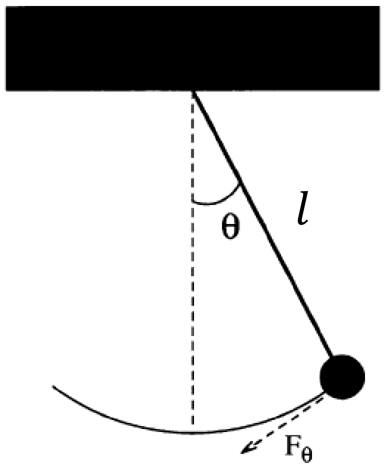
```cpp
#include "stdafx.h"

using namespace std;

void decay_carbon14()
{
    // Half-life of Carbon-14 (years)
    const double halfLife{1};

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

    // Set number of time steps in simulation
    const int timeSteps{100};

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

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

```

Provide correct values for these two constants



Radioactive Decay of Carbon-14
% of Original Amount
0
20
40
60
80
100
0
5000
10000
15000
20000
25000
30000
35000
40000
time (yrs)
Modern detectors can only measure C^{14} amounts going back $\sim 40{,}000$ years

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! ☹

$$s = l\theta$$

$$F = ma$$

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

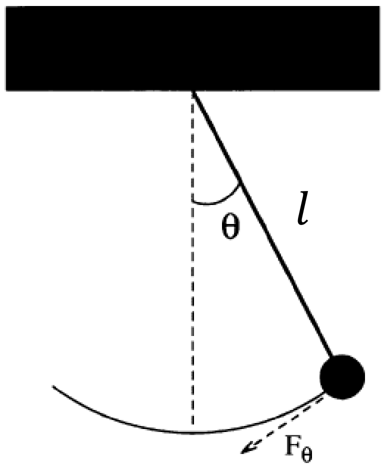
$$F = m\frac{d^2s}{dt^2}$$

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

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

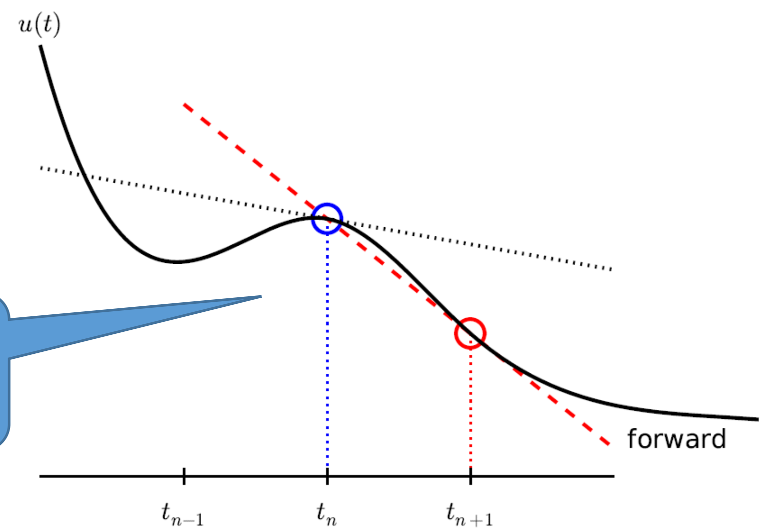
We can break the 2nd order ODE into two linked first order ODEs and use **Euler's method** on each

Modelling a Simple Pendulum



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

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



This is Euler's Method

Open Lab 4 – Simple Pendulum

```
void pendulum()
{
    const double length = 1.0; // (m)
    const double g = 9.8;      // (m/s^2)
    const double phaseConstant = g / length;

    // Set number of time steps in simulation
    const int timeSteps{250};

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

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

    vector<double> time(timeSteps,0);
    vector<double> omega(timeSteps,0);
    vector<double> theta(timeSteps,0);

```

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

$$\frac{d\theta}{dt} = \omega$$

View Lab 4 – Simple Pendulum

```cpp
        // Set initial pendulum angular velocity
        omega.at(0) = 0.0;

        // Set initial pendulum displacement
        theta.at(0) = M_PI / 18.0;

        // Perform Euler method to estimate differential equation
        for (int step{}; step < timeSteps - 1; ++step)
        {
            omega.at(step + 1) = omega.at(step) - phaseConstant * theta.at(step) * deltaTime;
            theta.at(step + 1) = theta.at(step) + omega.at(step) * deltaTime;
            time.at(step + 1) = time.at(step) + deltaTime;
        }

```

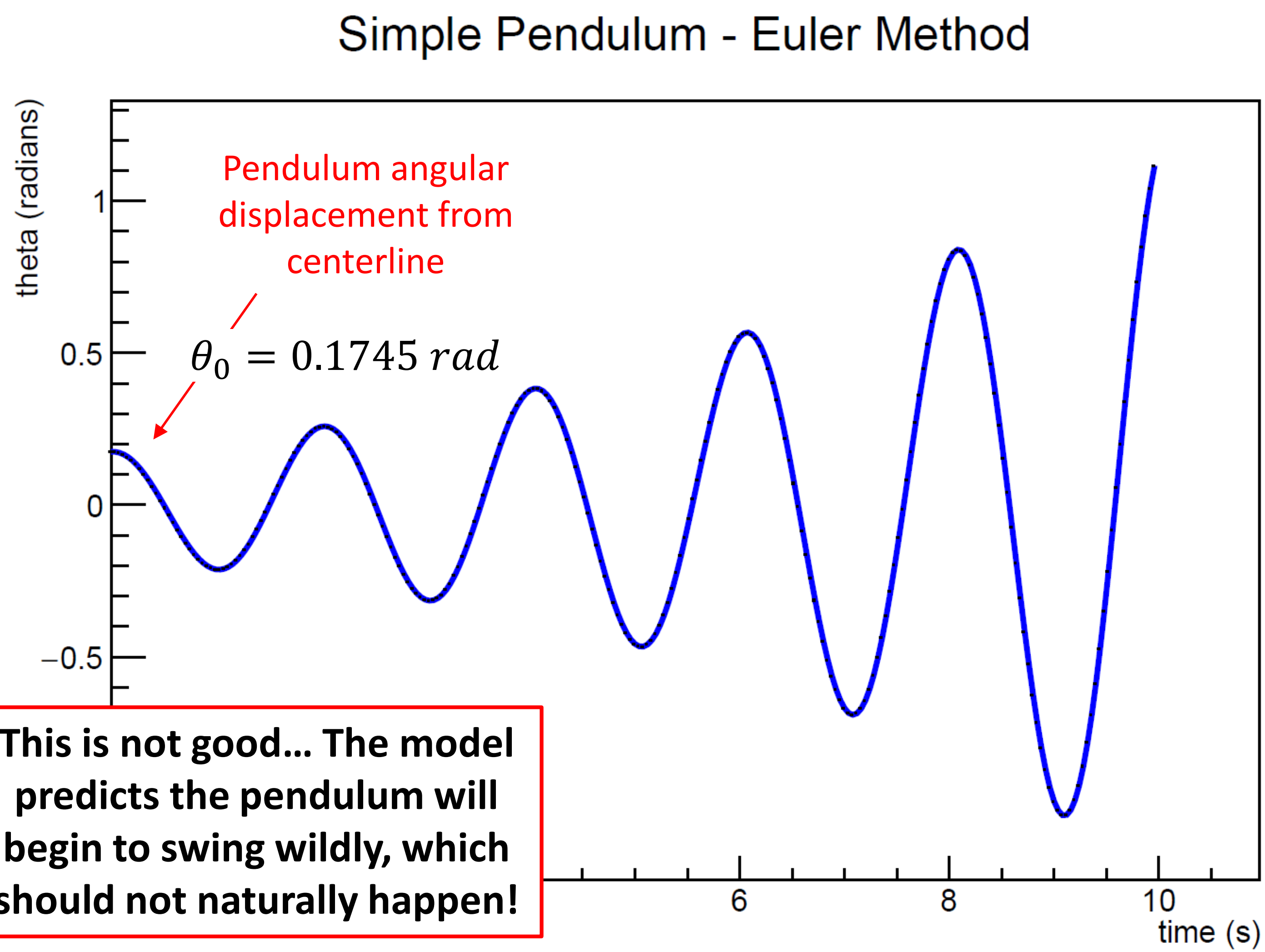
$$10^{\circ} = 10 \times \frac{2\pi}{360^{\circ}} = \frac{\pi}{18} = 0.1745\, rad$$

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

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

ROOT
Data Analysis Framework

Run 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

kinetic **potential**

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

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

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

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

Due to the squaring of both variables, the next energy state will always be larger than the previous energy state

The Euler-Cromer Method

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

< PREVIOUS ARTICLE | TABLE OF CONTENTS | NEXT ARTICLE >

Abstract | Full Text | References (0) | Cited By (30) | Data & Media | Metrics | Related

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.

Edit Lab 4 – Harmonic Motion

Add the **+1** to the term in line #34

Euler

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

Euler-Cromer

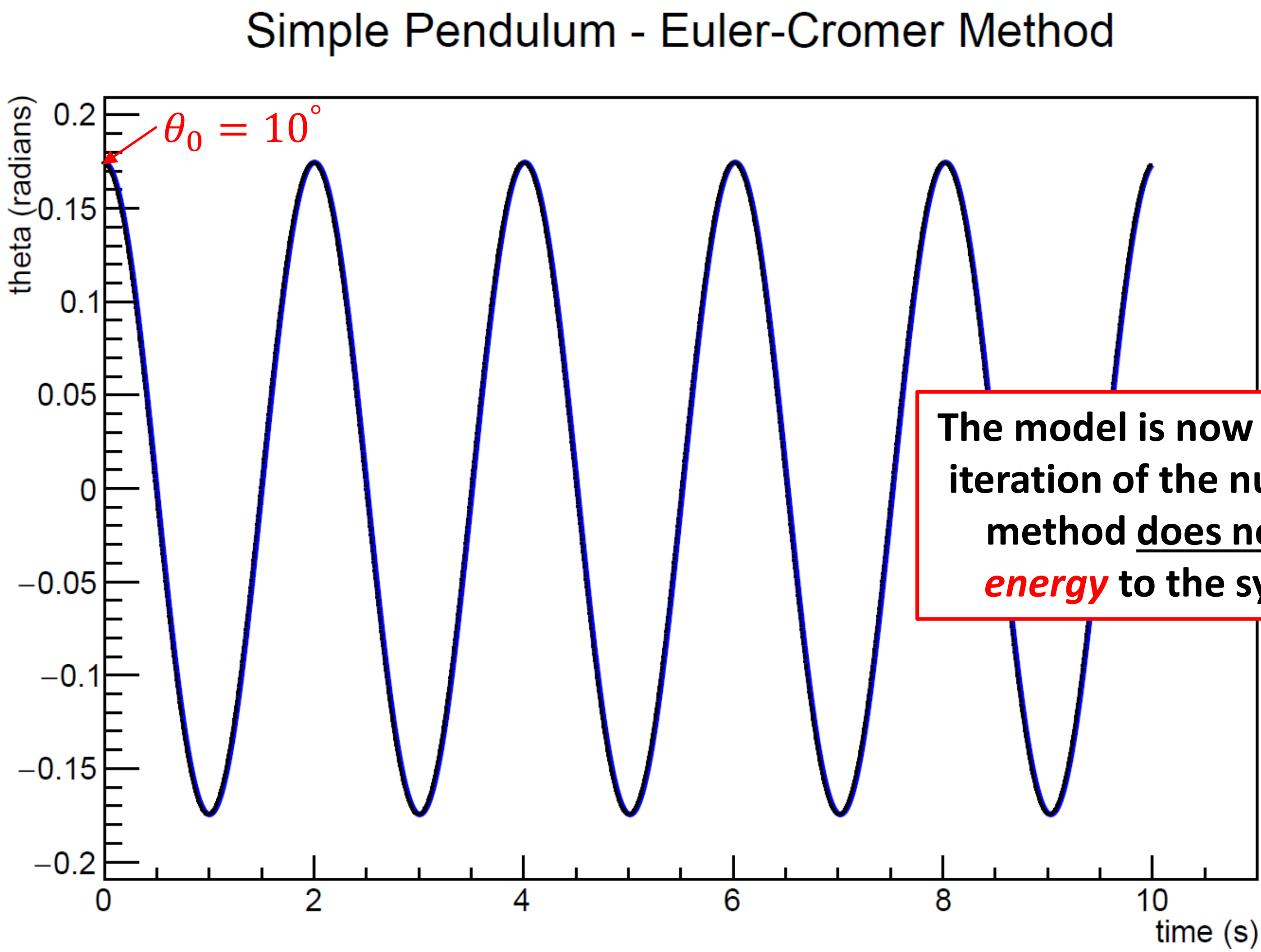
$$\omega_{i+1} = \omega_i - \frac{g}{l}\theta_i \Delta t$$
$$\theta_{i+1} = \theta_i + \omega_{i\boxed{+1}} \Delta t$$

```
30        // Perform Euler method to estimate differential equation
31        for (int step{}; step < timeSteps - 1; ++step)
32        {
33            omega.at(step + 1) = omega.at(step) - phaseConstant * theta.at(step) * deltaTime;
34            theta.at(step + 1) = theta.at(step) + omega.at(step + 1) * deltaTime;
35            time.at(step + 1) = time.at(step) + deltaTime;
36        }
```

Mr. Cromer was also the author of several widely-used textbooks, including "Physics for the Life 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.

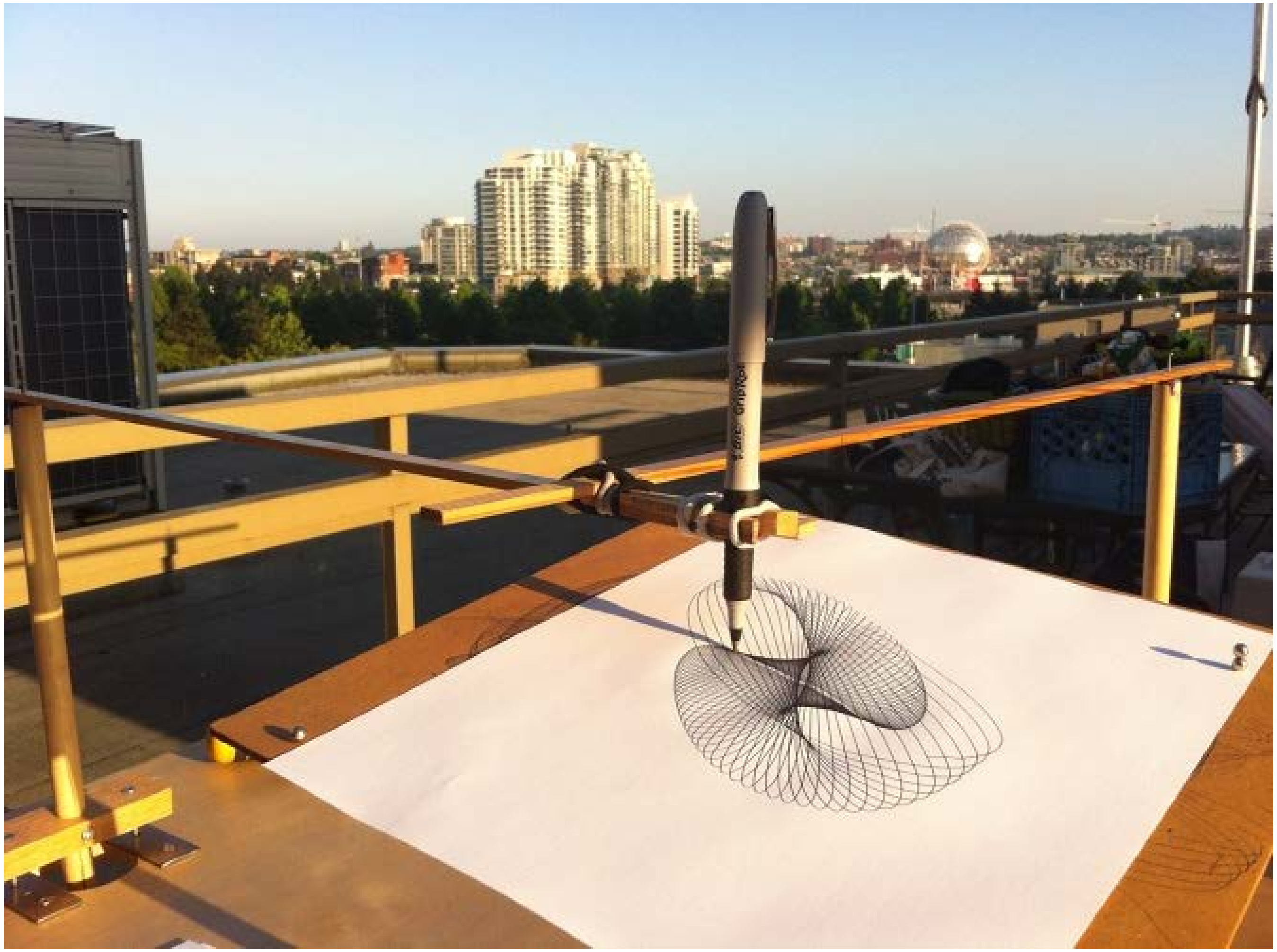
"I think he was one of the first people to recognize the importance of having a text book for biological physics," Nath said.

Run Lab 4 – Harmonic Motion

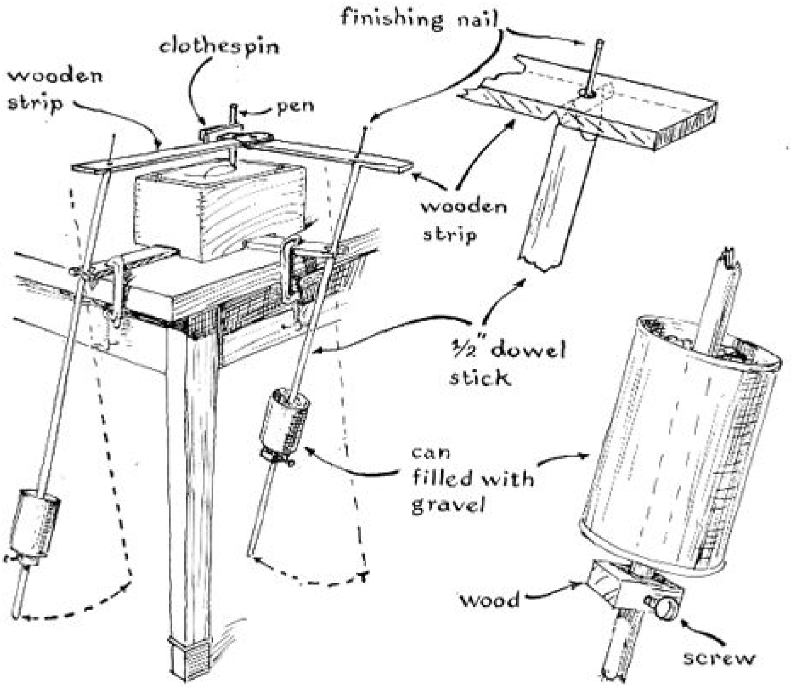


The model is now stable as iteration of the numerical method <u>does not</u> add *energy* to the system.

Coupled Harmonograph



Coupled Harmonograph



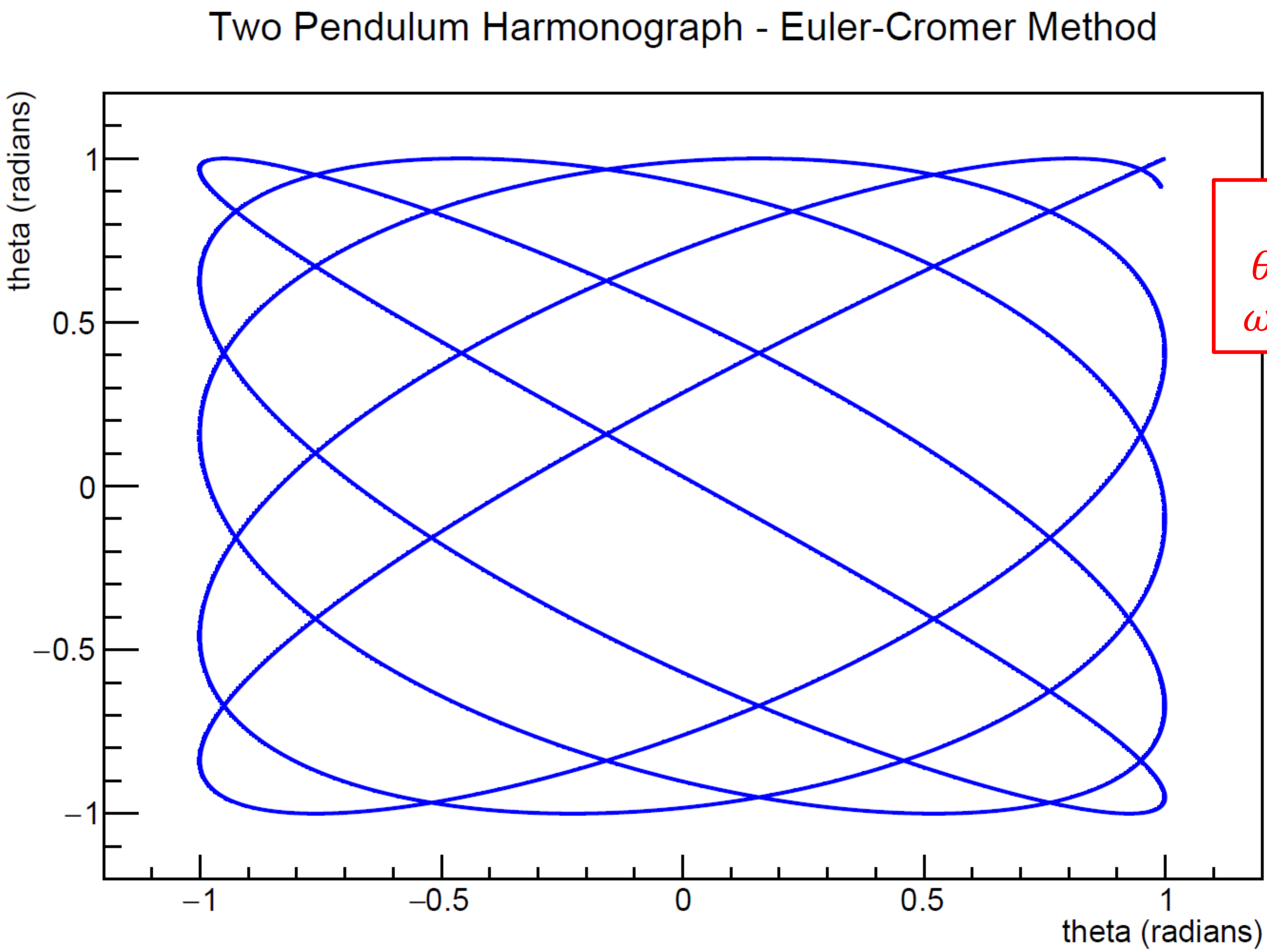
Open Lab 5 – Coupled Harmonograph

```
void harmonograph()
{
    const double g = 9.8; // (m/s^2)

    const int timeSteps{2500};
    const double endTime{10};
    const double deltaTime{endTime / timeSteps};

    vector<double> time(timeSteps, 0);
    vector<double> omega1(timeSteps, 0);
    vector<double> theta1(timeSteps, 0);
    vector<double> omega2(timeSteps, 0);
    vector<double> theta2(timeSteps, 0);

    // Set first pendulum initial conditions
    const double length1 = 1.0; // (m)
    theta1.at(0) = M_PI / 18.0; // (~10 degrees)
    omega1.at(0) = 0.0;         // (rads/s)

    // Set second pendulum initial conditions
    const double length2 = 1.5; // (m)
    theta2.at(0) = M_PI / 18.0; // (~10 degrees)
    omega2.at(0) = 0.0;         // (rads/s)

    const double phaseConstant1 = g / length1;
    const double phaseConstant2 = g / length2;

```

View Lab 5 – Coupled Harmonograph

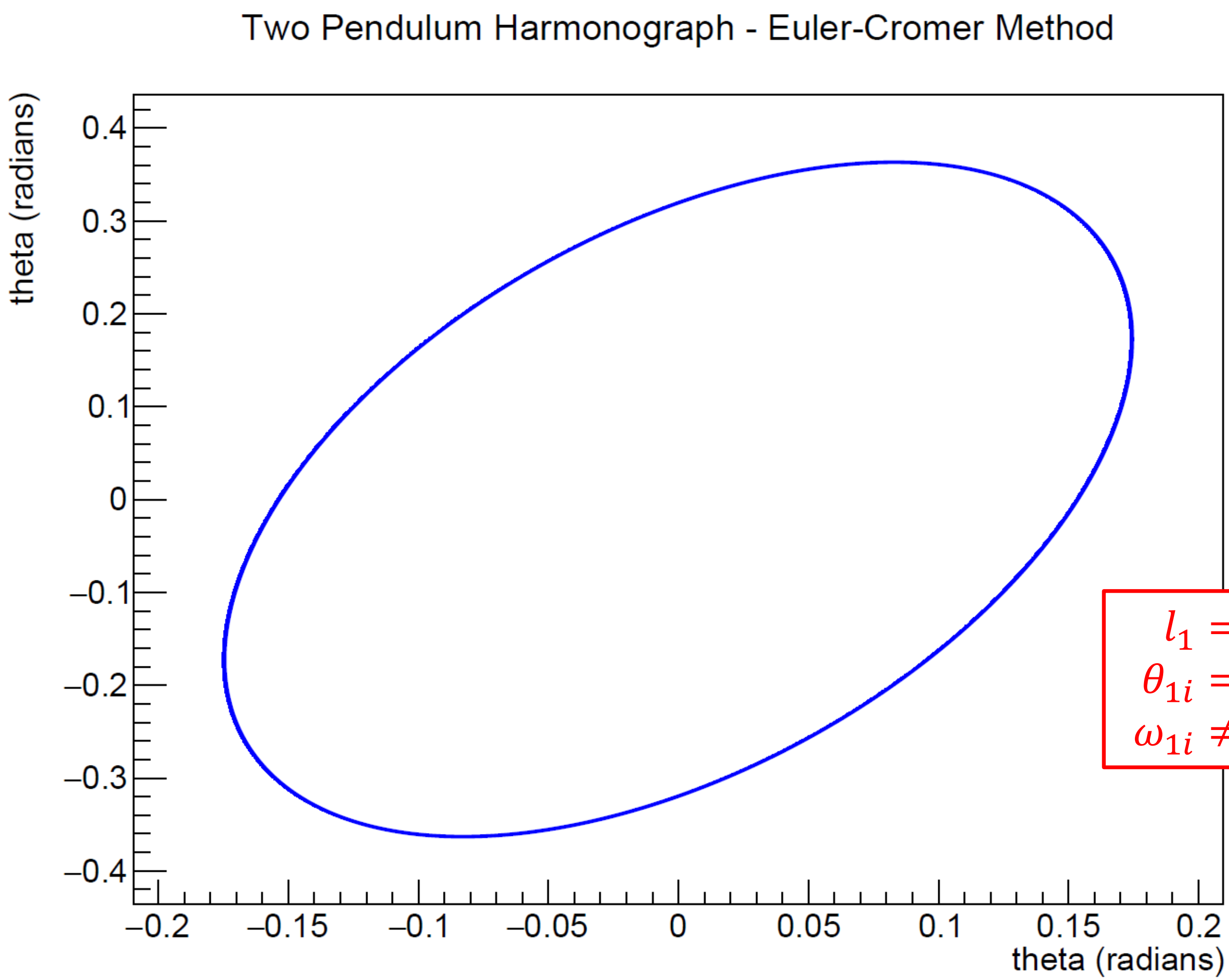
```
    // Perform Euler-Cromer method to estimate differential equation
    for (int step{}; step < timeSteps - 1; ++step)
    {
        // First pendulum
        omega1.at(step + 1) = omega1.at(step) - phaseConstant1 * theta1.at(step) * deltaTime;
        theta1.at(step + 1) = theta1.at(step) + omega1.at(step + 1) * deltaTime;
        // Second pendulum
        omega2.at(step + 1) = omega2.at(step) - phaseConstant2 * theta2.at(step) * deltaTime;
        theta2.at(step + 1) = theta2.at(step) + omega2.at(step + 1) * deltaTime;
        // Update time
        time.at(step + 1) = time.at(step) + deltaTime;
    }

    // Graph the decay curve using CERN's ROOT libraries
    TCanvas *c1 = new TCanvas("Two Pendulum Harmonograph");
    c1->SetTitle("Two Pendulum Harmonograph - Euler-Cromer Method");

    TGraph *g1 = new TGraph(timeSteps, theta1.data(), theta2.data());

```

Run Lab 5 – Coupled Harmonograph



$l_1 \neq l_2$
$\theta_{1i} = \theta_{2i}$
$\omega_{1i} = \omega_{2i}$

Edit Lab 5 – Coupled Harmonograph



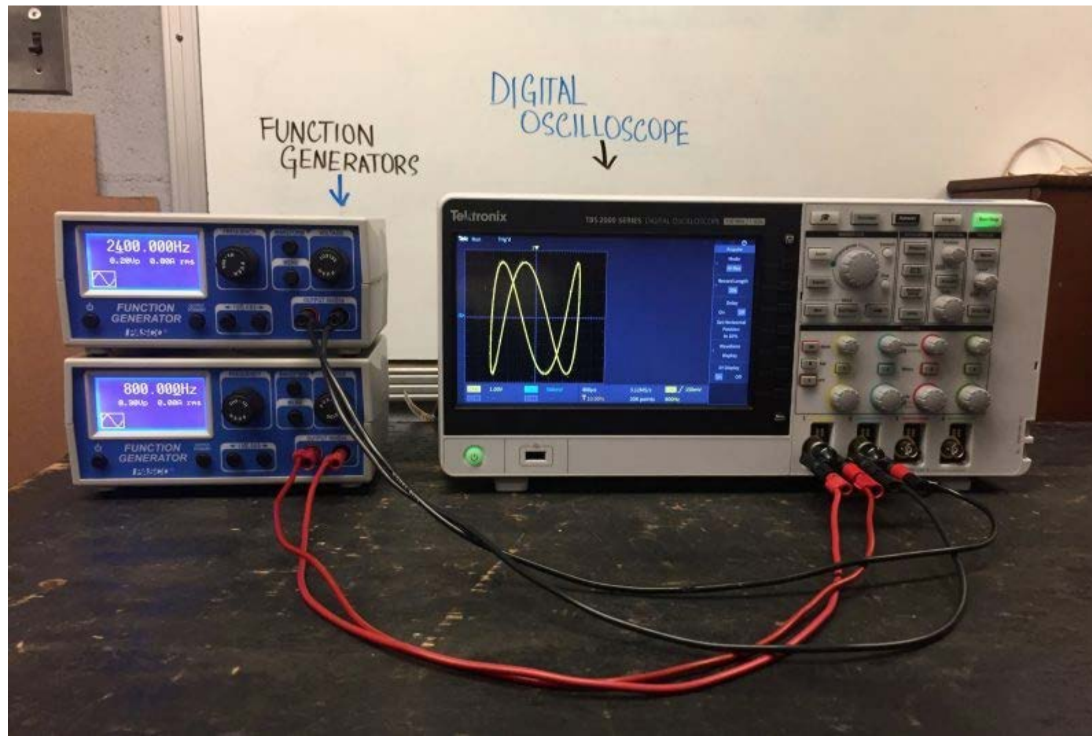
$l_1 = l_2$
$\theta_{1i} = \theta_{2i}$
$\omega_{1i} \neq \omega_{2i}$

Lissajous Figures

Jules Antoine Lissajous

Jules Antoine Lissajous, date and photographer unknown

Born	March 4, 1822 Versailles, France
Died	June 24, 1880 (aged 58) Plombières-les-Dijon, France
Known for	Lissajous figures



Lissajous Orbits

DEEP SPACE CLIMATE OBSERVATORY

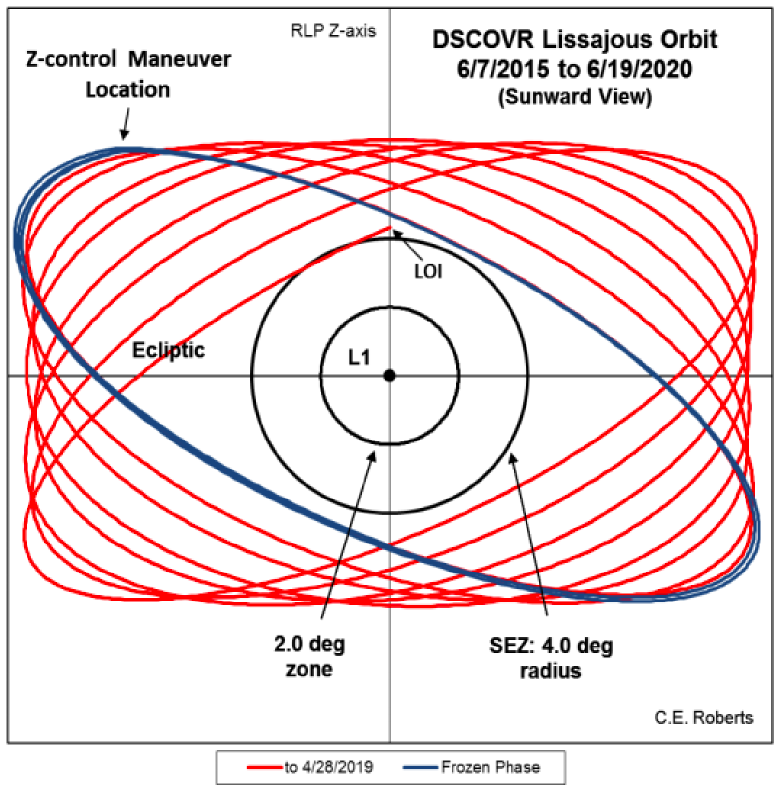
DSCOVR Lissajous with "Frozen Phase" Segment

First Z-control Burn at +Z extremum on 4/28/2019

ΔV_Z negative toward SEP

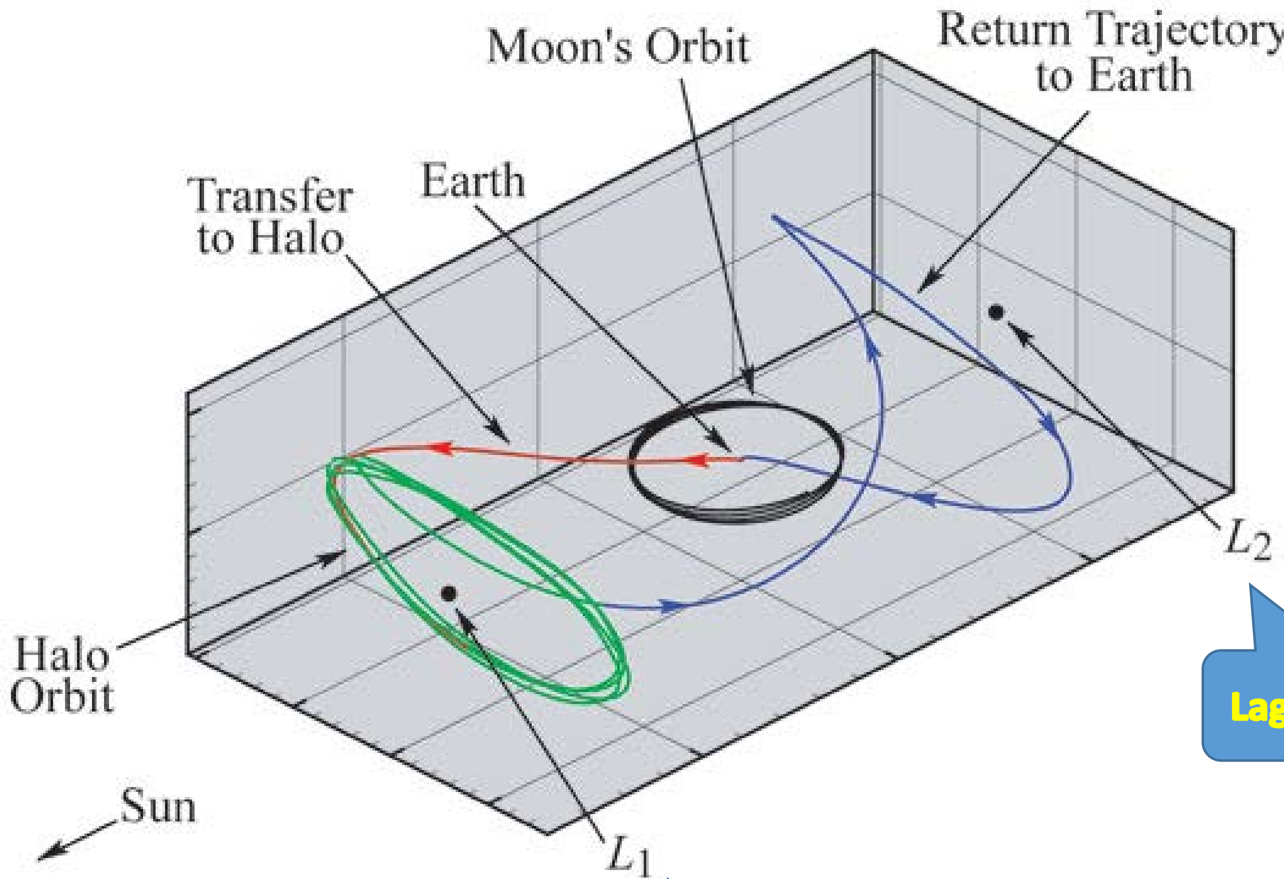
2-stage targeting achieves –Z position then the +Z position at RLP XZ plane

Repeat at each return to +Z extremum



Performed in conjunction with SK ΔV_X for stability

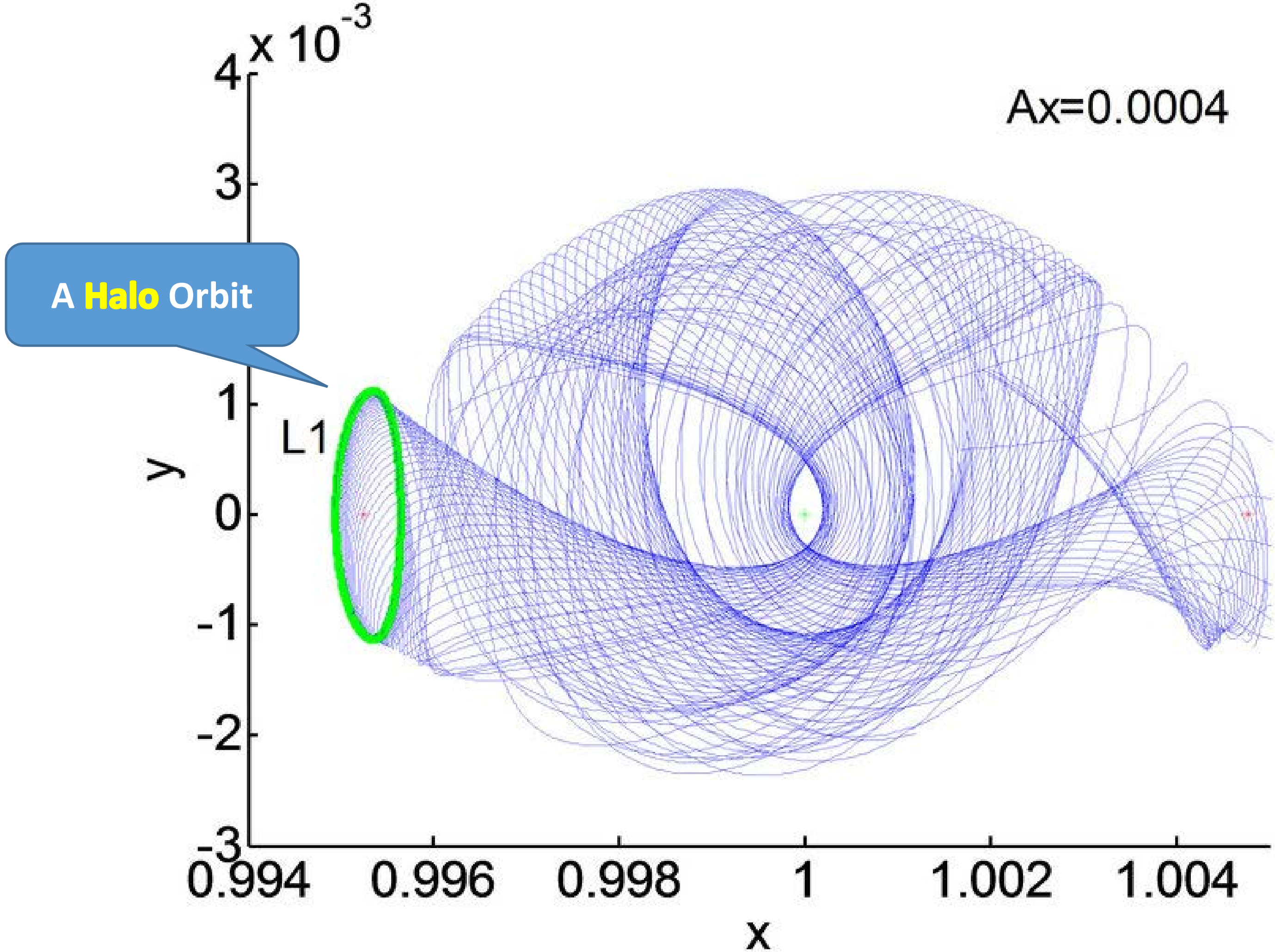
Lissajous Orbits

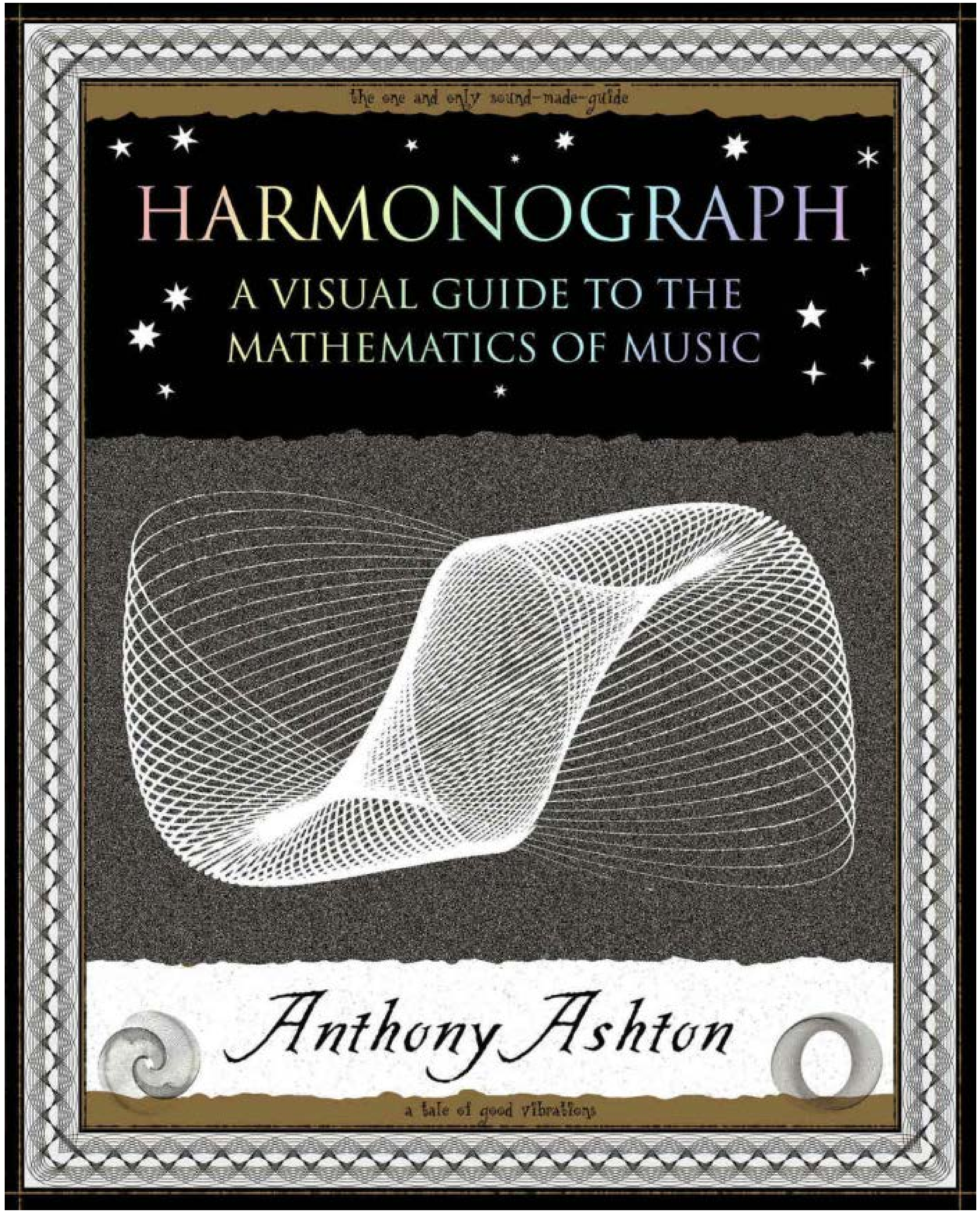


Lagrange Point

Lagrange Point

Lissajous Orbits




the one and only sound-made-guide
HARMONOGRAPH
A VISUAL GUIDE TO THE
MATHEMATICS OF MUSIC
Anthony Ashton
a tale of good vibrations

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 2^{nd} order differentials by representing them as a **chain of linked 1^{st} 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