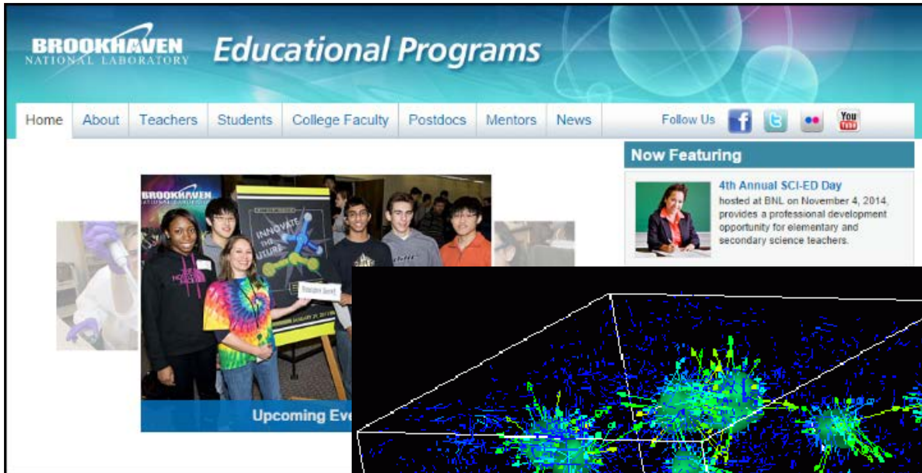
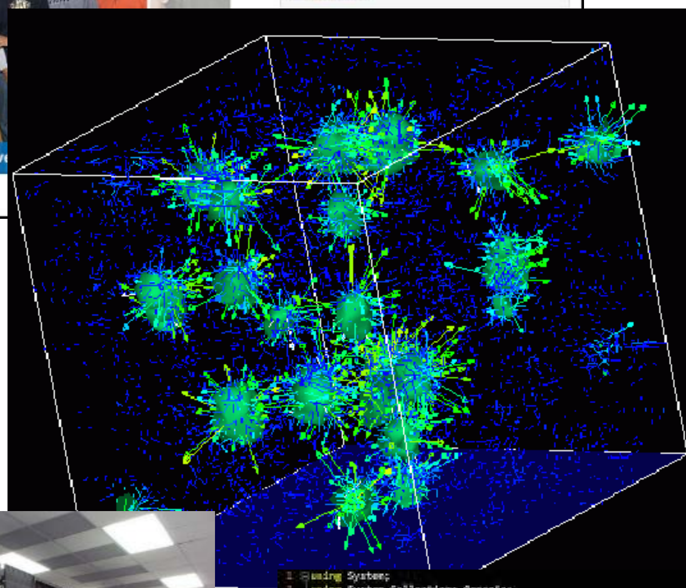
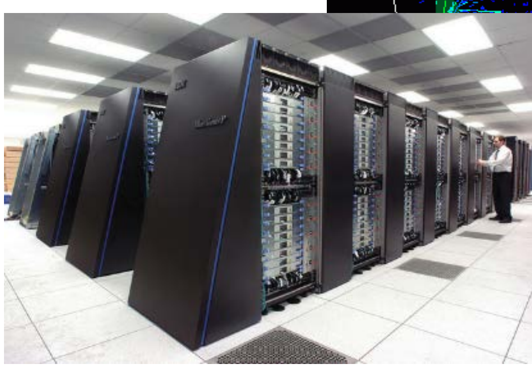



Survey of Scientific Computing
(SciComp 301)

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

Exam 3
Total of 100 points

10 pts

1. Predator-Prey Modelling

https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations

In the **q01** folder, edit the **C++ CERN ROOT** application to visualize the **Lotka-Volterra** (1920) differential equations with given characteristics & initial conditions

Fig. 1.1 – Alfred Lotka

$$\boxed{\begin{array}{c}\alpha = 2, \beta = 1.1, \gamma = 1.0, \delta = 0.9 \\ x(0) = 1, y(0) = 0.5\end{array}}$$

In this model, at any time $\boldsymbol{t}$:
$x(t)$ represents the **prey** population
$y(t)$ represents **predator** population

Fig. 1.2 – Vito Volterra

The solution to their system of *coupled* non-linear first order differential equations will be numerically estimated using the **4th order Runge-Kutta** method

1. Predator-Prey Modelling

```
// Lotka-Volterra {Prey} dx/dt
double d_prey(double x, double y, double t)
{
    return 0;
}

// Lotka-Volterra {Predator} dy/dt
double d_predator(double x, double y, double t)
{
    return 0;
}

void rk4_lv()
{
    // Initial time
    double t = 0.0;

    // Initial prey population %
    double x = 0.0;

    // Initial predator population %
    double y = 0.0;
```

You must write this function

You must write this function

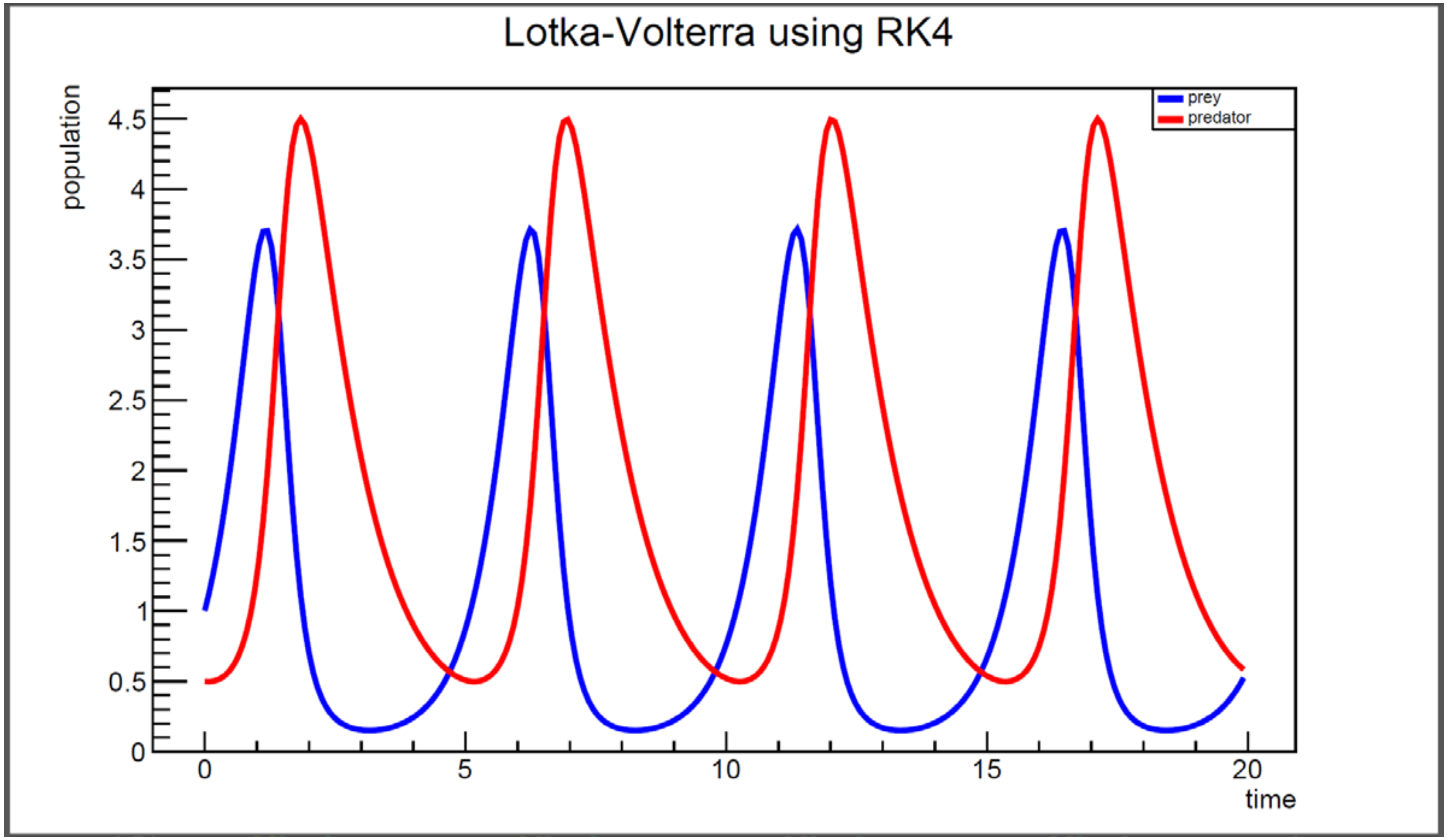
Provide these values

$$\texttt{d\_prey()} = \frac{dx}{dt}$$

$$\texttt{d\_predator()} = \frac{dy}{dt}$$

1. Predator-Prey Modelling

Expected Output (Approved Solution)

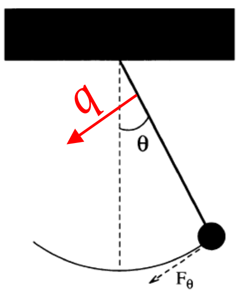


15 pts

2. Damped Pendulum

In the **q02** folder, edit the **C++ CERN ROOT** application to accurately model a pendulum damped with a frictional resistance ***q*** directly *proportional* to its **angular velocity**

Referring to Session 19 Lab 04, we must introduce an additional **resistive force term** into the equation of motion



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

Dampening force constant q

Euler-Cromer Difference Equations

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

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

2. Damped Pendulum

Assume a damping factor $q = 1$

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

Add damping term

```
const double phaseConstant = g / length;
```

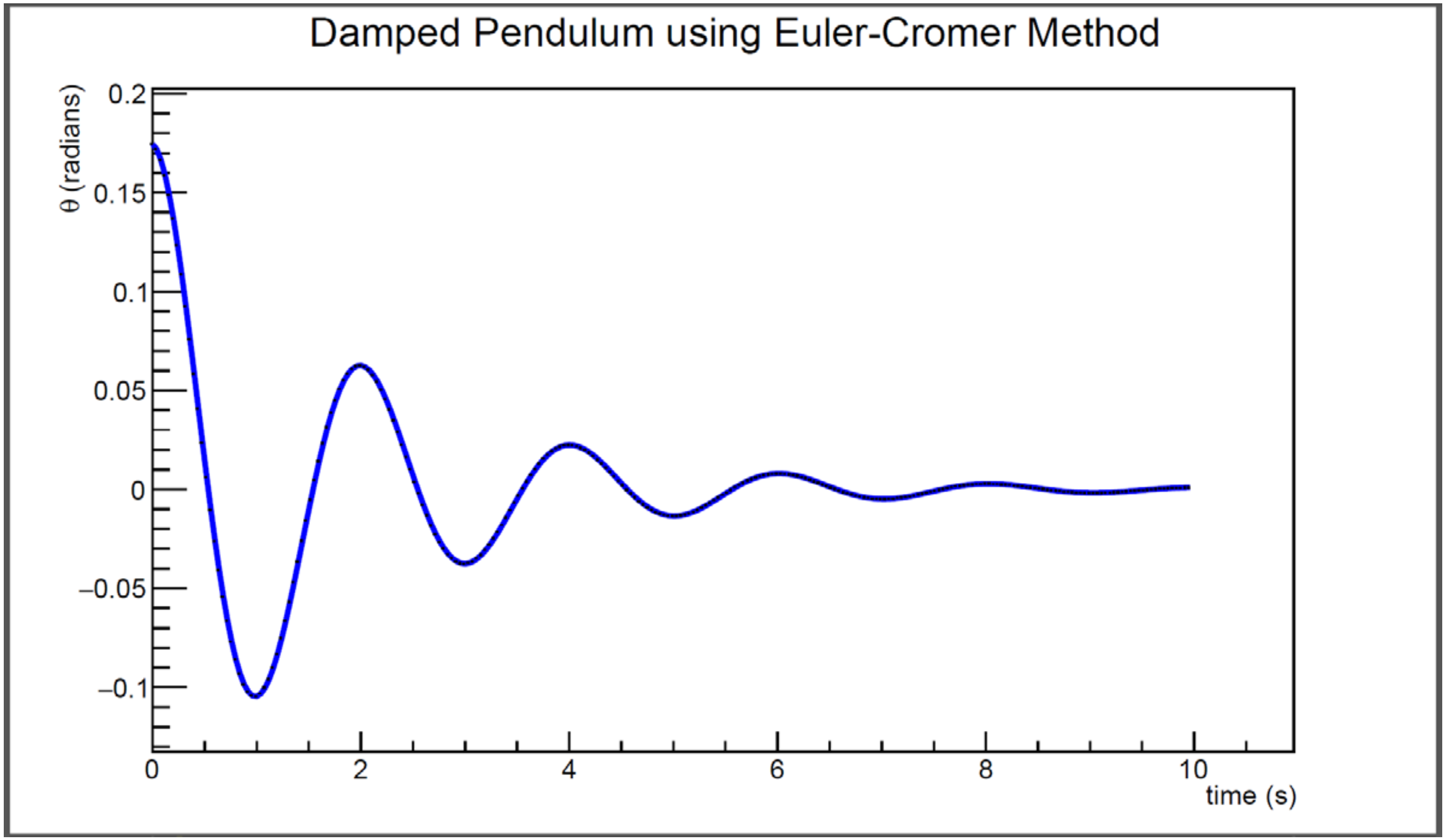
```
// Perform Euler method to estimate differential equation
for (int step{}; step < timeSteps - 1; ++step)
{
    omega[step + 1] = omega[step] - phaseConstant * theta[step] * deltaTime;
    theta[step + 1] = theta[step] + omega[step] * deltaTime;
    timeAt[step + 1] = timeAt[step] + deltaTime;
}
```

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

Insert Cromer's correction

2. Damped Pendulum

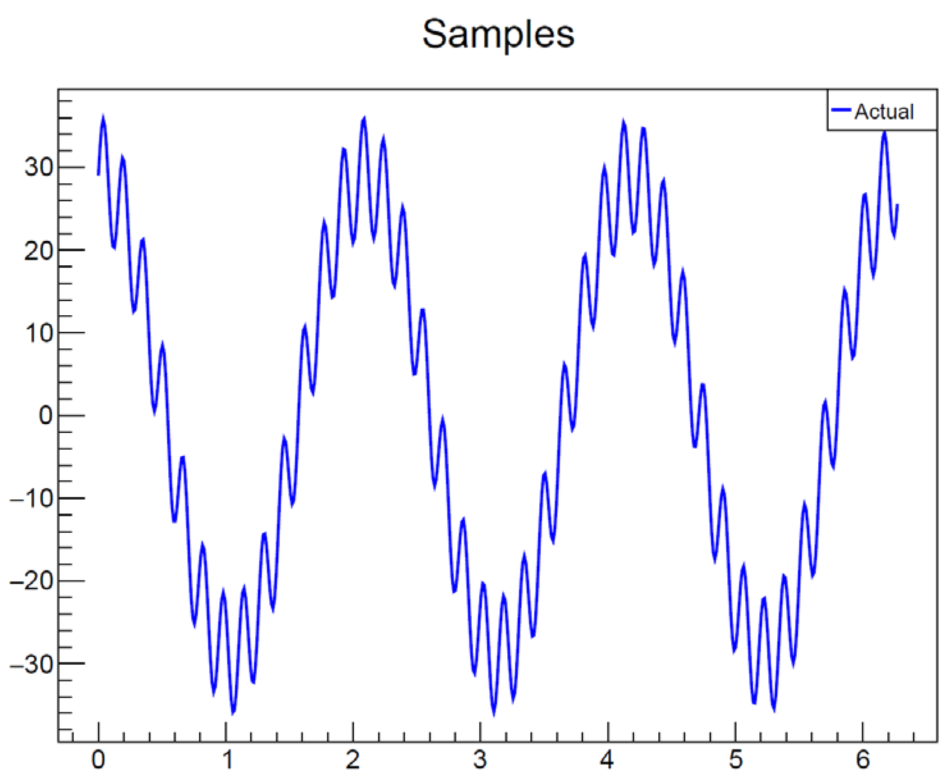
Expected Output (Approved Solution)



15 pts

3. High Frequency Filter

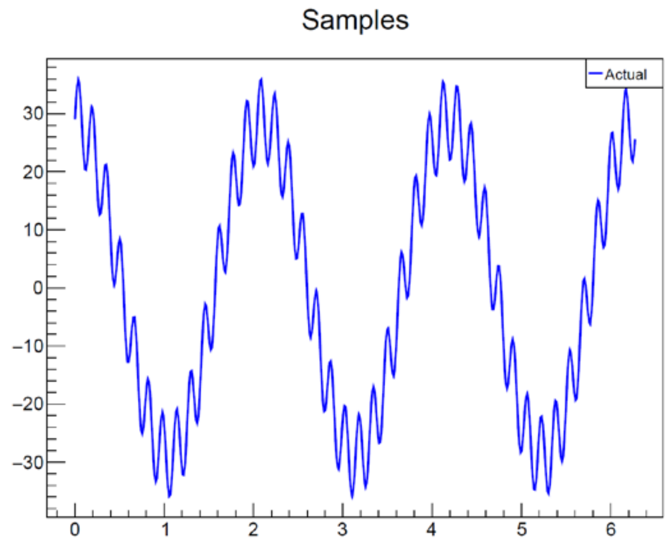
In the **q03** folder, edit the **C++ CERN ROOT** application to filter out the high frequency noise embedded in a signal using the methods learned in **Session 21**



High frequency interference is distorting the capture of this clean primary signal

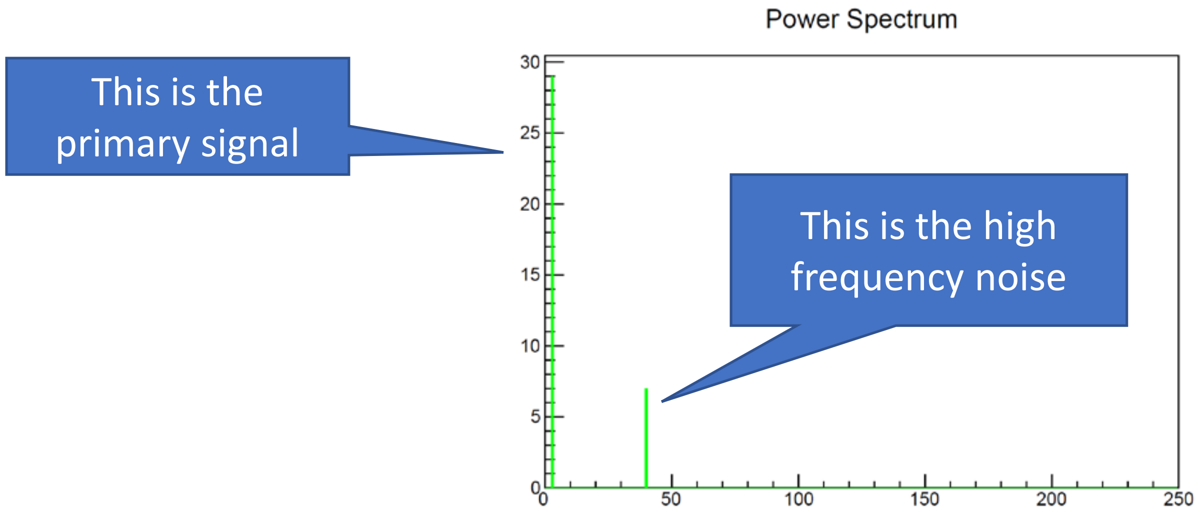
We want to remove this interference before using the inverse discrete Fourier transform (IDFT) to reconstruct the signal

3. High Frequency Filter



The DFT identifies the constituent simple waves which compose a complicated wave

The interference can be filtered out by eliminating the simple waves that have a high frequency



3. High Frequency Filter

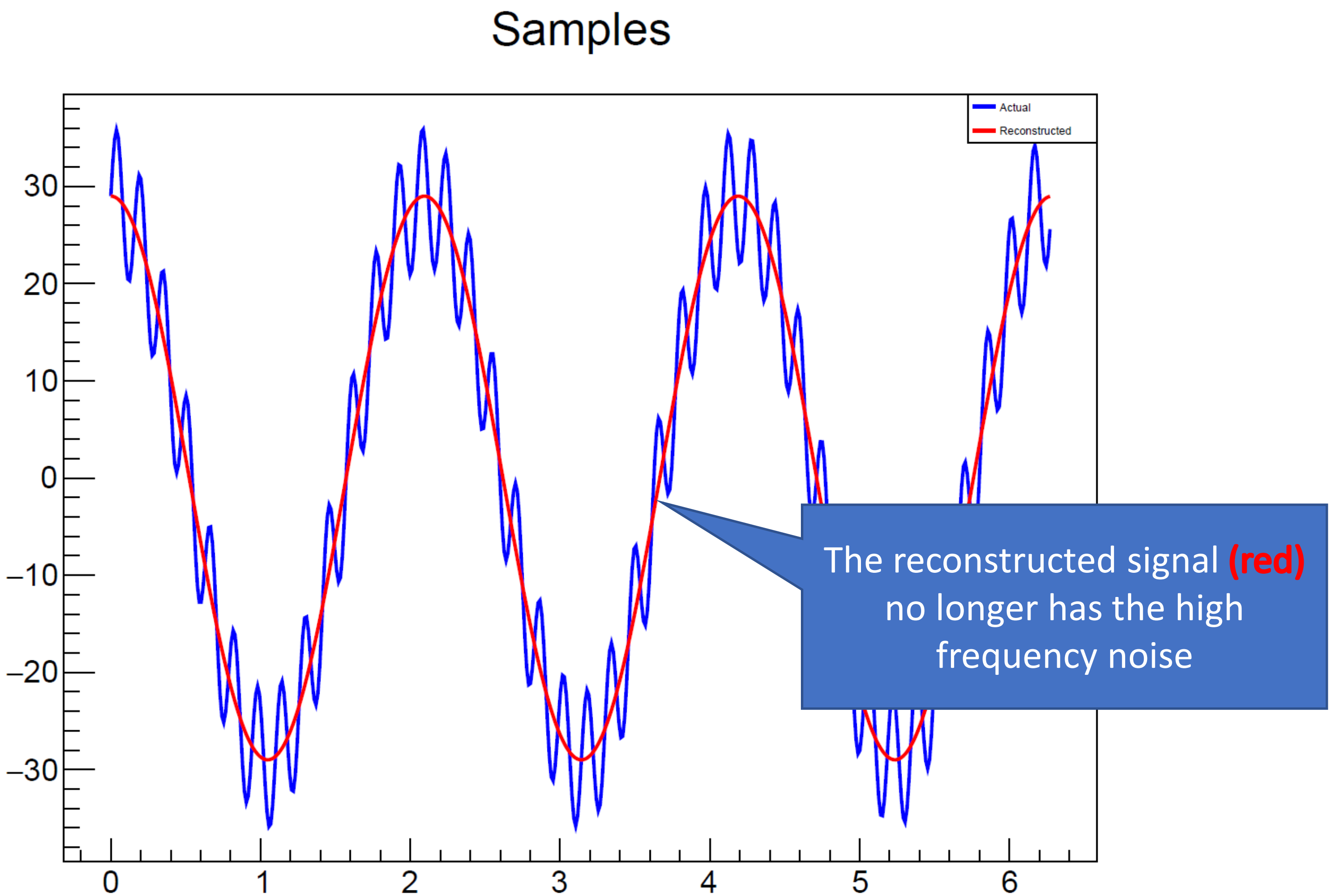
```
void fourier_filter()
{
    InitSamples();

    ScaleDomain();

    CalcDFT();

    ApplyFilter();

    CalcPowerSpectrum();

    CalcIDFT();

    PlotTransforms();
}
```

```
void ApplyFilter()
{
    size_t freq_start = 0;
    size_t freq_stop = fCos.size();
    for (size_t term{freq_start}; term < freq_stop; ++term)
    {
        fCos.at(term) = 0;
        fSin.at(term) = 0;
    }
}
```

Fix the bug in this code

3. High Frequency Filter

Expected Output (Approved Solution)



10 pts

4. Newtonian Kinematics

In the **q04** folder, edit the C++ console application to calculate and display the constant acceleration $\boldsymbol{a}$ and initial velocity $\boldsymbol{v_0}$ values for a particle travelling these distances per time:

time (s)	distance (m)
0.0000	0.0000
1.0000	29.1199
2.0000	83.5010
3.0000	163.1435
4.0000	268.0472
5.0000	398.2123
6.0000	553.6386
7.0000	734.3263
8.0000	940.2752
9.0000	1,171.4855
10.0000	1,427.9570

Using the **method of least squares**, fit an appropriate **quadratic** equation from **kinematics** that governs the behavior of this particle

Assume SI units

4. Newtonian Kinematics

Enter the given data
x = time, y = distance

```
int main()
{
    double vecX[11]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    double vecY[11]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

```

```
    cout << endl;
    cout << "Constant acceleration = "
         << " m/s^2" << endl;
    cout << "Initial velocity      = "
         << " m/s" << endl;
    cout << endl;
```

Edit the code to display the correct values for the constant acceleration and initial velocity

10 pts

5. Combustion of Octane

In the **q05** folder, edit file **octane.txt** to correctly balance the combustion reaction equation of **gasoline**

Ensure the application emits the minimum molar ratios

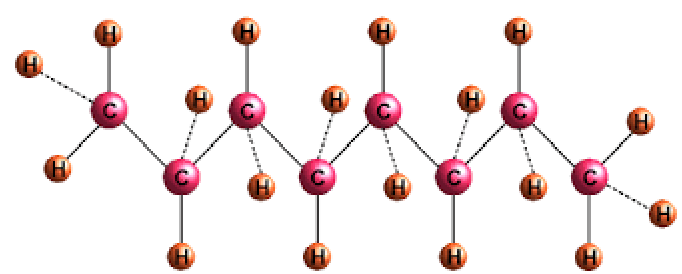
Refer to **Session 17** for assistance on how to encode a chemical equation into the expected input file format

Combustion

HydroCarbon — Carbon Dioxide

$$CH + O_2 \rightarrow CO_2 + H_2O$$

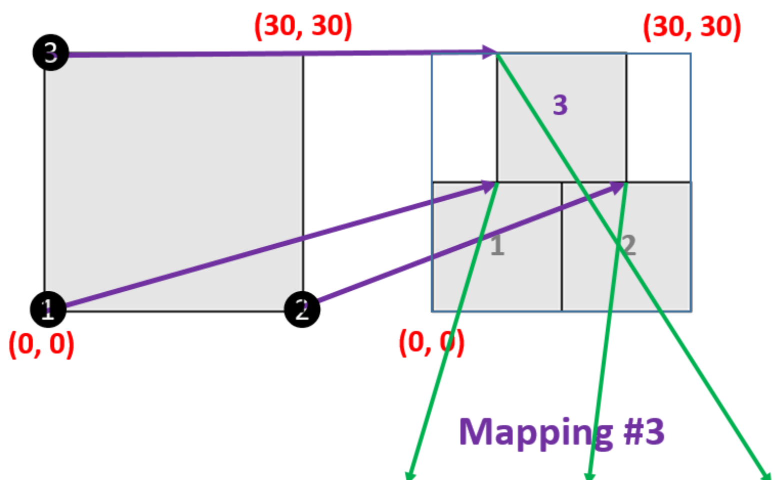
Oxygen — Water



15 pts

6. Hexagonal Fractal

In the **q06** folder, edit the C++ Allegro application to draw a **hexagonal** fractal using an **Iterated Function System**




```
ifs->AddMapping(7.5, 15, 22.5, 15, 7.5, 30, "blue", p);
```

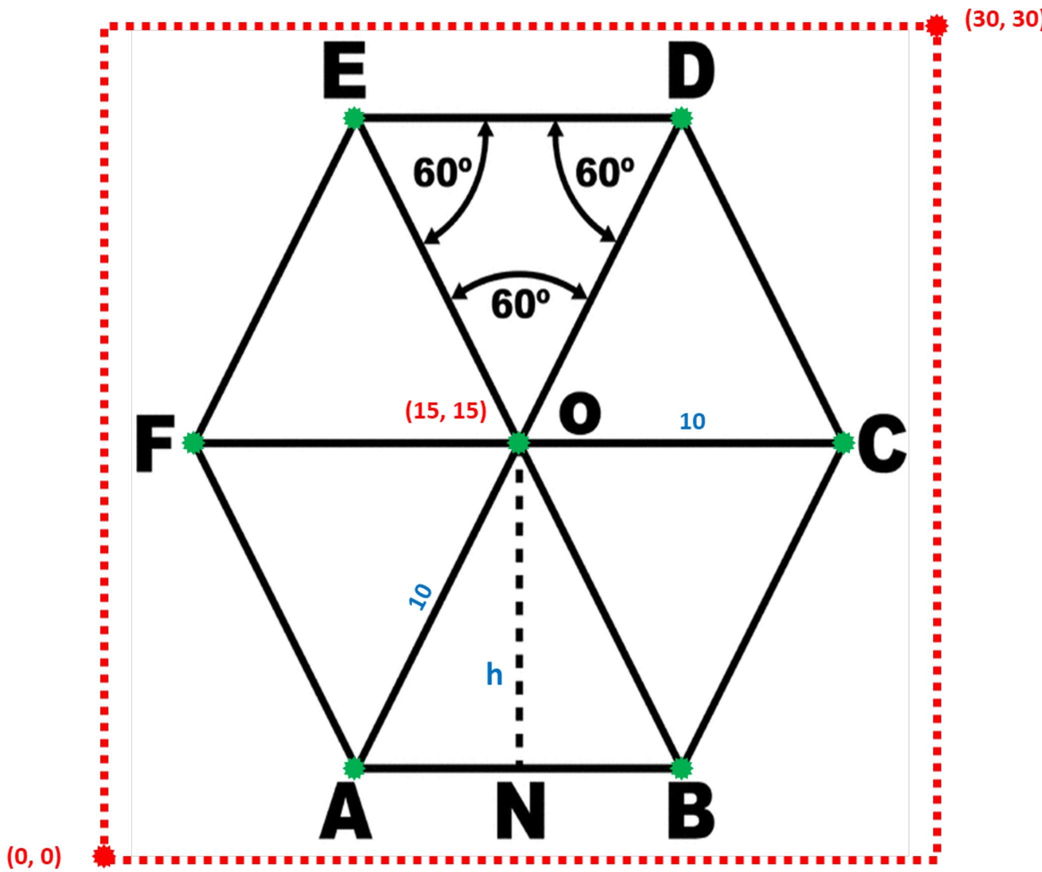
$x'_1, y'_1,$ $x'_2, y'_2,$ x'_3, y'_3

This example is from Sierpiński's Triangle IFS

Provide the final **coordinates** to create <u>six</u> affine transforms (mappings) that cover a regular **hexagon** with side **length 10**

Refer to **Session 24** for assistance on how to encode mappings

6. Hexagonal Fractal

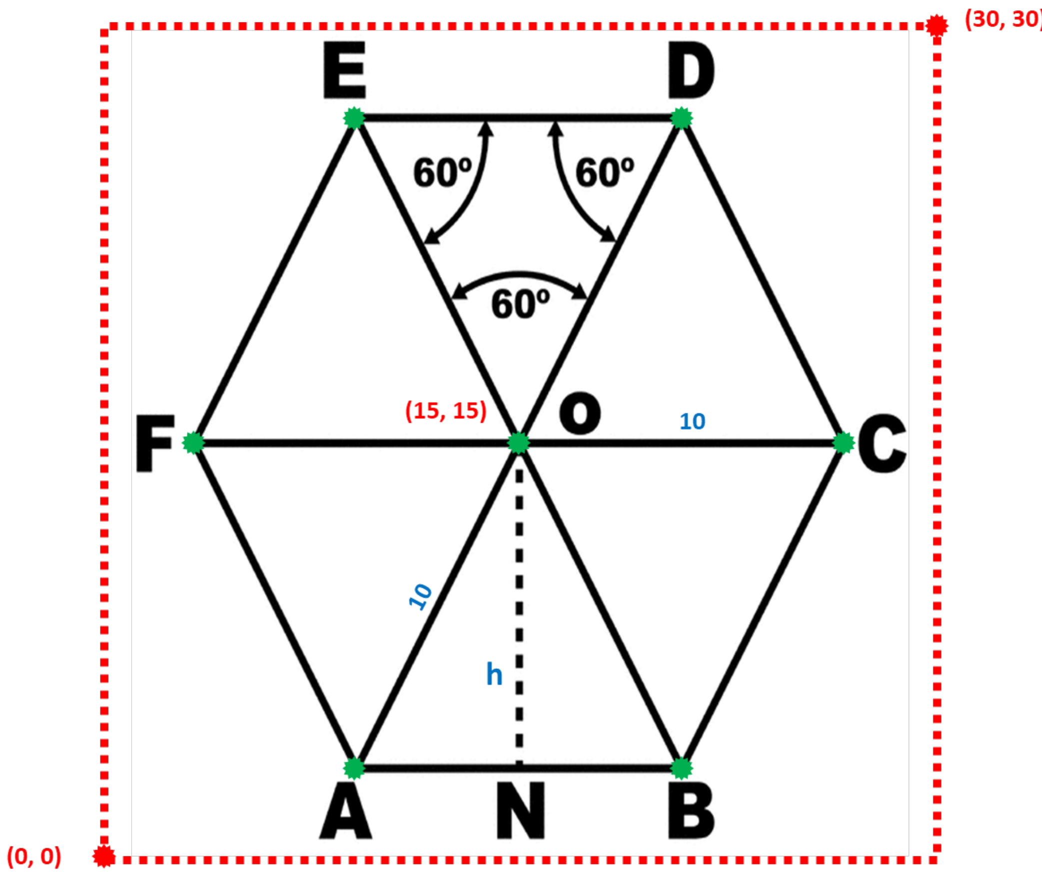


The IFS base frame is a square measuring $(0,0) - (30,30)$

The hexagon is centered on point $(15,15)$

The hexagon has side length of 10

6. Hexagonal Fractal

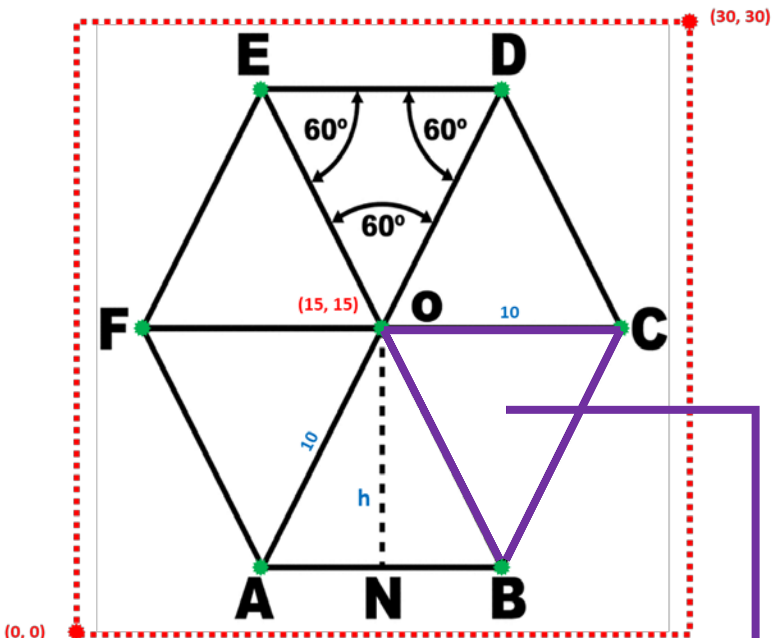


Find the Cartesian coordinates for points A, B, C, D, E, F, O

Encode these six mappings:

1. COD
2. DOE
3. EOF
4. FOA
5. AOB
6. BOC

6. Hexagonal Fractal

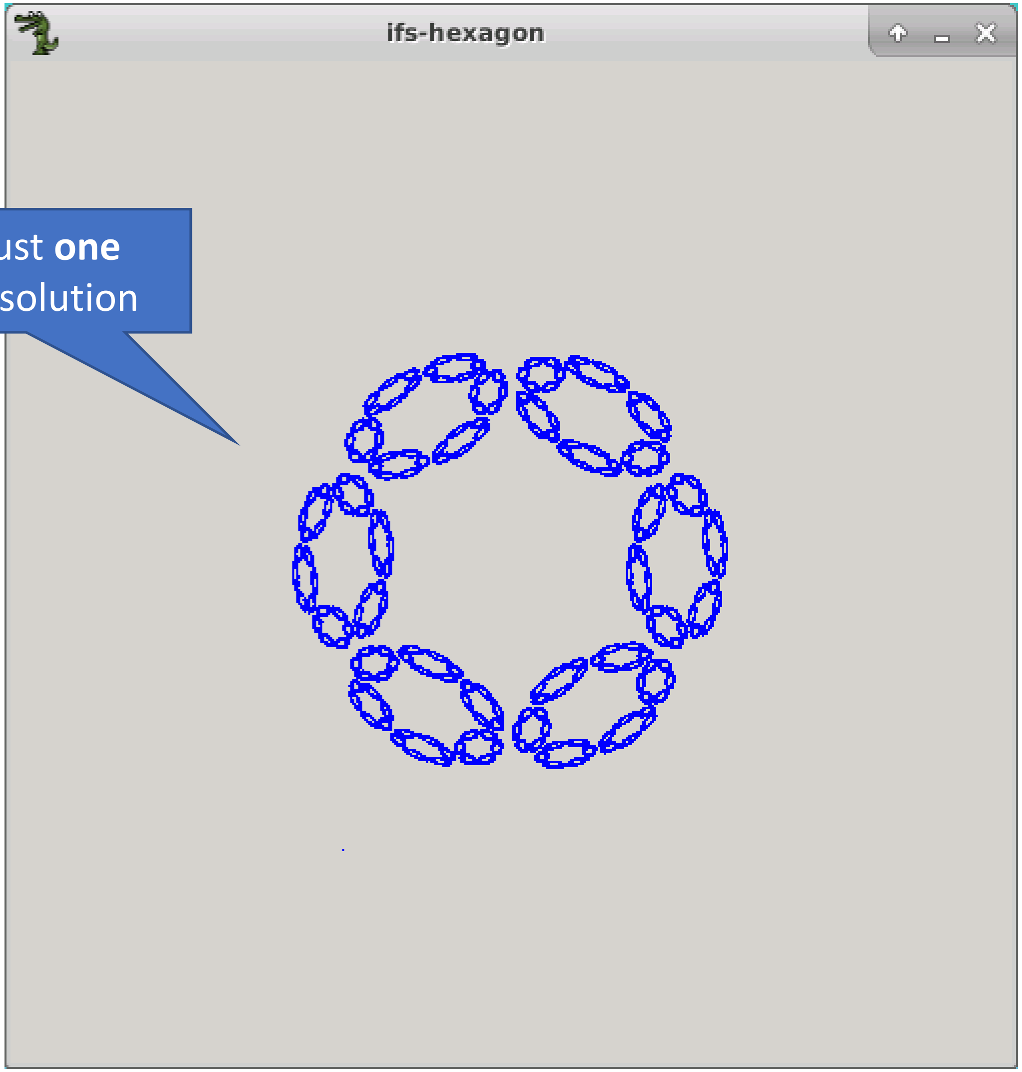


main.cpp

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

    ss.SetWorldRect(0, 0, 30, 30);

    ifs = new IteratedFunctionSystem();

    ifs->SetBaseFrame(0, 0, 30, 30);

    double p = 1. / 6;

    ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);     // COD
    ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);     // DOE
    ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);     // EOF
    ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);     // FOA
    ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);     // AOB
    ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);     // BOC

    ifs->GenerateTransforms();

    ss.HandleEvents();
```

Add & Edit Code Here

6. Hexagonal Fractal



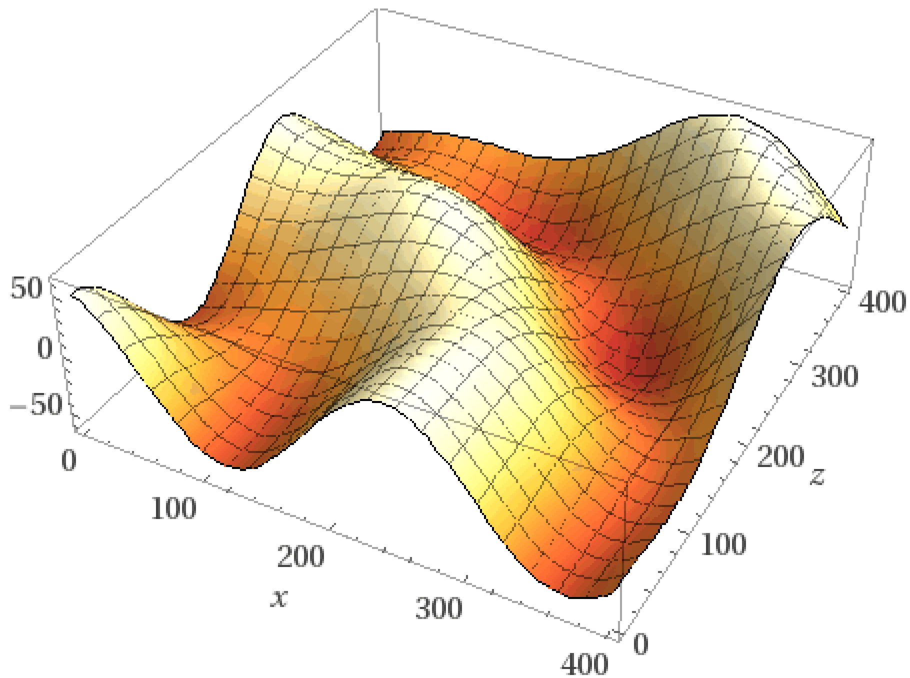
This is just **one** possible solution

5 pts

7. Surface Interpolation

In the **q07** folder, edit the C++ Allegro application to determine the optimal IDW **power** that minimizes the RMSD of this model

$$y = -15\sin\left(\frac{x}{40}\right)\cos\left(\frac{z}{40}\right) + 50\cos\left(\frac{\sqrt{x^2+z^2}}{40}\right)$$

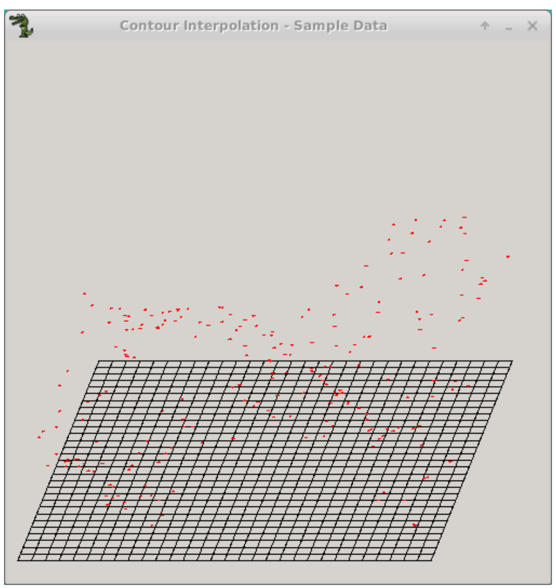


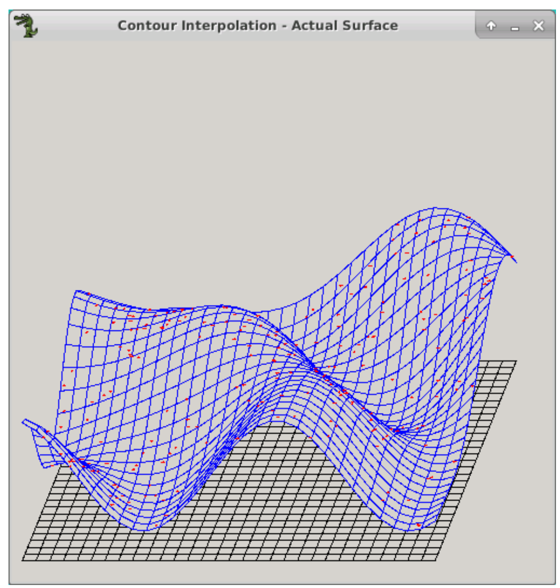
7. Surface Interpolation

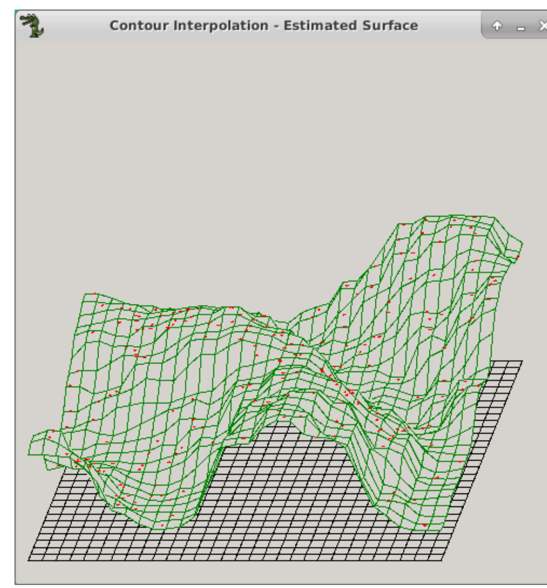
```
double GetActHeight(double x, double z)
{
    return 0;
}
```

Edit this function

```
File  Edit  View  Terminal  Tabs  Help
Press S to see only sample data
Press A to see actual ocean floor
Press E to see estimated ocean floor
Press - to reduce p by 0.1
Press + to increase p by 0.1
```





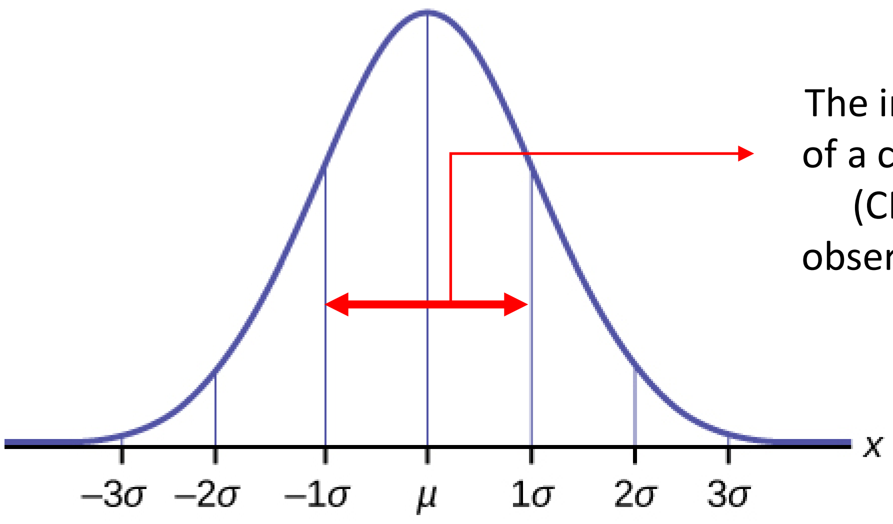


This is the approved solution

15 pts

8. Standard Normal Monte Carlo

In the **q08** folder, edit the C++ Allegro application to use the **Monte Carlo** method to estimate the probability that a normally distributed random variable will fall within $\pm$ the first standard deviation (σ) of its mean (μ)



The integral (the area under the curve) of a continuous probability distribution (CDF) indicates the probability an observation will fall within that interval

Assume we have a **standard normal** distribution for this problem!

8. Standard Normal Monte Carlo

We will use the **Niederreiter** QRNG

main.cpp

```cpp
#include "stdafx.h"
#include "simplescreen.h"
#include "niederreiter.h"

using namespace std;

double f(double x)
{
    return 0;
}

```

Implement the function for a **standard normal** CDF

```cpp
    for (int i{}; i < iterations; ++i)
    {
        qrng.Next(2, &seed, r);
        double x = r[0] * -2.0 - 1.0;
        double y = r[1] * -0.5;
        if (true)
        {
            ss.DrawPoint(x, y, "red");
            count++;
        }
        else
            ss.DrawPoint(x, y, "blue");
    }

```

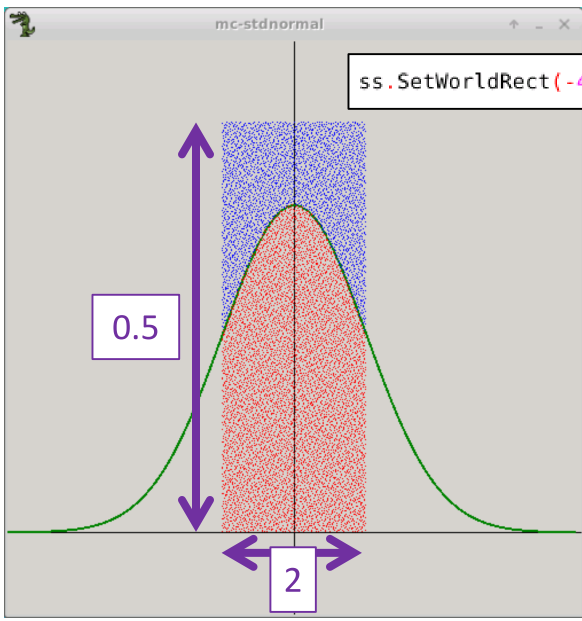
Edit this logic to only count points that are under the curve **f(x)**

8. Standard Normal Monte Carlo

Insert the actual (expected) value for this area

```
    double estArea = (double)count / iterations;
    double actArea = 1.0;
    double err = (actArea - estArea) / actArea * 100;

    cout << "Std Normal 1st Deviation Area QRNG" << endl
         << "Iterations = " << iterations << endl
         << "Est. Area  = " << estArea << endl
         << "Act. Area  = " << actArea << endl
         << "Abs. % Err = " << abs(err) << endl
         << endl;
}
```

$$\frac{dots_{inside}}{dots_{total}} = \frac{area_{under\ curve}}{area_{rectangle}}$$




```
ss.SetWorldRect(-4, -0.1, 4, 0.6);
```

Expected Output (Approved Solution)

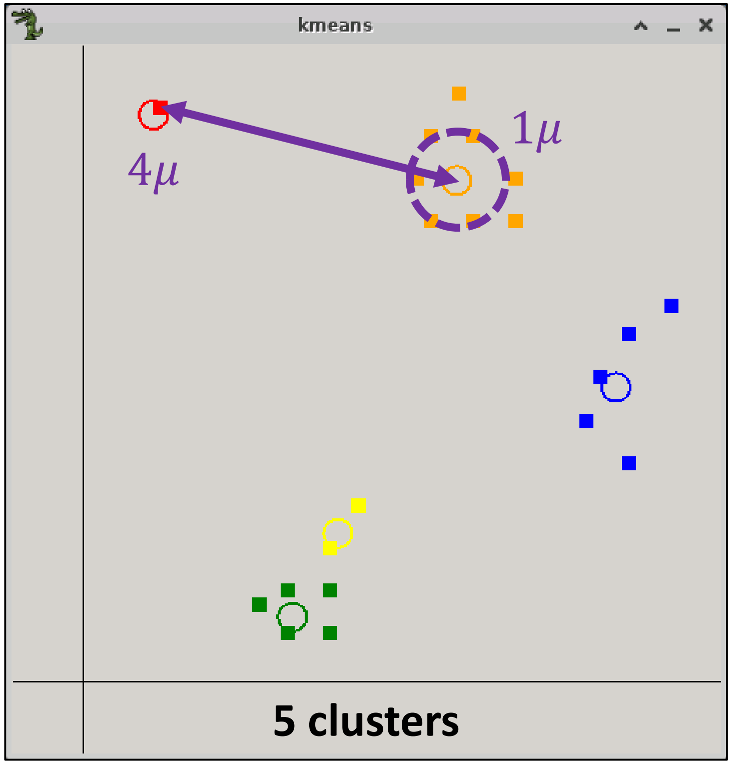
mc-stdnormal

File Edit View Terminal Tabs Help

```
Std Normal 1st Deviation Area QRNG
Iterations = 10,000
Est. Area  = 0.683
Act. Area  = 0.682689
Abs. % Err = 0.045483
```

5 pts

9. kMeans Eviction Criteria



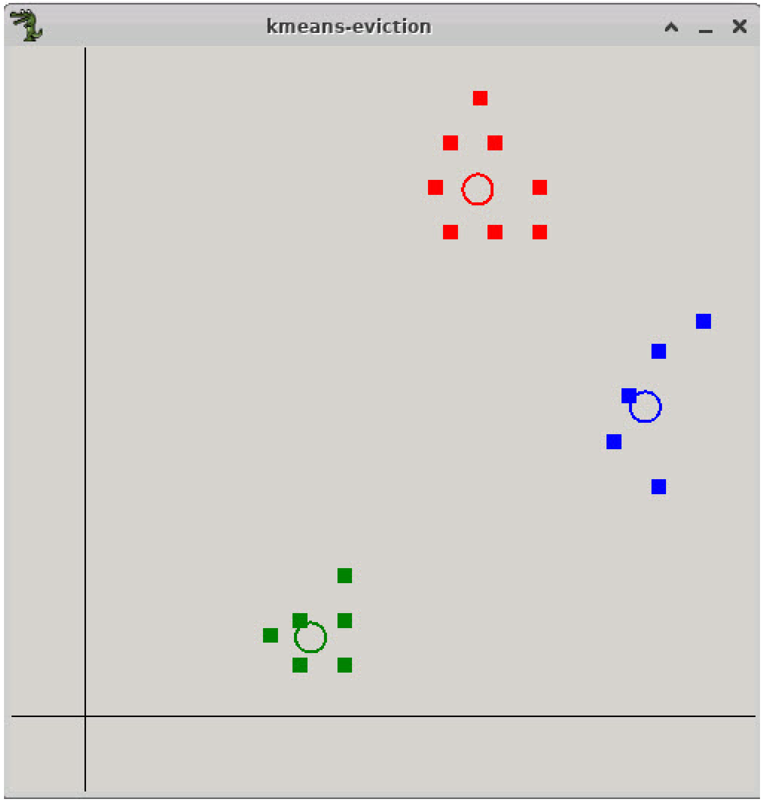
1. In the **q09** folder, within the C++ Allegro application, edit the **GetDistance()** function to use the Manhattan (Taxicab) distance formula

2. When using three clusters, determine the proper value for the **mean_multiple** variable that eliminates the data outlier, but does not evict reasonable data points from their assigned cluster

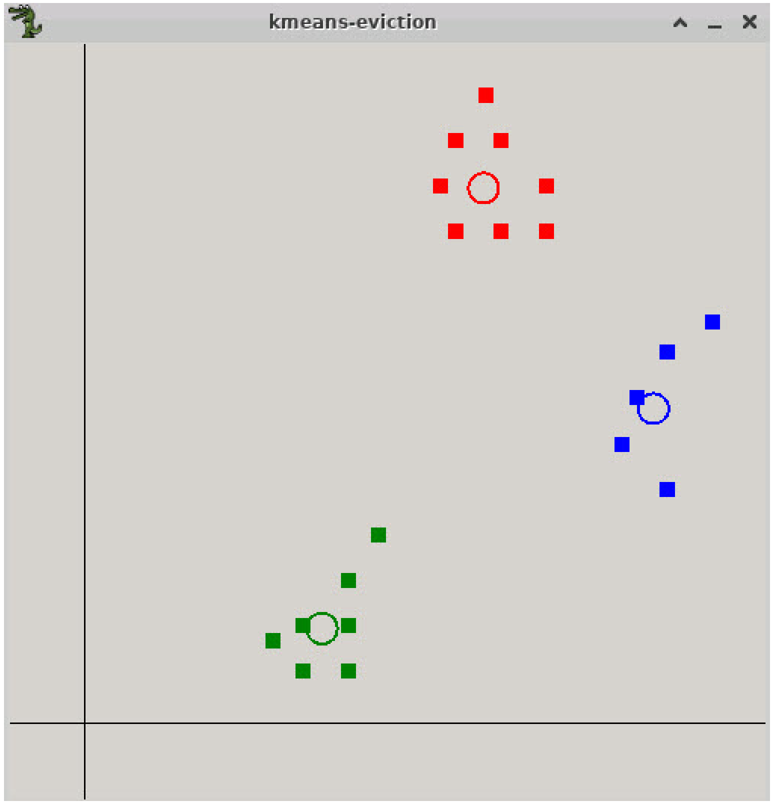
```
int num_clusters{3};
double mean_multiple{0};
```

5 pts

9. kMeans Eviction Criteria



Incorrect mean_multiple



Correct mean_multiple