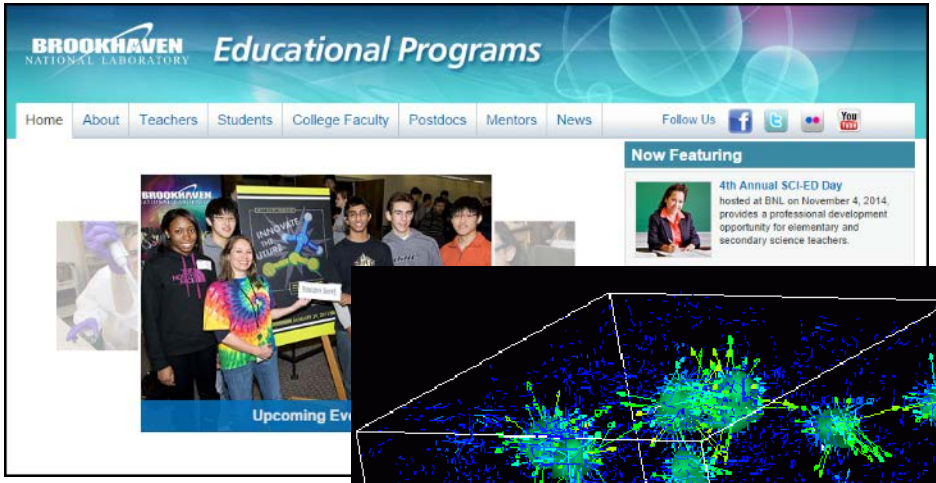
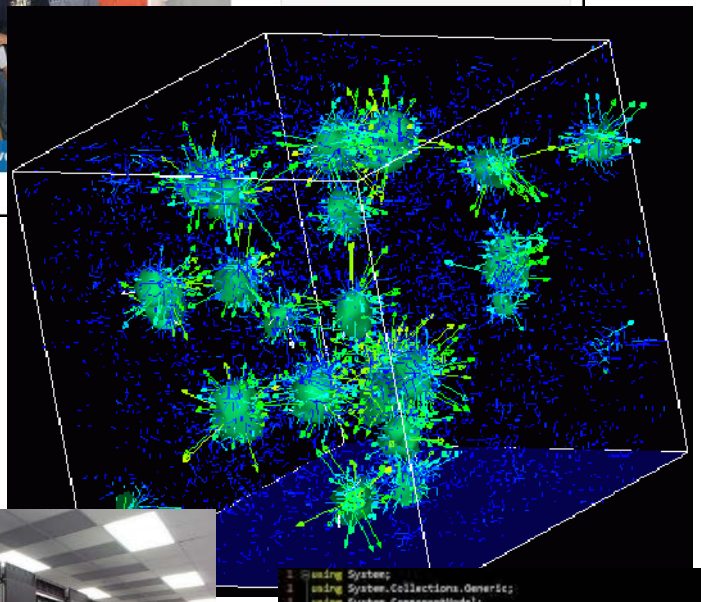
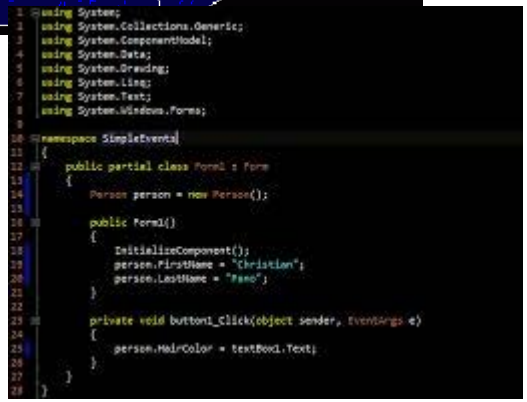






Survey of Scientific Computing (SciComp 301)

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 calculate the **Lotka-Volterra** (1920) differential equations for given characteristics & initial conditions

Fig. 1.1 – Alfred Lotka

$$\alpha = 2, \beta = 1.1, \gamma = 1.0, \delta = 0.9$$
$$x(0) = 1, y(0) = 0.5$$

In this model, at time t:
$x(t)$ represents the **prey** population
$y(t)$ represents **predator** population

Fig. 1.2 – Vito Volterra

Their system of *coupled* non-linear first order differential equations will be solved using the **4th order Runge-Kutta** method

1. Predator-Prey Modelling

```
29  // Lotka-Volterra {Prey} dx/dt
30  double d_prey(double x, double y, double t)
31  {
32      return 0;
33  }
34
35  // Lotka-Volterra {Predator} dy/dt
36  double d_predator(double x, double y, double t)
37  {
38      return 0;
39  }
40
41  int main()
42  {
43      // Initial time
44      double t = 0.0;
45
46      // Initial prey population %
47      double x = 0.0;
48
49      // Initial predator population %
50      double y = 0.0;
51
```

You must write this function

You must write this function

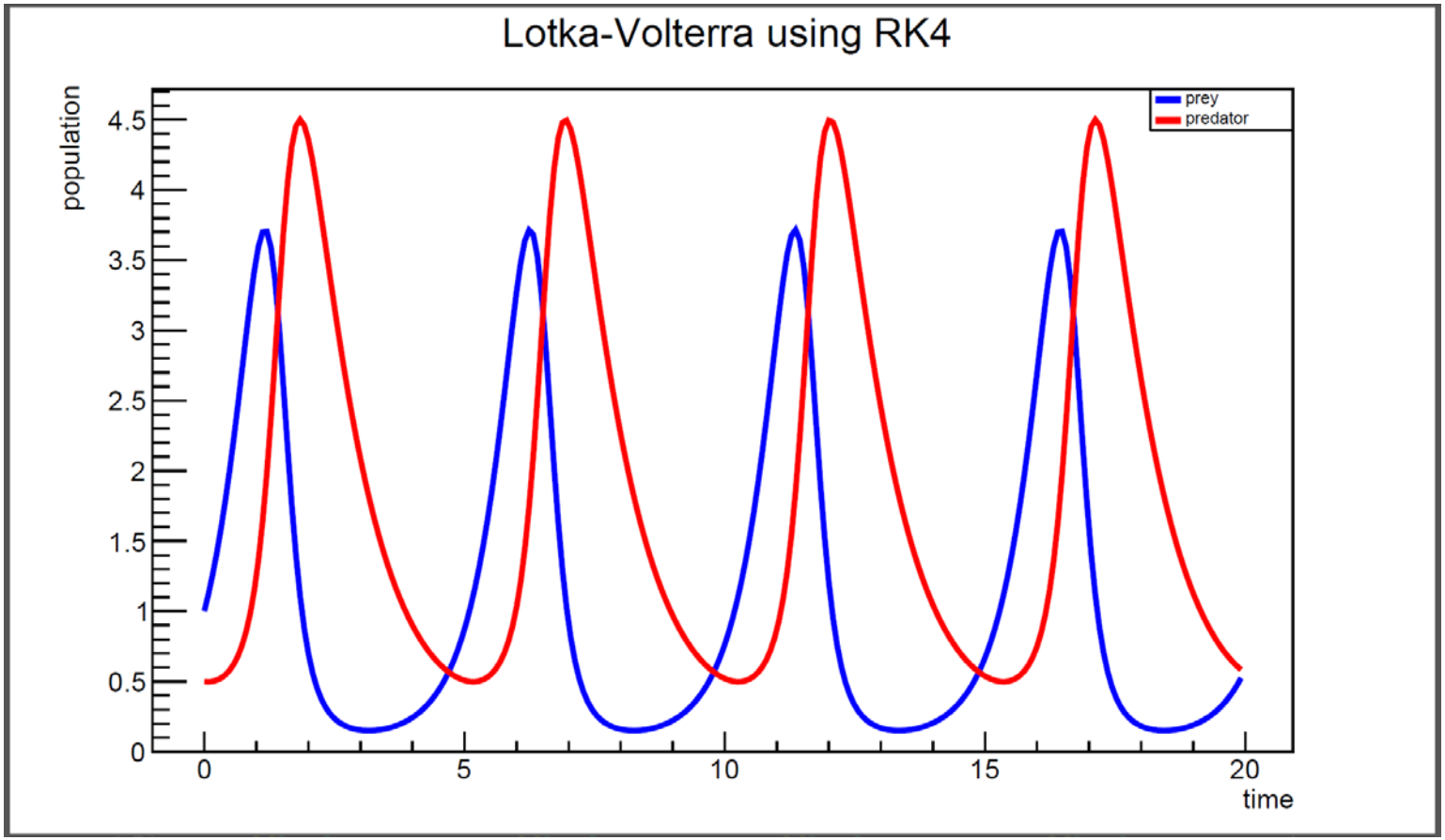
Provide these values

d_prey() = $\frac{dx}{dt}$

d_predator() = $\frac{dy}{dt}$

1. Predator-Prey Modelling

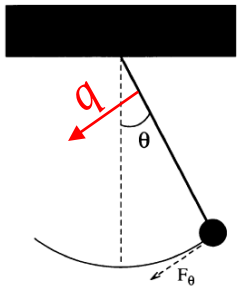
Expected Output (Approved Solution)



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

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

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

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;
```

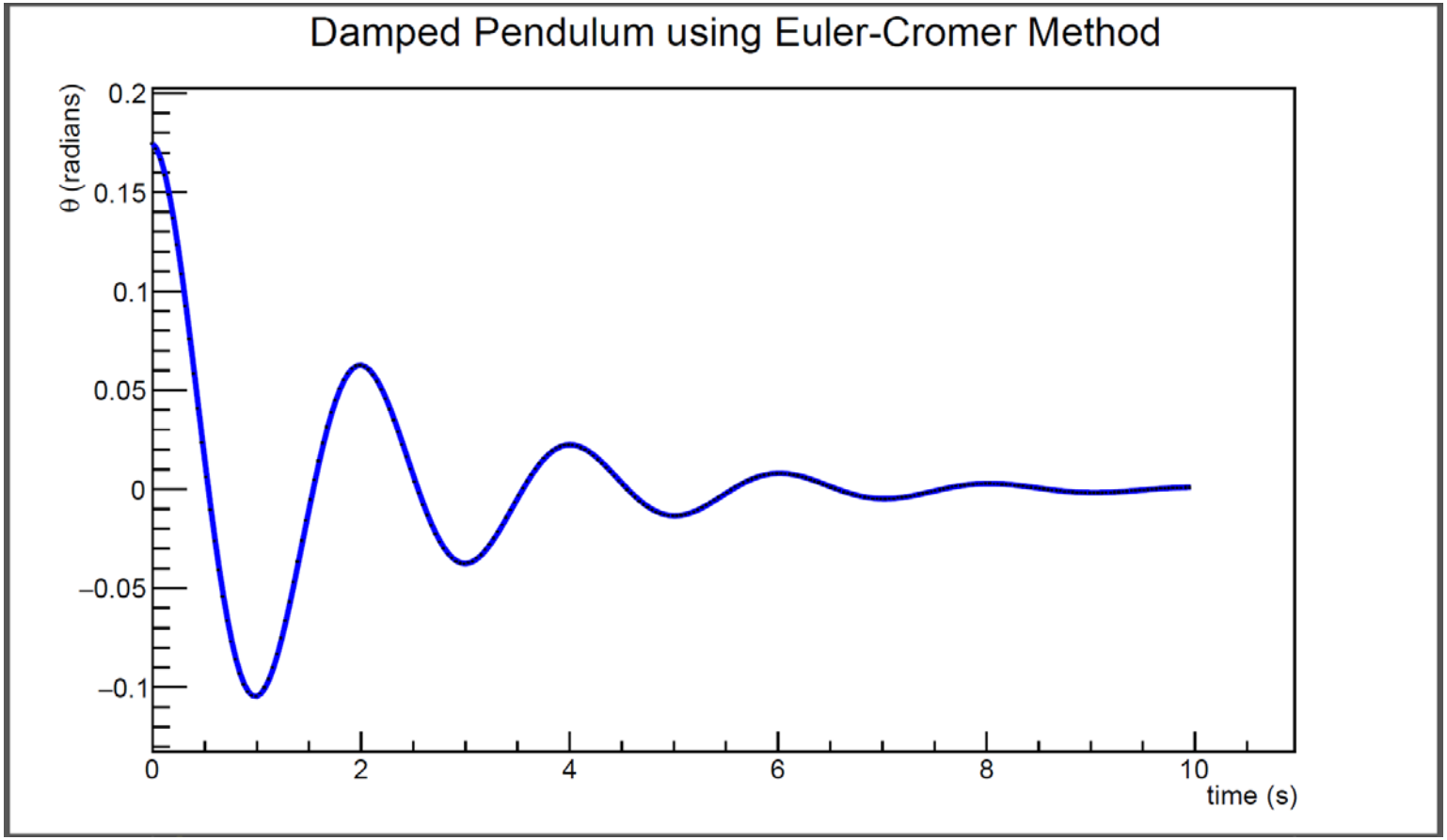
```
36      // Perform Euler method to estimate differential equation
37      for (int step{}; step < timeSteps - 1; ++step) {
38          omega[step + 1] = omega[step] - phaseConstant * theta[step] * deltaTime;
39          theta[step + 1] = theta[step] + omega[step] * deltaTime;
40          timeAt[step + 1] = timeAt[step] + deltaTime;
41      }
42
```

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

Insert Cromer's correction

2. Damped Pendulum

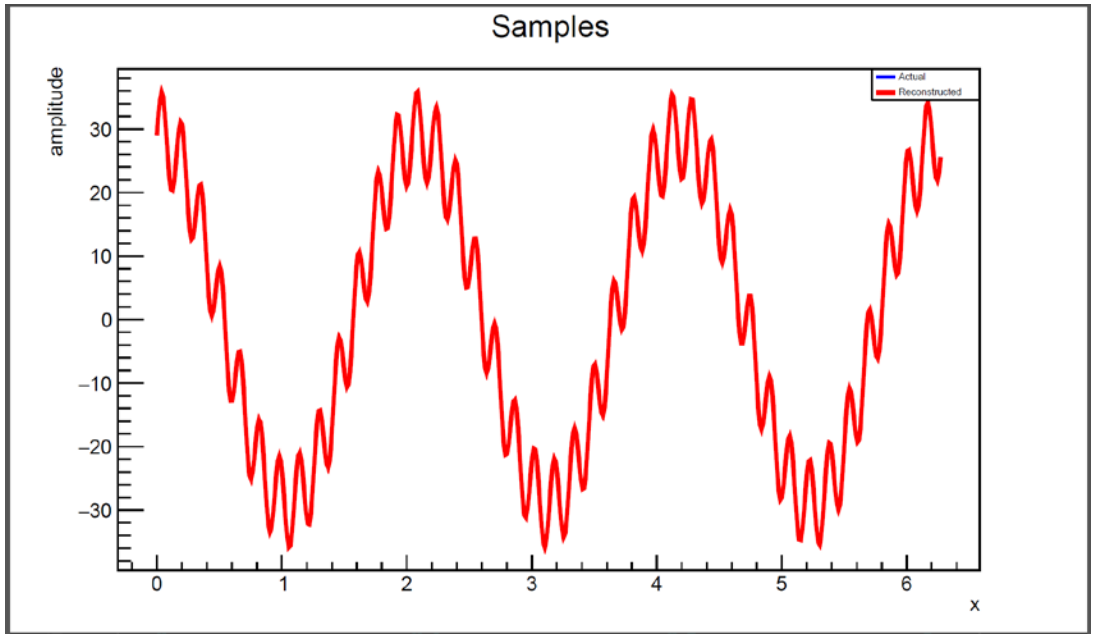
Expected Output (Approved Solution)



15 pts

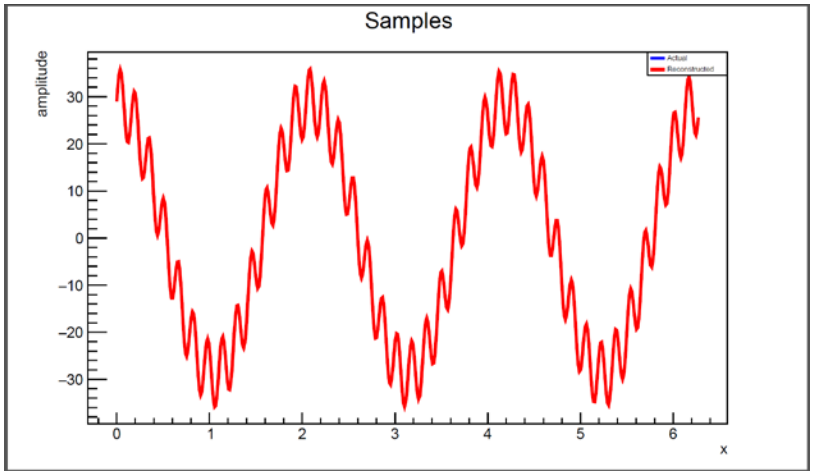
3. High Frequency Filter

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



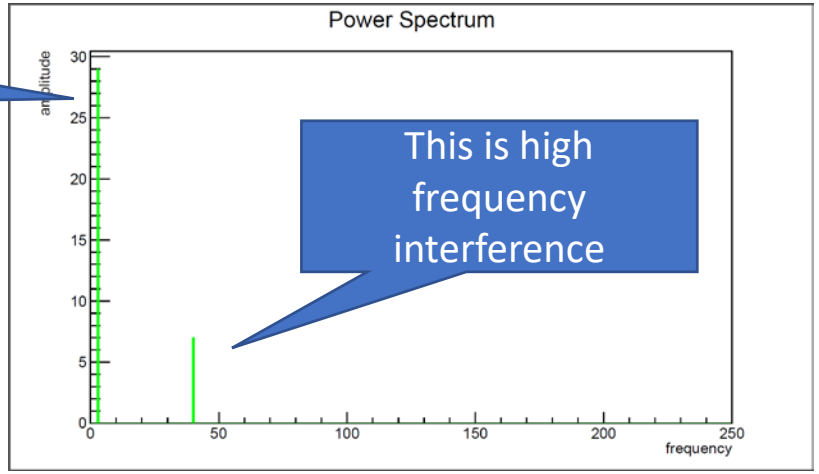
High frequency interference is distorting the capture of a clean primary signal. We want to remove this interference when reconstructing the signal using the inverse discrete Fourier transform (IDFT)

3. High Frequency Filter

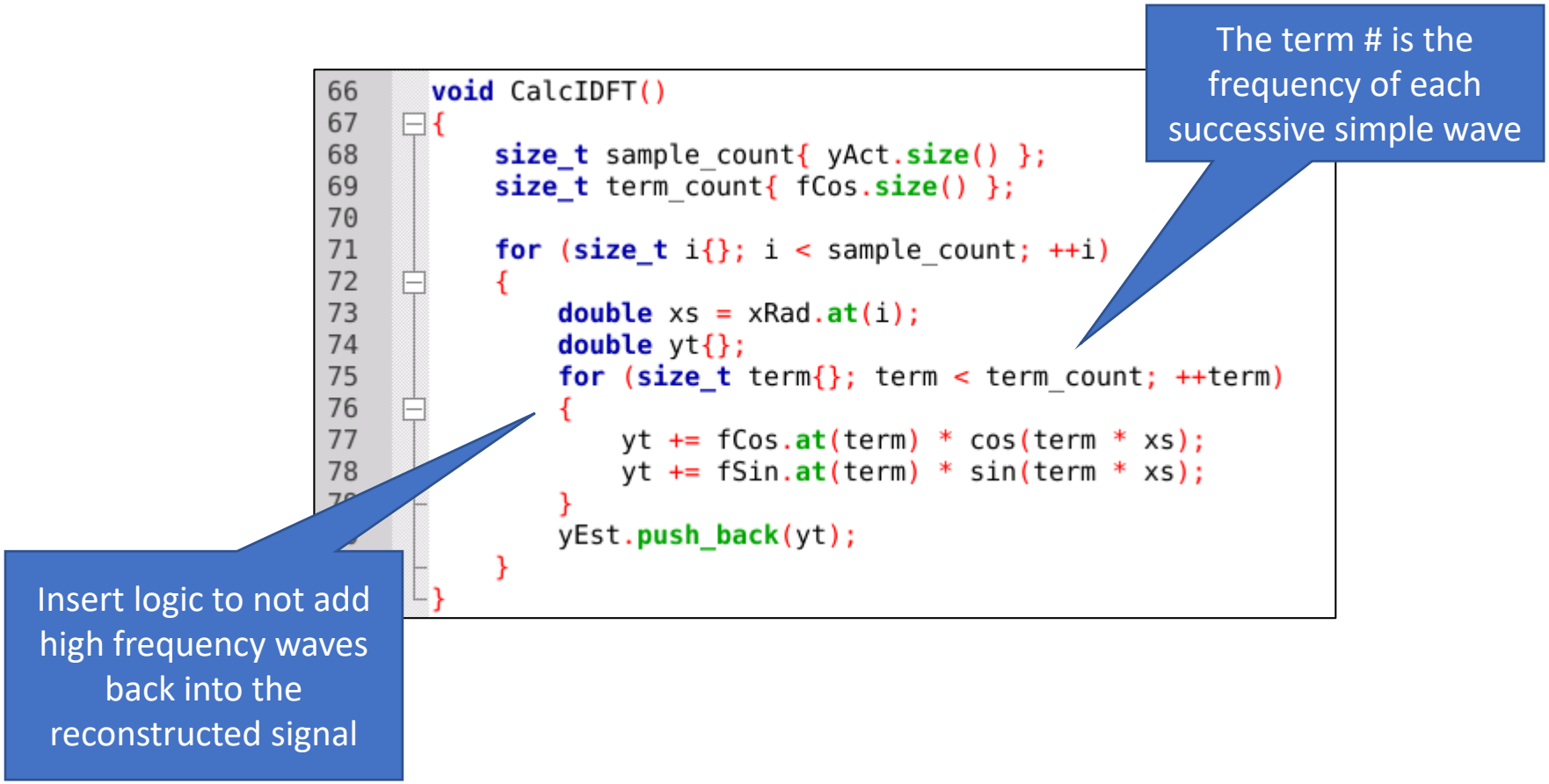


The DFT identifies the constituent simple waves composing the complex wave. By ignoring high frequency simple waves when reconstructing the signal using the IDFT, the interference can be removed

This is the primary signal



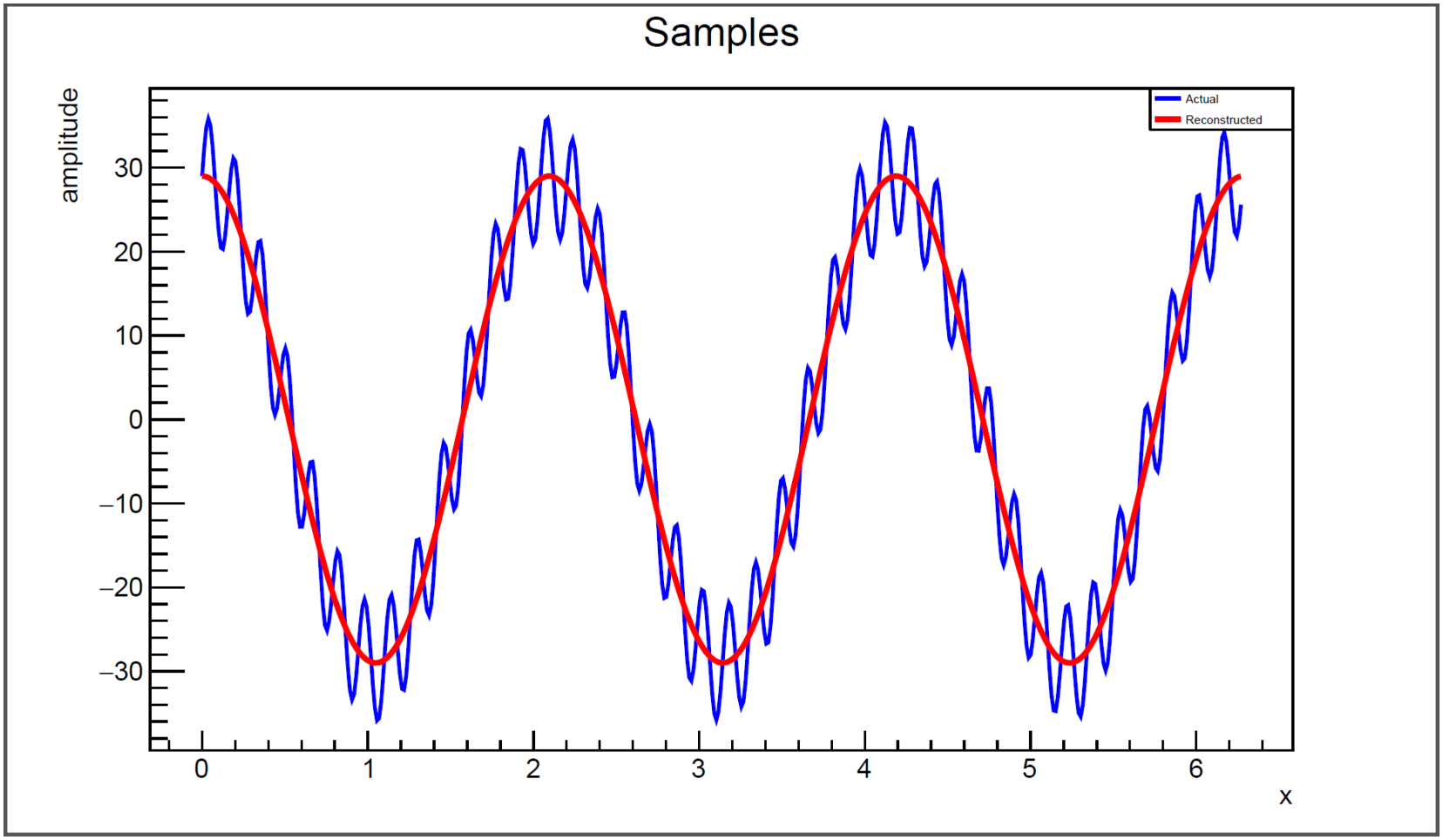
3. High Frequency Filter




```
66  void CalcIDFT()
67  {
68      size_t sample_count{ yAct.size() };
69      size_t term_count{ fCos.size() };
70
71      for (size_t i{}; i < sample_count; ++i)
72      {
73          double xs = xRad.at(i);
74          double yt{};
75          for (size_t term{}; term < term_count; ++term)
76          {
77              yt += fCos.at(term) * cos(term * xs);
78              yt += fSin.at(term) * sin(term * xs);
            }
            yEst.push_back(yt);
        }
    }
```

3. High Frequency Filter

Expected Output (Approved Solution)



10 pts

4. Newtonian Kinematics

In the **q04** folder, edit the C++ console application to determine the initial velocity v_0 and constant acceleration a for a particle obeying these distance measures over the stated time. Assume SI units.

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 equation from **kinematics** that governs the behavior of this particle

4. Newtonian Kinematics

Enter given data
x = time, y = distance

```
92  int main()
93  {
94      double vecX[10] { 1,2,3,4,5,6,7,8,9,10 };
95      double vecY[10] { 1,2,3,4,5,6,7,8,9,10 };
96
```

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

Edit code to display correct values for 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 correct molar ratios

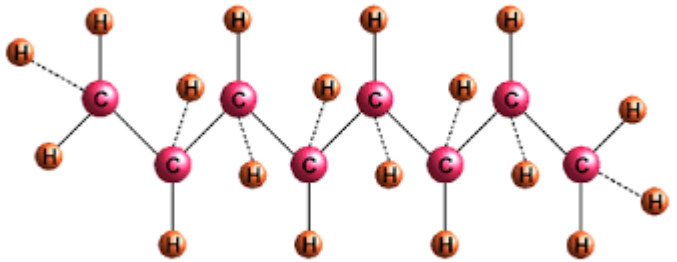
Refer to Session 17 for assistance on how to encode a chemical equation into the expected input file format

Combustion

HydroCarbon + O_2 → Carbon Dioxide + Water

$CH + O_2 \rightarrow CO_2 + H_2O$

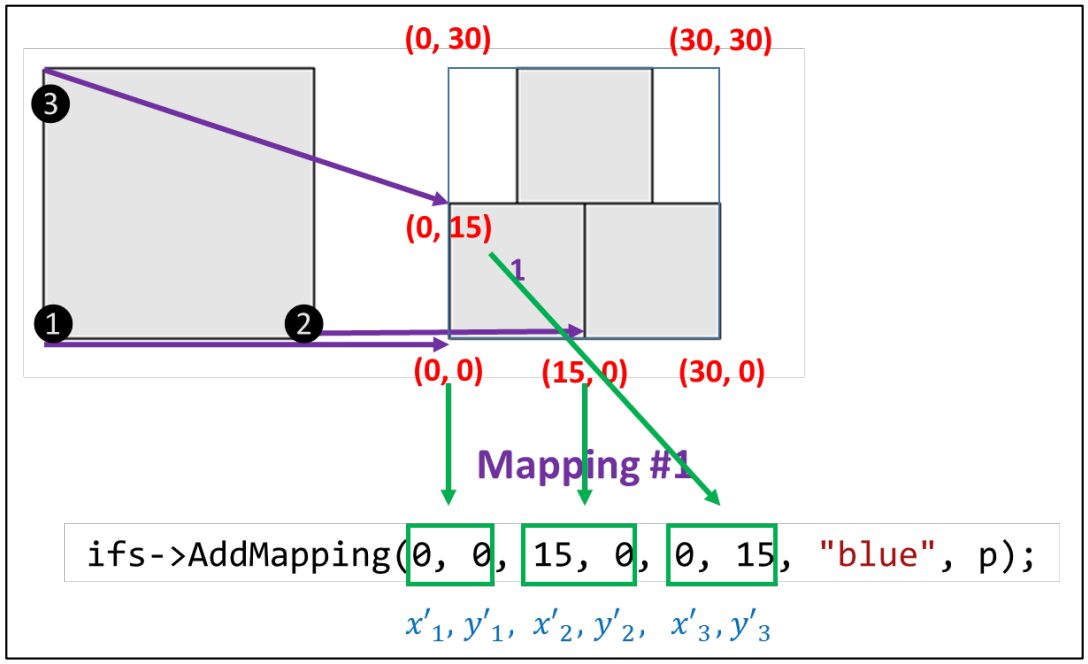
Oxygen



15 pts

6. Hexagonal Fractal

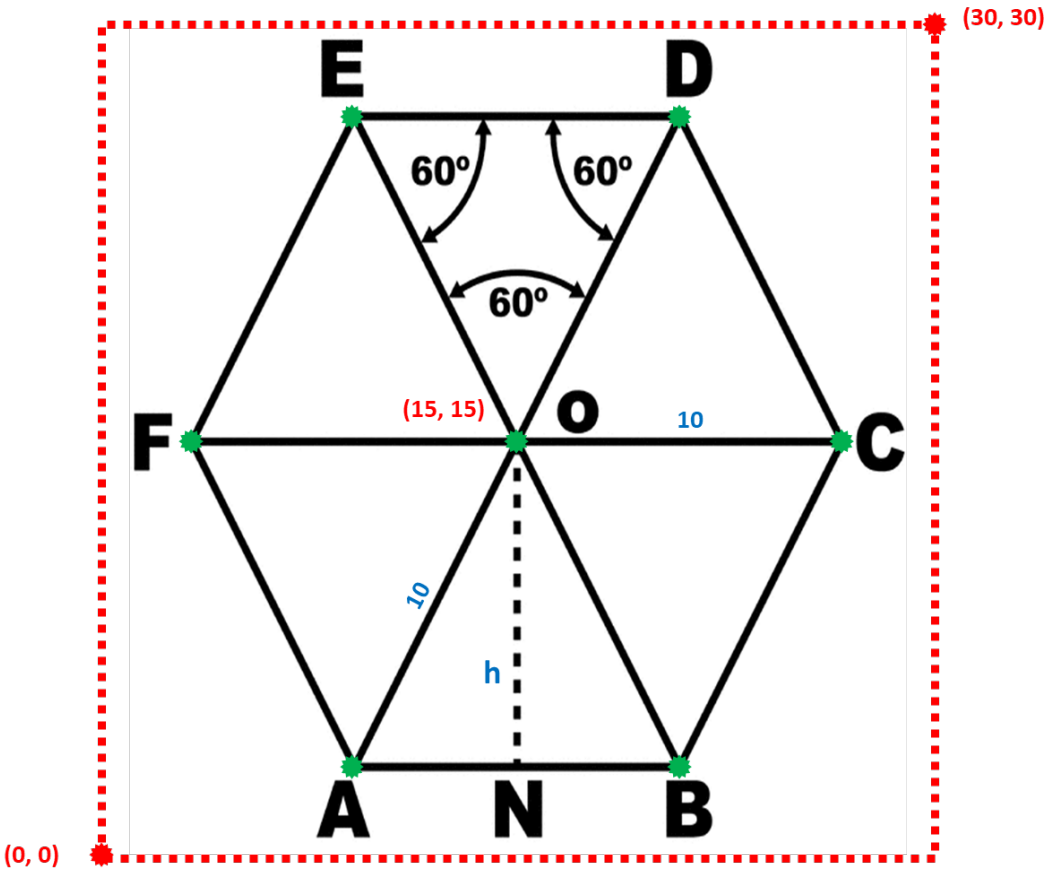
In the **q06** folder, edit the C++ Allegro application to draw a **hexagonal** fractal using an **Iterated Function System**



Provide the necessary coordinates to create six 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

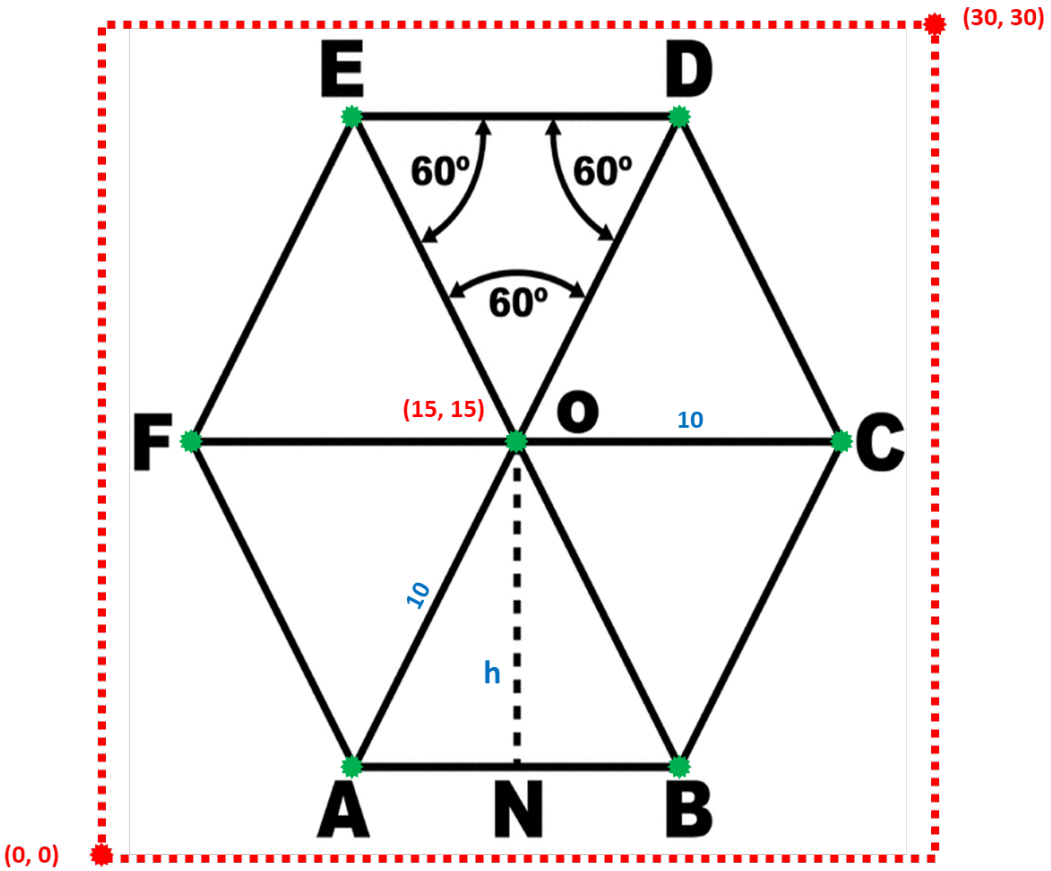


The IFS base frame is a square measuring $(0, 0) - (30, 30)$

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

The hexagon has side length of 10

6. Hexagonal Fractal

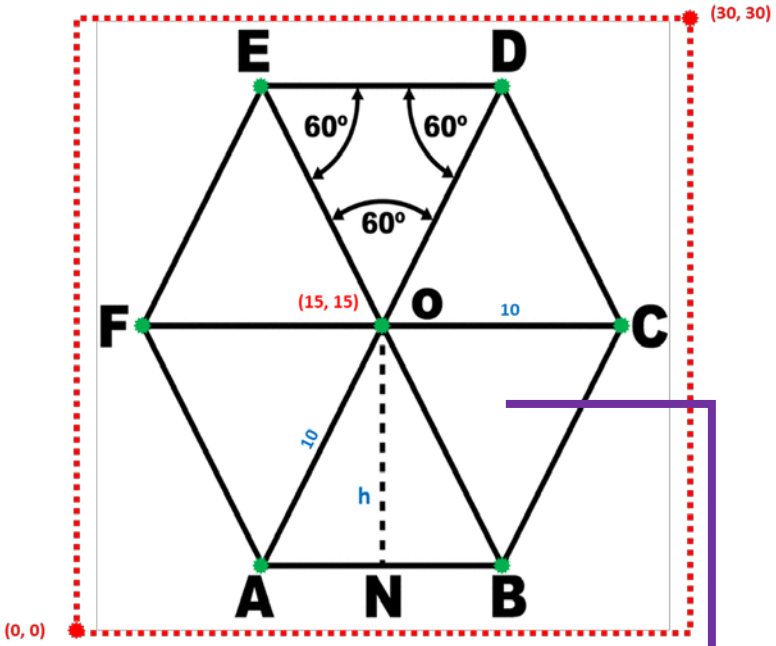


Find the Cartesian coordinates for points A, B, C, D, E, F, O

Encode these six mappings:

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

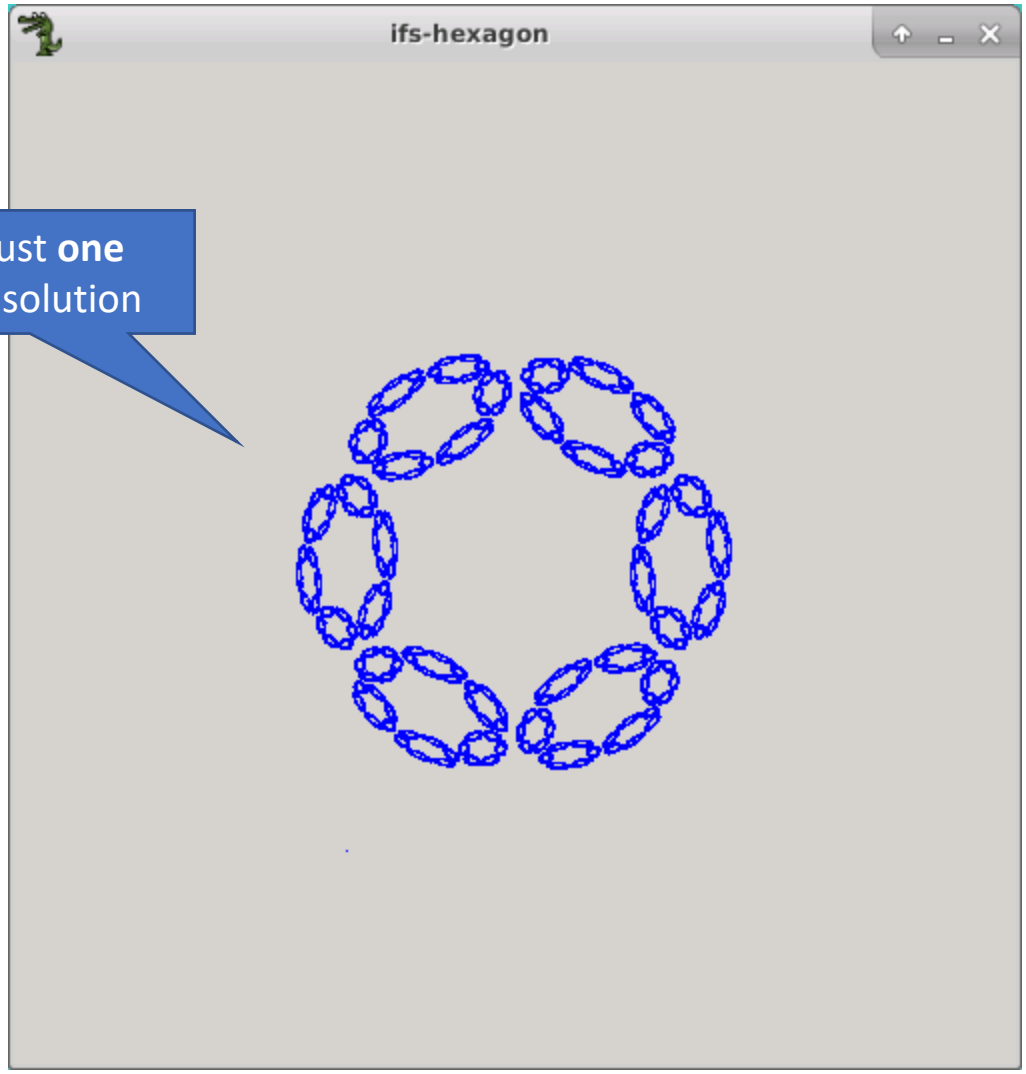
6. Hexagonal Fractal




```
29  int main()
30  {
31      SimpleScreen ss(draw);
32      ss.SetZoomFrame("white", 3);
33
34      ss.SetWorldRect(0, 0, 30, 30);
35
36      ifs = new IteratedFunctionSystem();
37
38      ifs->SetBaseFrame(0, 0, 30, 30);
39
40      double p = 1. / 6;
41
42      ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);      // COD
43      ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);      // DOE
44      ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);      // EOF
45      ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);      // FOA
46      ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);      // AOB
47      ifs->AddMapping(0, 0, 0, 0, 0, 0, "blue", p);      // BOC
48
49      ifs->GenerateTransforms();
50
51      ss.HandleEvents();
```

Edit these lines

6. Hexagonal Fractal

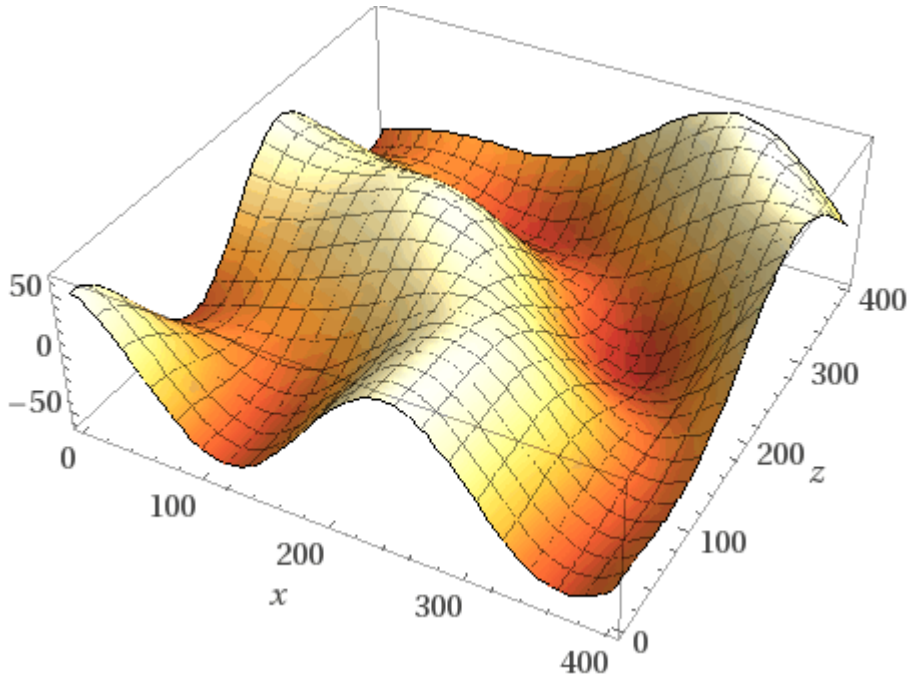


5 pts

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

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

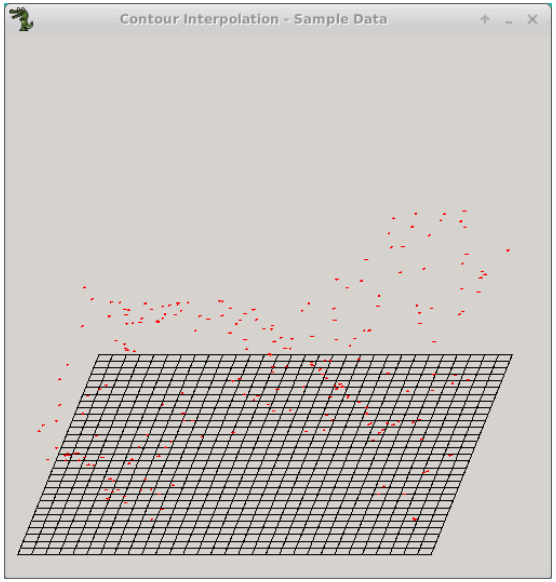


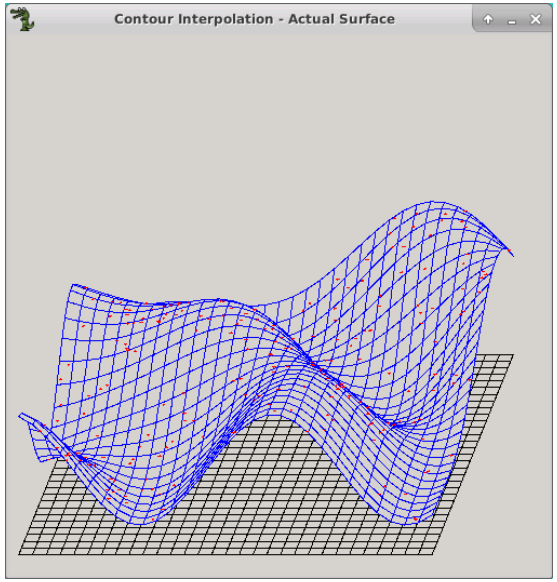
7. Surface Interpolation

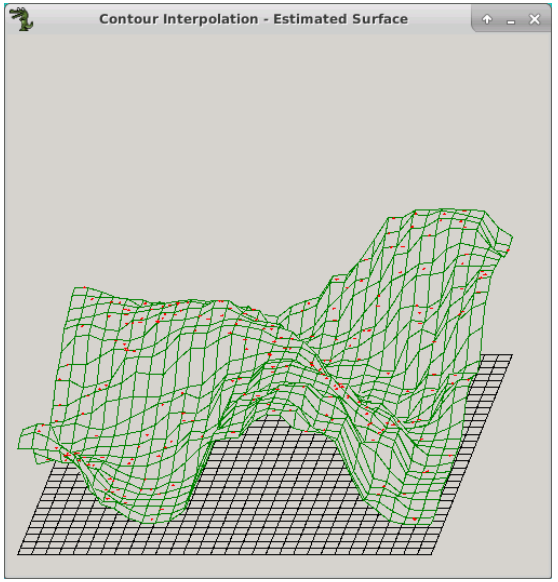
```
29  double GetActHeight(double x, double z)
30  {
31      return 0;
32  }
```

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
```



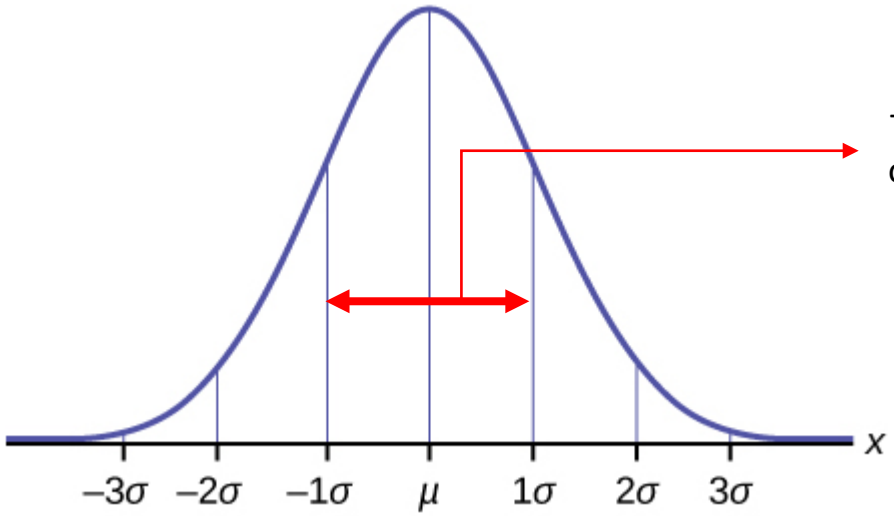




15 pts

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 occur within that measure window

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

8. Standard Normal Monte Carlo

We will use the **Niederreiter** QRNG

```
1   // mc-stdnormal.cpp
2
3   #include "stdafx.h"
4   #include "SimpleScreen.h"
5   #include "Niederreiter2.h"
6
7   using namespace std;
8
9   double f(double x)
10  {
11      return 0;
12  }
```

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

```
38      for (int i{}; i < iterations; ++i)
39      {
40          qrng.Next(2, &seed, r);
41          double x = r[0] * -2.0 - 1.0;
42          double y = r[1] * -0.5;
43          if (true)
45          {
46              ss.DrawPoint(x, y, "red");
47              count++;
48          }
49          else
50              ss.DrawPoint(x, y, "blue");
        }
```

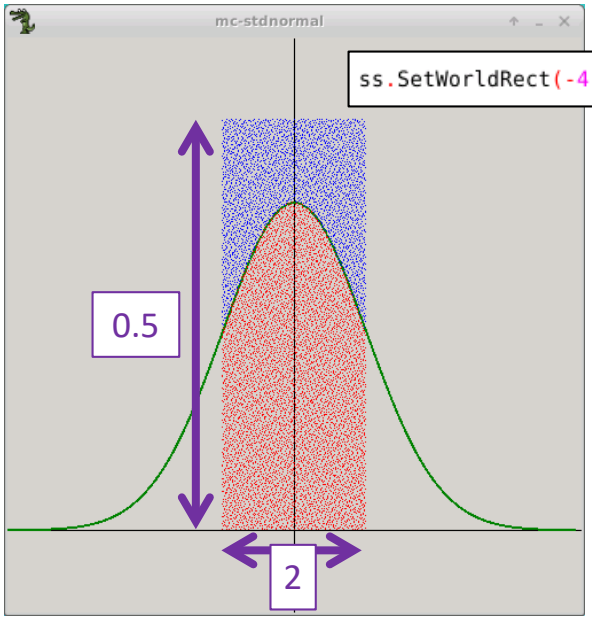
Edit this logic to only count points that are under the curve **f(x)**

8. Standard Normal Monte Carlo

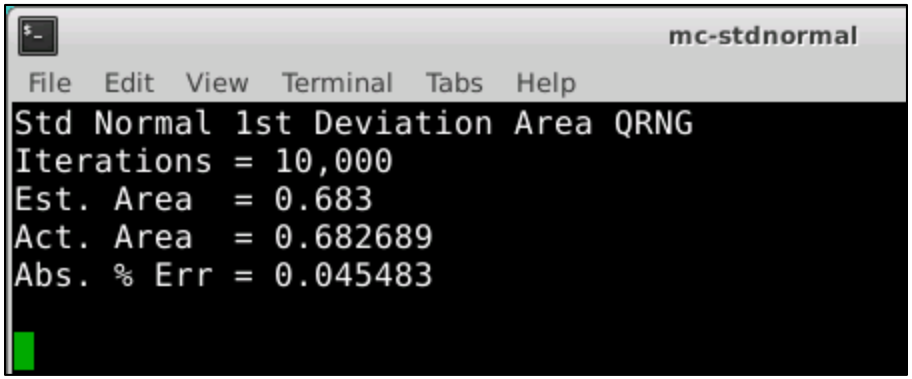
Insert the actual (expected) value for this area

```
54      double estArea = (double)count / iterations;
55      double actArea = 1;
56      double err = (actArea - estArea) / actArea * 100;
57
58      cout << "Std Normal 1st Deviation Area QRNG" << endl
59           << "Iterations = " << iterations << endl
60           << "Est. Area  = " << estArea << endl
61           << "Act. Area  = " << actArea << endl
62           << "Abs. % Err = " << abs(err) << endl << endl;
63  }
64
```

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

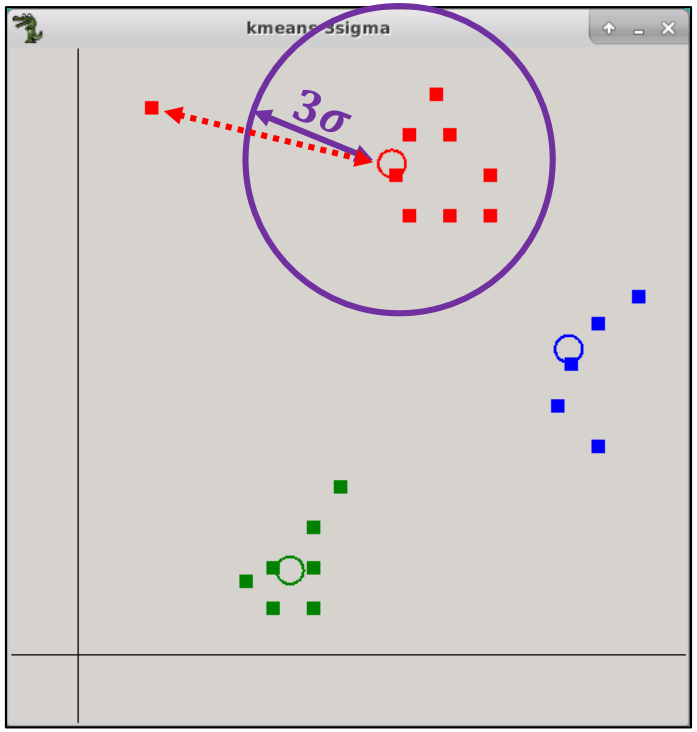


Expected Output (Approved Solution)



5 pts

9. kMeans Eviction Criteria



In the **q09** folder, edit the C++ Allegro application to determine the *reasonableness* of evicting data outliers that are beyond three sigma's (3σ) distance from the centroid {mean μ point} of all the kMeans clusters

If the points are distributed ***normally*** around the cluster's mean point, then 99.97% of all points should fall within 3σ distance of the cluster centroid

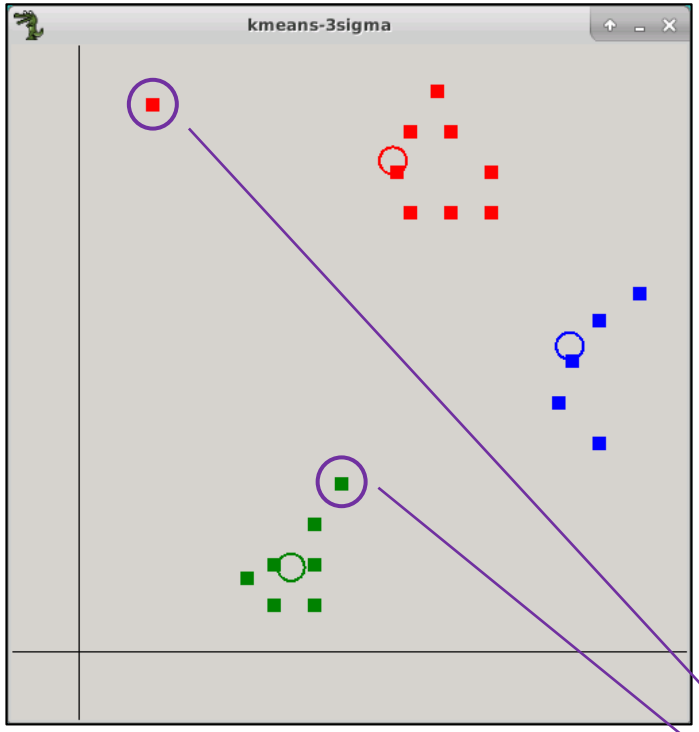
$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

What are the possible complications with using 3σ as a measure of cluster inclusion?

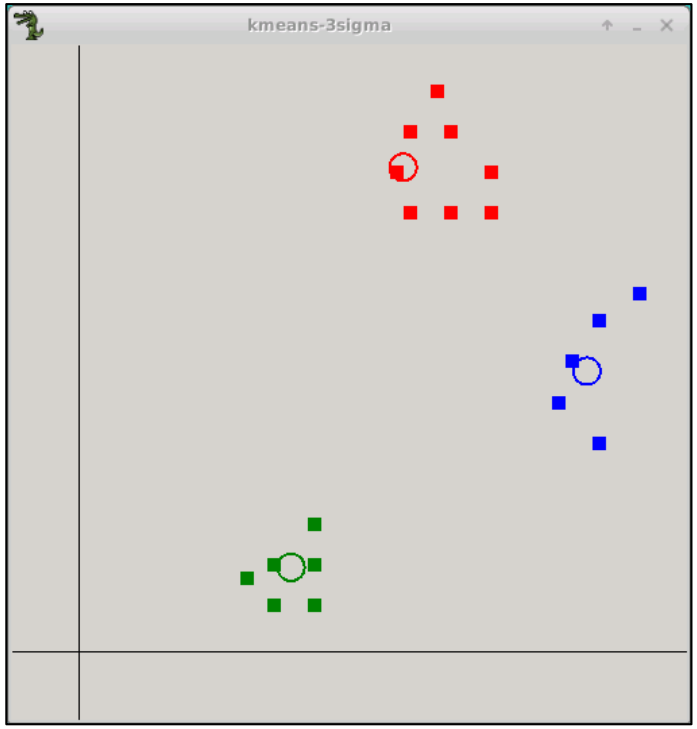
5 pts

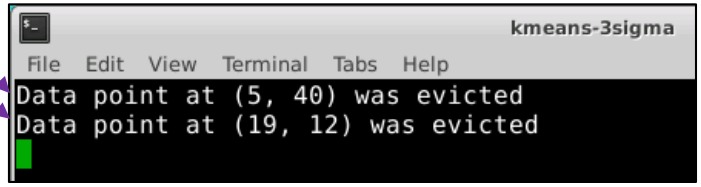
9. kMeans Eviction Criteria

Before evictions



After evictions





Did we evict too many data points?
Does the algorithm converge?

5 pts

9. kMeans Eviction Criteria

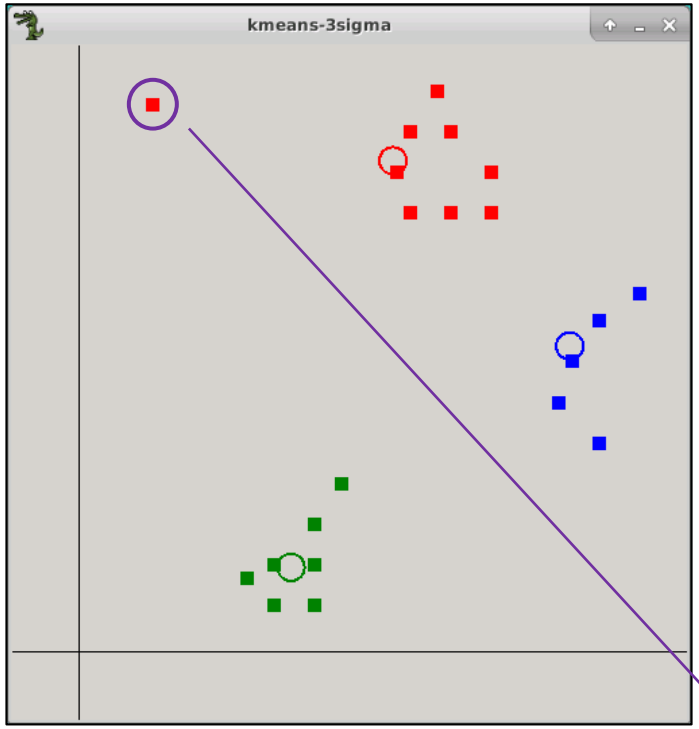
```
232     // Phase III
233     // Evict any data point more than 4 sigmas away
234     // from the mean of its currently assigned cluster
235     if (converged)
236     {
237         for (auto c : *clusters)
238         {
239             // Calculate the standard deviation of distance from
240             // each cluster's center to each of its assigned points
241             c->dist_sigmas = 0;
242             int count = 0;
243             for (auto dp : *dataPoints)
244             {
245                 if (dp->c == c)
246                 {
247                     c->dist_sigmas += sqrt(pow(dp->x - c->x, 2) +
248                                            pow(dp->y - c->y, 2));
249                     count++;
250                 }
251             }
252             c->dist_sigmas = sqrt(c->dist_sigmas / count) * 3;
253
254             // For each data point belonging to this cluster,
255             // check if its distance is > 4 sigmas from the center
256             for (auto dp : *dataPoints)
257             {
258                 if (dp->c == c)
259                 {
260                     double dist = sqrt(pow(dp->x - c->x, 2) +
261                                        pow(dp->y - c->y, 2));
262                     if (dist > c->dist_sigmas)
263                     {
264                         // We need to evict this data point from its cluster
265                         // so the algorithm clearly has not yet converged
266                         converged = false;
```

This is the **3** in the 3σ eviction threshold

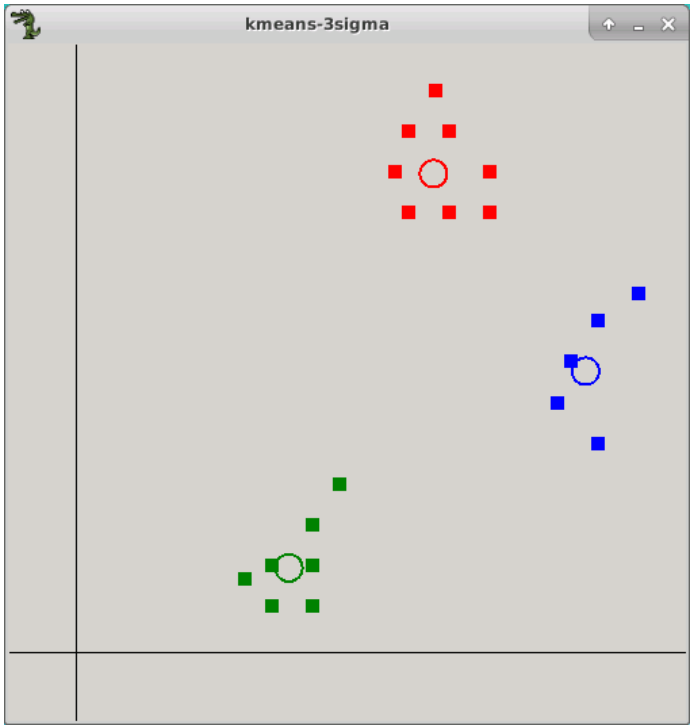
5 pts

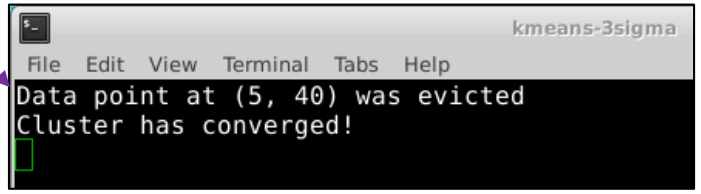
9. kMeans Eviction Criteria

Before evictions



After evictions





Which values of n in the cutoff metric of $n\sigma$ will only prune likely true outliers?