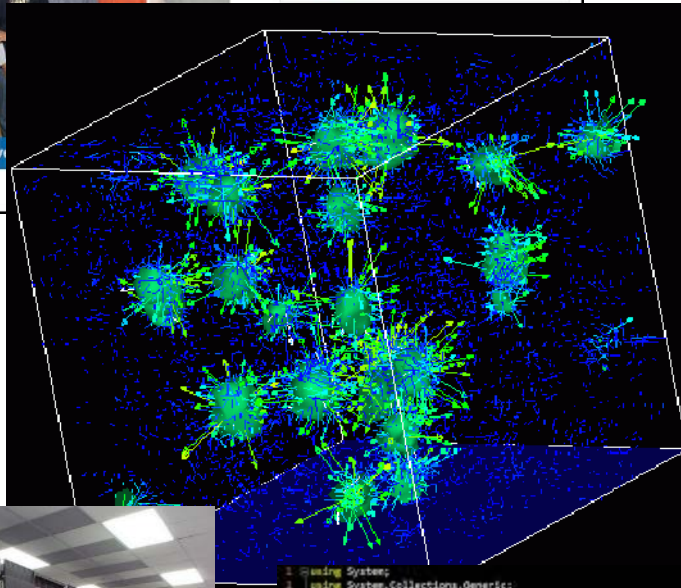




Survey of Scientific Computing (SciComp 301)

Dave Biersach
Brookhaven National
Laboratory
dbiersach@bnl.gov



```

1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9
10 namespace SimpleEvents
11 {
12     public partial class Form1 : Form
13     {
14         Person person = new Person();
15
16         public Form1()
17         {
18             InitializeComponent();
19             person.FirstName = "Christian";
20             person.LastName = "Pano";
21         }
22
23         private void button1_Click(object sender, EventArgs e)
24         {
25             person.MainColor = textBox1.Text;
26         }
27     }
28 }
  
```

Session 06
Statistics,
Euler Line

Session Goals

- Generate **Hero ability scores** in role-playing games
- Create and call **functions** in C++
- Calculate the **mean**, **variance**, and **standard deviation** of a sequence of numbers
- How to request “random” integers within a given range
- Develop a **computational mathematics experiment** that uncovers a **magic number** hidden in all uniform random number distributions
- Draw **Euler’s Line**



Generating Hero Ability Values

- In most role-playing games, heroes have abilities such as strength, dexterity, intelligence, charisma, etc.
- Initial abilities are often measured in ranges like **3 – 18**
- At the beginning of the game, players *roll dice* to determine the initial values for each ability
- The higher the value, the more likely the player will succeed while adventuring



	ABILITY SCORE	ABILITY MODIFIER	TEMP SCORE	TEMP MODIFIER
STR	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
DEX	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
CON	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
INT	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
WIS	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
CHA	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Generating Hero Ability Values

- Two ways of rolling for initial abilities between **3 and 18**
 1. Roll a **20**-sided die just once (**1d20**), but *reroll* if face value is 1, 2, 19, or 20
 2. Roll a **6**-sided die three times (**3d6**), summing the value of each roll
- Using the **1d20** method is faster than **3d6**, especially when having to roll for six separate abilities

	ABILITY SCORE	ABILITY MODIFIER	TEMP SCORE	TEMP MODIFIER
STR	17			
DEX	11			
CON	15			
INT	5			
WIS	7			
CHA	3			



Functions

- You can declare and define your own custom functions
 - Function names use **CamelCase #1** – the *first* letter is Capitalized
 - A function is essentially a custom scope (a group of statements)
- Functions receive value via a **parameter list**
 - A function can get inbound values passed to it from somewhere else in the source code
 - Each parameter has a data type and variable identifier
- Functions output a value via the **return** statement
 - A function can only return one intrinsic data type
 - The return statement is usually at the end of the function

Declaring Functions – Function Prototypes

The diagram shows a snippet of C++ code with several annotations. A red dashed box highlights the first two lines of function prototypes: `double CalcMean1d20(double totalRolls);` and `double CalcMean3d6(double totalRolls);`. Four red callout boxes point to specific parts of this code: 'Function name' points to `CalcMean1d20`, 'Return type' points to `double`, 'Parameter list (types & names)' points to `(double totalRolls)`, and 'Function prototype' points to the entire line `double CalcMean1d20(double totalRolls);`.

```
// HeroAbilities.cpp
#include <random>
#include "stdafx.h"

using namespace std;

double CalcMean1d20(double totalRolls);
double CalcMean3d6(double totalRolls);

double CalcStdDev1d20(double totalRolls, double mean);
double CalcStdDev3d6(double totalRolls, double mean);

seed_seq seed{ 2016 };
default_random_engine generator{ seed };
uniform_int_distribution<> distribution1d20(3, 18);
uniform_int_distribution<> distribution1d6(1, 6);
```

Function name

Return type

Parameter list (types & names)

Function prototype

Functions must be declared before they are referenced!

Defining Functions

Functions open a scope

Multiple parameters

```
double CalcStdDev1d20(double totalRolls, double mean)
{
    double sum{};
    for (double rollNumber{}; rollNumber <= totalRolls; ++rollNumber) {
        int roll = distribution1d20(generator);
        sum = sum + pow(roll - mean, 2);
    }
    double variance = sum / totalRolls;
    double stdDev = sqrt(variance);
    return stdDev;
}
```

Closing
function scope

Function ends here, returns
this value to the caller

Calling Functions

```
int main()
{
    double totalRolls = 1000000;

    cout.imbue(locale(""));

    cout << "Total number of dice rolls: " << totalRolls << endl;
    cout.precision(0) << fixed;

    double mean1d20 = CalcMean1d20(totalRolls);
    double mean3d6 = CalcMean3d6(totalRolls);

    distribution1d20.reset();
    distribution1d6.reset();
}
```

Return value stored in local variables

Don't need types in front of the parameters when calling a function

Generating Hero Ability Values

- Two ways of rolling for initial abilities between **3 and 18**:
 - Roll a **20**-sided die just once (**1d20**), but *reroll* if face value is 1, 2, 19, or 20
 - Roll a **6**-sided die three times (**3d6**), summing the value of each roll
- Which method would you want to use? Why?**

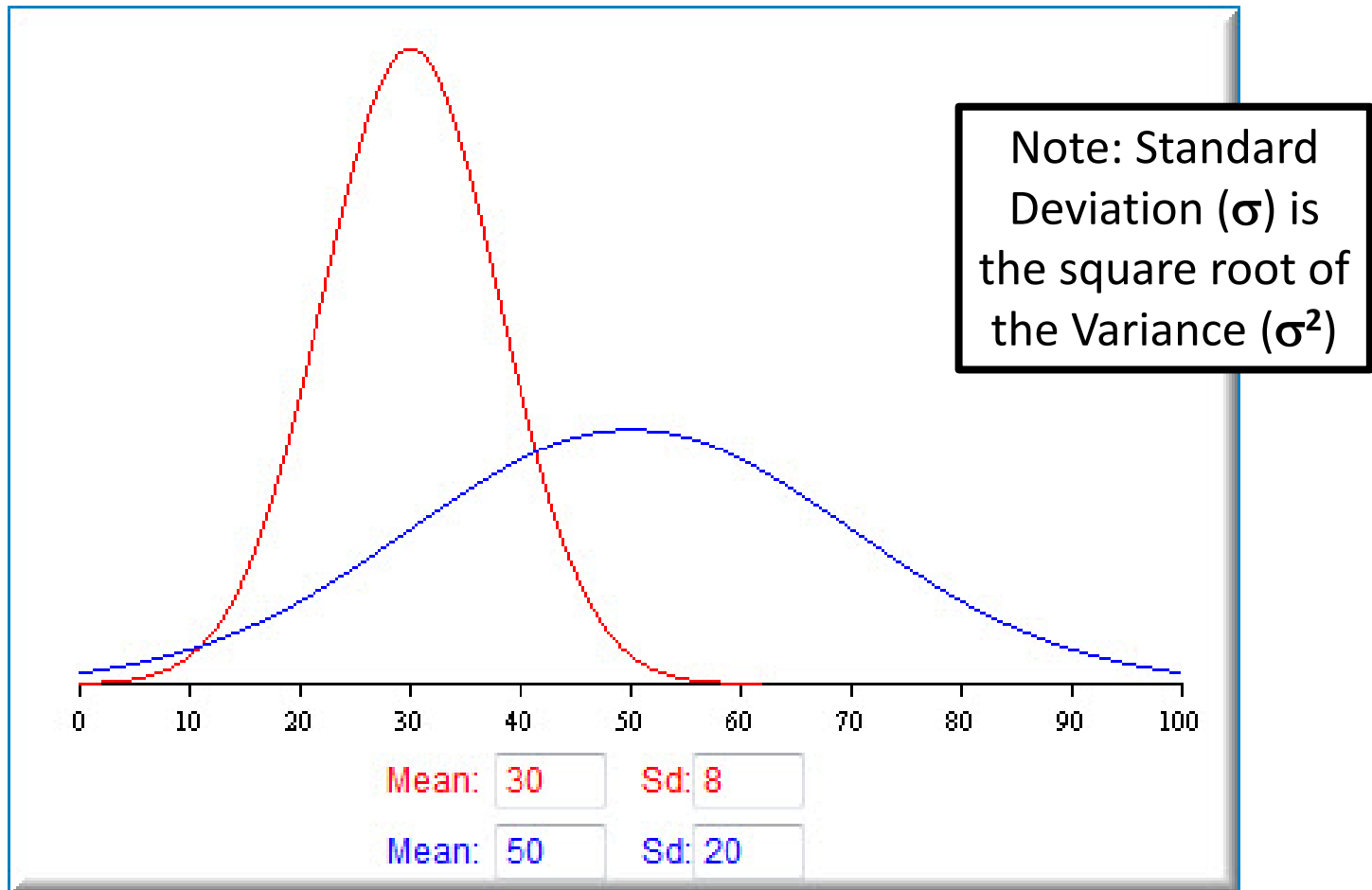
	ABILITY SCORE	ABILITY MODIFIER	TEMP SCORE	TEMP MODIFIER
STR	17			
DEX	11			
CON	15			
INT	5			
WIS	7			
CHA	3			



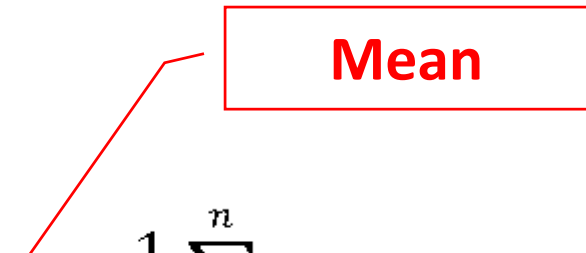
Mean vs. Variance

- Imagine two different classes take the same test
 - 1st period students score between 50 and 100 with $\mu = 75$
 - 2nd period students score between 70 and 80 with $\mu = 75$
- **Variance** (σ^2) is the average “distance” between each number in a set and the mean (μ) of that set
 - 1st period students have a greater **variance** in scores than 2nd period
 - Variance is a measure of **central tendency** – on average how close around the mean do all the numbers fall?
 - For every data point, we sum the **square** of the difference between the number and the mean. Then we divide that sum by the total number of data points

Mean vs. Standard Deviation

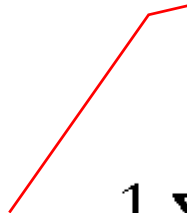


Mean, Variance, Standard Deviation


$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

Mean, Variance, Standard Deviation

Mean


$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

$$x = \{2, 9, 11, 5, 6\} \therefore n = 5$$

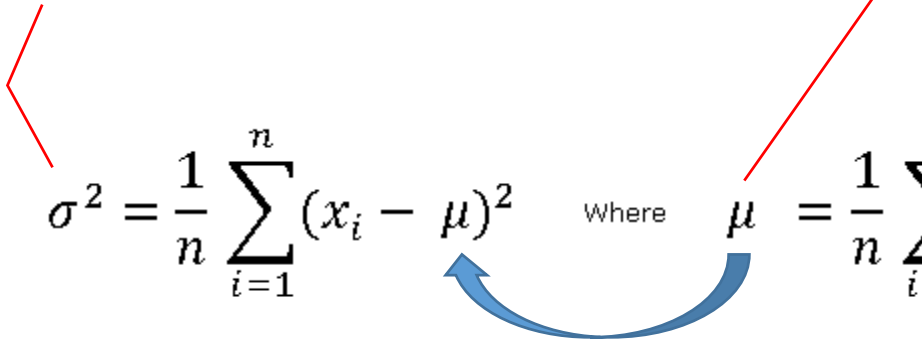
$$\sum_{i=1}^n x_i = (2 + 9 + 11 + 5 + 6) = 33$$

$$\mu = \frac{33}{5} = 6.6$$

Mean, Variance, Standard Deviation

Variance

Mean

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2 \quad \text{Where} \quad \mu = \frac{1}{n} \sum_{i=1}^n x_i$$


$$x = \{2, 9, 11, 5, 6\} \therefore n = 5, \mu = 6.6$$

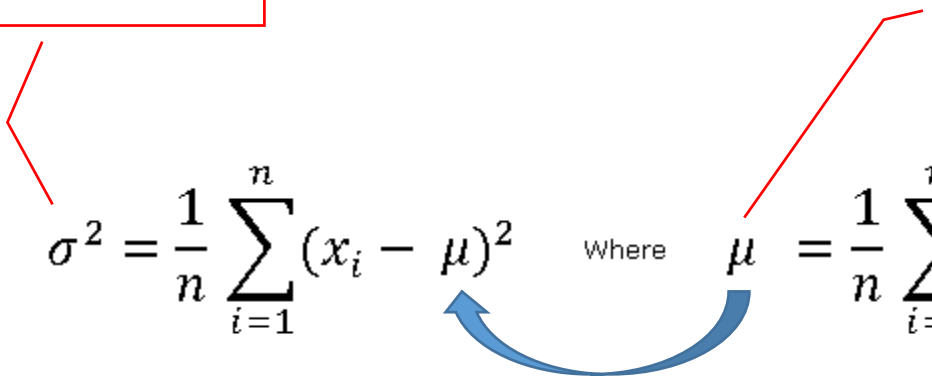
$$\sum_{i=1}^n (x_i - \mu)^2 = (2 - 6.6)^2 + (9 - 6.6)^2 + (11 - 6.6)^2 + \dots = 49.2$$


$$\sigma^2 = \frac{49.2}{5} = 9.84$$

Mean, Variance, Standard Deviation

Variance

Mean

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2 \quad \text{Where} \quad \mu = \frac{1}{n} \sum_{i=1}^n x_i$$


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


Standard Deviation

$$x = \{2, 9, 11, 5, 6\}$$

$$\sigma^2 = 9.84$$

$$\sigma = \sqrt{9.84} \cong 3.13$$

Pseudorandom Numbers

We will all use 2016 to verify
your code with mine

Generator
initialized to seed

```
seed_seq seed{ 2016 };  
default_random_engine generator{ seed };  
uniform_int_distribution<int> distribution1d20(3, 18);  
uniform_int_distribution<int> distribution1d6(1, 6);
```

Emits random
integers using a
uniform distribution

Sets low &
high range,
both inclusive

Pseudorandom Numbers

```
double CalcStdDev3d6(double totalRolls, double mean) {  
    double sum{};  
    for (double rollNumber{}; rollNumber <= totalRolls; ++rollNumber) {  
        int roll = distribution1d6(generator)  
            + distribution1d6(generator)  
            + distribution1d6(generator);  
        sum = sum + pow(roll - mean, 2);  
    }  
    double variance = sum / totalRolls;  
    double stdDev = sqrt(variance);  
    return stdDev;  
}
```

Calling the
distribution1d6()
function 3 times.
Each invocation moves the
generator to the next state

Pseudorandom Numbers

```
cout << "Total number of dice rolls: "  
    << setprecision(0) << fixed << totalRolls  
    << endl << endl;
```

```
double mean1d20 = CalcMean1d20(totalRolls);
```

```
double mean3d6 = CalcMean3d6(totalRolls);
```

```
distribution1d20.reset();
```

```
distribution1d6.reset();
```

.reset() returns the
distribution back to
the original seed value

```
double stdDev1d20 = CalcStdDev1d20(totalRolls, mean1d20);
```

```
double stdDev3d6 = CalcStdDev3d6(totalRolls, mean3d6);
```

Open Lab 1 – Hero Abilities

- Update a program to generate **1,000,000** hero ability scores, comparing the *mean* and *standard deviation* of the 1d20 versus the 3d6 dice roll methods
- In particular, write the missing code to correctly calculate the **CalcStdDev1d20** function
- Use **pow**(x, y) to calculate x^y
- **Which dice roll method would you want to use to generate your hero's abilities?**

Edit Lab 1 – Hero Abilities

```
hero-abilities.cpp x
1 // hero-abilities.cpp
2
3 #include "stdafx.h"
4
5 using namespace std;
6
7 seed_seq seed{ 2016 };
8 default_random_engine generator{ seed };
9 uniform_int_distribution<> distribution1d20(3, 18);
10 uniform_int_distribution<> distribution1d6(1, 6);
11
12 double CalcMean1d20(double totalRolls)
13 {
14     double sum{};
15     for (double rollNumber{}; rollNumber <= totalRolls; ++rollNumber) {
16         int roll = distribution1d20(generator);
17         sum = sum + roll;
18     }
19     double mean = sum / totalRolls;
20     return mean;
21 }
22
23 double CalcStdDev1d20(double totalRolls, double mean)
24 {
25     double sum{};
26     for (double rollNumber{}; rollNumber <= totalRolls; ++rollNumber) {
27         // Insert the correct code here
28     }
29     double variance = sum / totalRolls;
30     double stdDev = sqrt(variance);
31     return stdDev;
32 }
33
34
```



Edit Lab 1 – Hero Abilities

$roll = x_i$

$totalRolls = n$

$mean = \mu$

```
23 double CalcStdDev1d20(double totalRolls, double mean)
24 {
25     double sum{};
26     for (double rollNumber{}; rollNumber <= totalRolls; ++rollNumber) {
27         int roll = distribution1d20(generator);
28         sum = sum + pow(roll - mean, 2);
29     }
30     double variance = sum / totalRolls;
31     double stdDev = sqrt(variance);
32     return stdDev;
33 }
```

$sum = \sum (x_i - \mu)^2$

$variance = \frac{sum}{n} = \sigma^2$

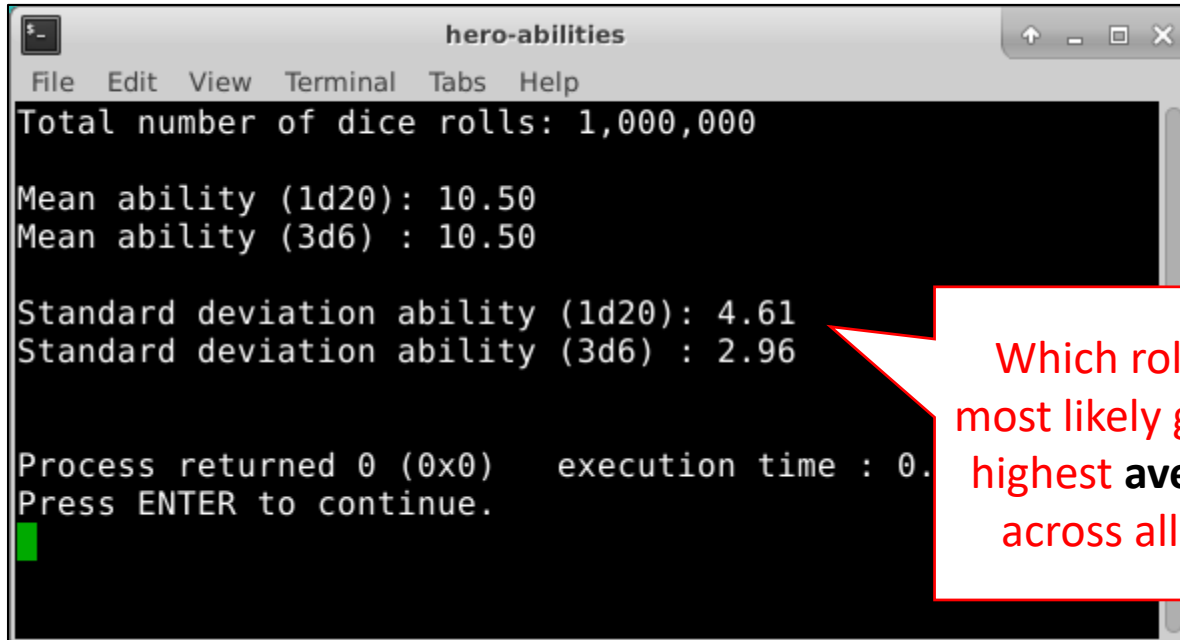
$stdDev = \sqrt{\sigma^2}$

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2 \quad \text{Where} \quad \mu = \frac{1}{n} \sum_{i=1}^n x_i$$

Run Lab 1 – Hero Abilities

```
23 double CalcStdDev1d20(double totalRolls, double mean)
24 {
25     double sum{};
26     for (double rollNumber{}; rollNumber <= totalRolls; ++rollNumber) {
27         int roll = distribution1d20(generator);
28         sum = sum + pow(roll - mean, 2);
29     }
30     double variance = sum / totalRolls;
31     double stdDev = sqrt(variance);
32     return stdDev;
33 }
```

Check Lab 1 – Hero Abilities



```
hero-abilities
File Edit View Terminal Tabs Help
Total number of dice rolls: 1,000,000
Mean ability (1d20): 10.50
Mean ability (3d6) : 10.50
Standard deviation ability (1d20): 4.61
Standard deviation ability (3d6) : 2.96
Process returned 0 (0x0)   execution time : 0.
Press ENTER to continue.
```

Which roll type will most likely give you the highest **average** score across all abilities?

Variance of Uniform Distributions

- Given an existing program that can:
 - Generate 15 sets of **random size** between 1 million & 2 million items
 - Within each set, every item is a random integer chosen within a range between a **lower limit** and an **upper limit**
 - The **lower limit** is a random number between **0 and 1000**
 - The **upper limit** is **2x** that set's lower limit + another random number between 0 and 1000
 - Calculate the mean (μ) and variance (σ^2) for each set

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2 \quad \text{Where} \quad \mu = \frac{1}{n} \sum_{i=1}^n x_i$$

Variance of Uniform Distributions

- Your assignment is to discover a magic number hidden in all uniform random number distributions
 - Calculate and display this “constant” for each set:

$$\text{magicNumber} = \frac{(\text{upperLimit} - \text{lowerLimit})^2}{\text{variance}}$$


- Is this number the same for ALL uniform distributions?
- Can we use this value to test if dice are loaded?



Edit Lab 2 – Variance of Uniform Distributions

```
VarianceUniformDistribution.cpp X
28
29     int lowerLimit = distLimits(generator);
30     int upperLimit = 2 * lowerLimit + distLimits(generator);
31
32     uniform_int_distribution<> distRange(lowerLimit, upperLimit);
33
34     double sum{};
35     for (int n{}; n < setSize; ++n)
36         sum += distRange(generator);
37     double mean = sum / setSize;
38
39     distRange.reset();
40
41     sum = 0;
42     for (int n{}; n < setSize; ++n)
43         sum += pow(distRange(generator) - mean, 2);
44     double variance = sum / setSize;
45
46     double magicNumber = 0;
47
48     cout << right << fixed
49         << setw(5) << setNumber
50         << setw(7) << lowerLimit
51         << setw(7) << upperLimit
52         << setw(12) << setSize
53         << setw(12) << setprecision(3) << mean
54         << setw(13) << setprecision(3) << variance
55         << setw(8) << setprecision(0) << magicNumber
56         << endl;
57 }
58
```

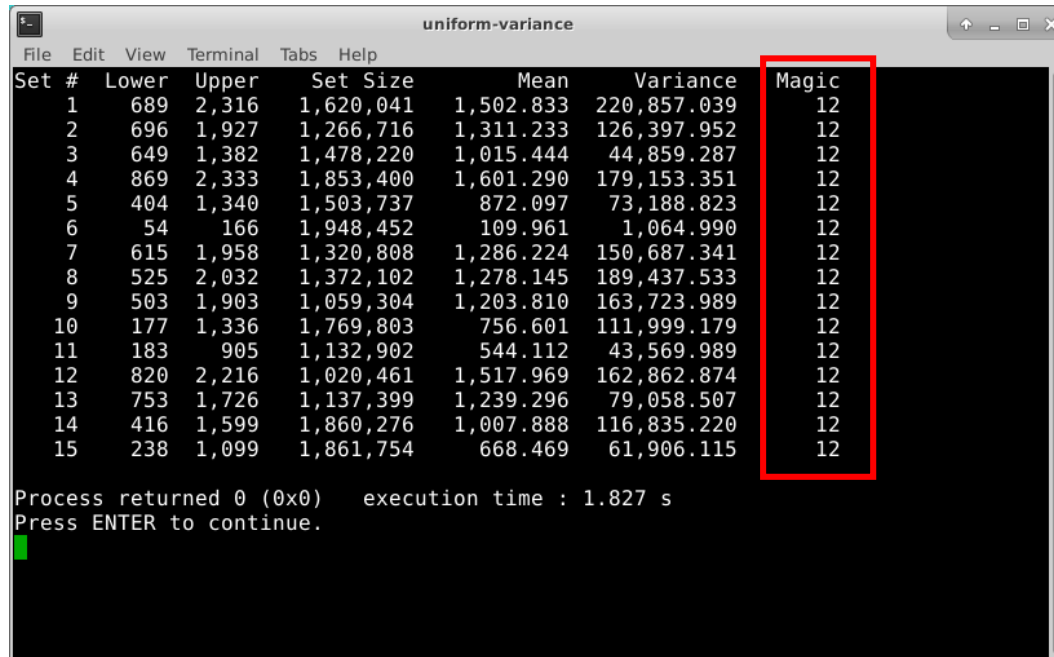
Fix this formula



Run Lab 2 – Variance of Uniform Distributions

```
VarianceUniformDistribution.cpp ✕
28
29     int lowerLimit = distLimits(generator);
30     int upperLimit = 2 * lowerLimit + distLimits(generator);
31
32     uniform_int_distribution<> distRange(lowerLimit, upperLimit);
33
34     double sum{};
35     for (int n{}; n < setSize; ++n)
36         sum += distRange(generator);
37     double mean = sum / setSize;
38
39     distRange.reset();
40
41     sum = 0;
42     for (int n{}; n < setSize; ++n)
43         sum += pow(distRange(generator) - mean, 2);
44     double variance = sum / setSize;
45
46     double magicNumber = pow(upperLimit - lowerLimit, 2) / variance;
47
48     cout << right << fixed
49         << setw(5) << setNumber
50         << setw(7) << lowerLimit
51         << setw(7) << upperLimit
52         << setw(12) << setSize
53         << setw(12) << setprecision(3) << mean
54         << setw(13) << setprecision(3) << variance
55         << setw(8) << setprecision(0) << magicNumber
56         << endl;
57 }
58
```

Check Lab 2 – Variance of Uniform Distributions



Set #	Lower	Upper	Set Size	Mean	Variance	Magic
1	689	2,316	1,620,041	1,502.833	220,857.039	12
2	696	1,927	1,266,716	1,311.233	126,397.952	12
3	649	1,382	1,478,220	1,015.444	44,859.287	12
4	869	2,333	1,853,400	1,601.290	179,153.351	12
5	404	1,340	1,503,737	872.097	73,188.823	12
6	54	166	1,948,452	109.961	1,064.990	12
7	615	1,958	1,320,808	1,286.224	150,687.341	12
8	525	2,032	1,372,102	1,278.145	189,437.533	12
9	503	1,903	1,059,304	1,203.810	163,723.989	12
10	177	1,336	1,769,803	756.601	111,999.179	12
11	183	905	1,132,902	544.112	43,569.989	12
12	820	2,216	1,020,461	1,517.969	162,862.874	12
13	753	1,726	1,137,399	1,239.296	79,058.507	12
14	416	1,599	1,860,276	1,007.888	116,835.220	12
15	238	1,099	1,861,754	668.469	61,906.115	12

Process returned 0 (0x0) execution time : 1.827 s
Press ENTER to continue.

- Every set had a different lower and upper limit, size, mean, and variance... yet the magic number was **12** for all of them!
- **Why** would Mother Nature pick the value 12 for this magic number? What is so special about 12? Why not pick a nice even 10?
- Boundless natural curiosity is what makes a good scientist...

Variance of Uniform Distributions

Moment Generating Function

$$\begin{aligned}\mu &= \int_a^b x \frac{1}{b-a} dx = \frac{1}{b-a} \left(\frac{x^2}{2} \Big|_a^b \right) = \frac{b^2 - a^2}{2(b-a)} \\ &= \frac{(b-a)(b+a)}{2(b-a)} = \frac{b+a}{2}\end{aligned}$$

$$\sigma^2 = \mathbb{E} (X - \mathbb{E}(X))^2$$

$$\sigma^2 = \mathbb{E} \left[X^2 - 2X\mathbb{E}(X) + (\mathbb{E}(X))^2 \right]$$

$$\sigma^2 = \mathbb{E}(X^2) - 2\mathbb{E}(X)\mathbb{E}(X) + \mathbb{E}(X)^2$$

$$\sigma^2 = \mathbb{E}(X^2) - 2\mathbb{E}(X)^2 + \mathbb{E}(X)^2$$

$$\sigma^2 = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$

$$\sigma^2 = \mathbb{E}(X^2) - \mu^2$$

Variance of Uniform Distributions

Moment Generating Function

$$\begin{aligned}\mu &= \int_a^b x \frac{1}{b-a} dx = \frac{1}{b-a} \left(\frac{x^2}{2} \Big|_a^b \right) = \frac{b^2 - a^2}{2(b-a)} \\ &= \frac{(b-a)(b+a)}{2(b-a)} = \frac{b+a}{2}\end{aligned}$$

$$\begin{aligned}E(X^2) &= \int_a^b x^2 \frac{1}{b-a} dx = \frac{1}{(b-a)} \left(\frac{x^3}{3} \Big|_a^b \right) \\ &= \frac{b^3 - a^3}{3(b-a)} = \frac{(b-a)(b^2 + ab + a^2)}{3(b-a)} = \frac{b^2 + ab + a^2}{3}\end{aligned}$$

$$\sigma^2 = \mathbb{E} (X - \mathbb{E}(X))^2$$

$$\sigma^2 = \mathbb{E} \left[X^2 - 2X\mathbb{E}(X) + (\mathbb{E}(X))^2 \right]$$

$$\sigma^2 = \mathbb{E}(X^2) - 2\mathbb{E}(X)\mathbb{E}(X) + \mathbb{E}(X)^2$$

$$\sigma^2 = \mathbb{E}(X^2) - 2\mathbb{E}(X)^2 + \mathbb{E}(X)^2$$

$$\sigma^2 = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$

$$\sigma^2 = \mathbb{E}(X^2) - \mu^2$$

Variance of Uniform Distributions

Moment Generating Function

$$\begin{aligned}\mu &= \int_a^b x \frac{1}{b-a} dx = \frac{1}{b-a} \left(\frac{x^2}{2} \Big|_a^b \right) = \frac{b^2 - a^2}{2(b-a)} \\ &= \frac{(b-a)(b+a)}{2(b-a)} = \frac{b+a}{2}\end{aligned}$$

$$\begin{aligned}E(X^2) &= \int_a^b x^2 \frac{1}{b-a} dx = \frac{1}{(b-a)} \left(\frac{x^3}{3} \Big|_a^b \right) \\ &= \frac{b^3 - a^3}{3(b-a)} = \frac{(b-a)(b^2 + ab + a^2)}{3(b-a)} = \frac{b^2 + ab + a^2}{3}\end{aligned}$$

$$\begin{aligned}\sigma^2 &= E(X^2) - \mu^2 = \frac{b^2 + ab + a^2}{3} - \left(\frac{b+a}{2} \right)^2 = \frac{4b^2 + 4ab + 4a^2 - 3b^2 - 6ab - 3a^2}{12} \\ &= \frac{b^2 - 2ab + a^2}{12} = \frac{(b-a)^2}{12}\end{aligned}$$

$$\sigma^2 = \mathbb{E} (X - \mathbb{E}(X))^2$$

$$\sigma^2 = \mathbb{E} \left[X^2 - 2X\mathbb{E}(X) + (\mathbb{E}(X))^2 \right]$$

$$\sigma^2 = \mathbb{E}(X^2) - 2\mathbb{E}(X)\mathbb{E}(X) + \mathbb{E}(X)^2$$

$$\sigma^2 = \mathbb{E}(X^2) - 2\mathbb{E}(X)^2 + \mathbb{E}(X)^2$$

$$\sigma^2 = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$

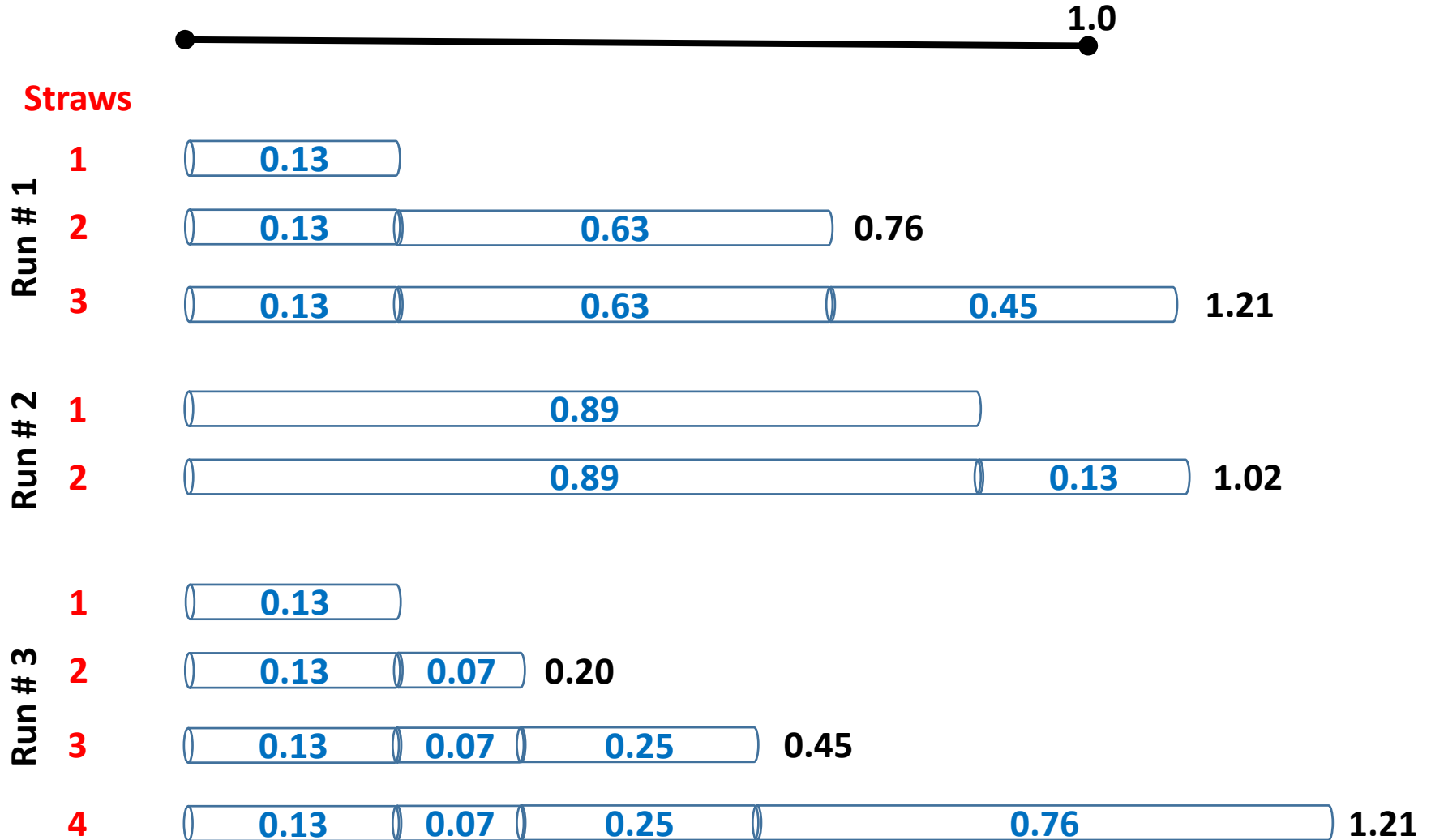
$$\sigma^2 = \mathbb{E}(X^2) - \mu^2$$

Random Straws

- Write a program to perform **ten million runs** of an experiment that places a varying number of straws *end-to-end* each run
- In each run, start with a single straw of **random** length between $0 \leq n < 1$
- Then enter a loop that keeps adding additional straws of **random** length ($0 \leq n < 1$) until the total length is > 1
- Find the **mean** number of straws added before the total length exceeds 1, across *all* million runs of the experiment




Random Straws



Edit Lab 3 – Random Straws

```
12 int main()
13 {
14     double maxIterations = 10000000;
15     double totalStraws = 0;
16
17     cout.imbue(locale(""));
18     cout << fixed << setprecision(0);
19     cout << "Performing " << maxIterations
20         << " experiments.." << endl;
21
22     for (double iteration = 0; iteration < maxIterations; iteration++) {
23         // Insert your code here
24     }
25
26     double meanStrawsPerUnitLength = totalStraws / maxIterations;
27
28     cout << setprecision(6) << endl
29         << "Mean straws per iteration: "
30         << meanStrawsPerUnitLength << endl;
31
32     cout << "Base of natural logarithm: "
33         << exp(1) << endl;
34
35     return 0;
36 }
37
```

Write code to perform each step of the experiment



Run Lab 3 – Random Straws

```
12  int main()
13  {
14      double maxIterations = 10000000;
15      double totalStraws = 0;
16
17      cout.imbue(locale(""));
18      cout << fixed << setprecision(0);
19      cout << "Performing " << maxIterations
20           << " experiments.." << endl;
21
22      for (double iteration = 0; iteration < maxIterations; iteration++) {
23          double straws = 1;
24          double length = distribution(generator);
25          while (length <= 1.0) {
26              straws++;
27              length += distribution(generator);
28          }
29          totalStraws += straws;
30      }
31      double meanStrawsPerUnitLength = totalStraws / maxIterations;
32
33      cout << setprecision(6) << endl
34           << "Mean straws per iteration: "
35           << meanStrawsPerUnitLength << endl;
36
37      cout << "Base of natural logarithm: "
38           << exp(1) << endl;
39
40      return 0;
41  }
42
```

Check Lab 3 – Random Straws

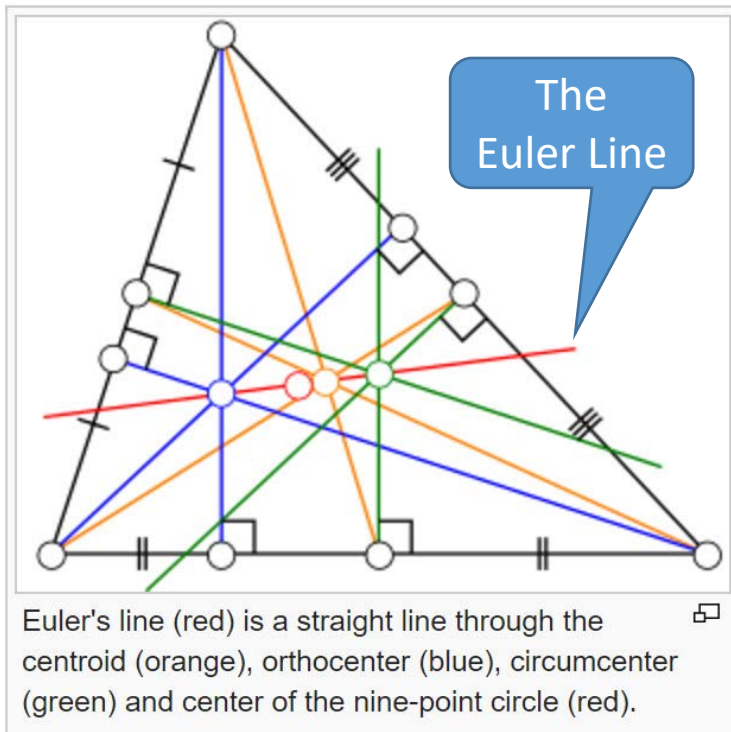
```
random-straws
File Edit View Terminal Tabs Help
Performing 10,000,000 experiments..
Mean straws per iteration: 2.718281
Base of natural logarithm: 2.718282
Process returned 0 (0x0)    execution time : 5.789 s
Press ENTER to continue.
█
```

$$e = \lim_{x \rightarrow \infty} \left(1 + \frac{1}{x}\right)^x = 2.718281828459...$$

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$



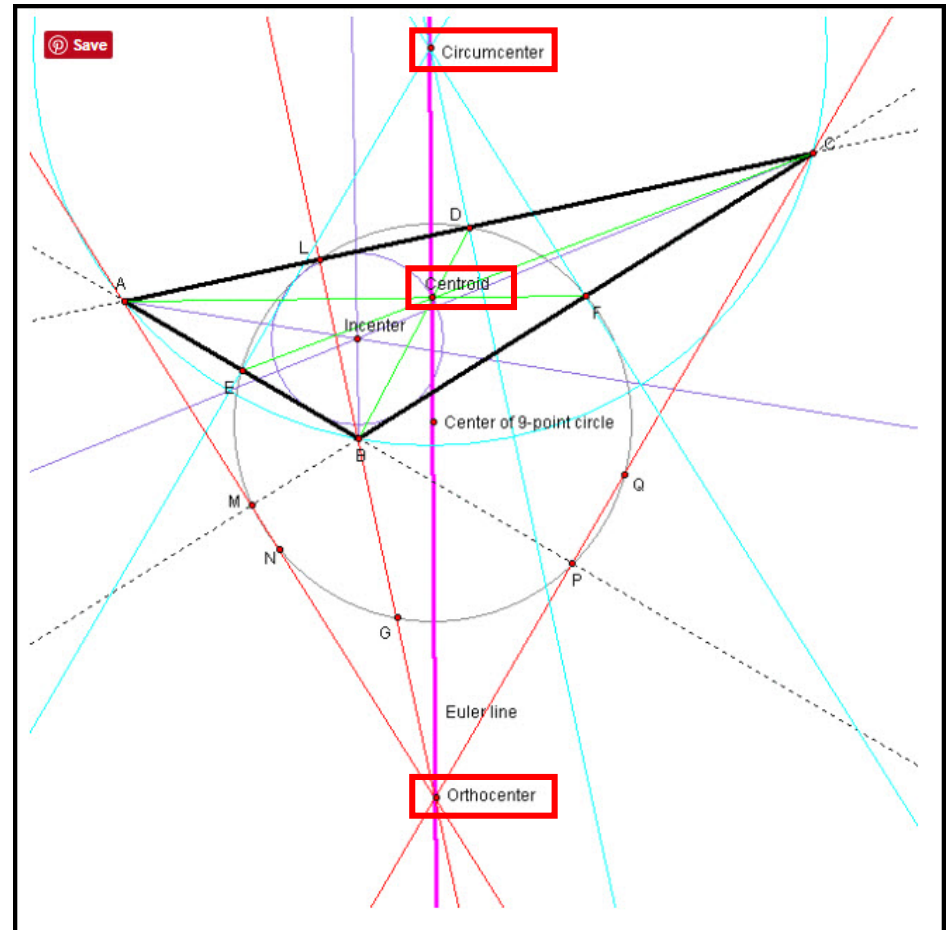
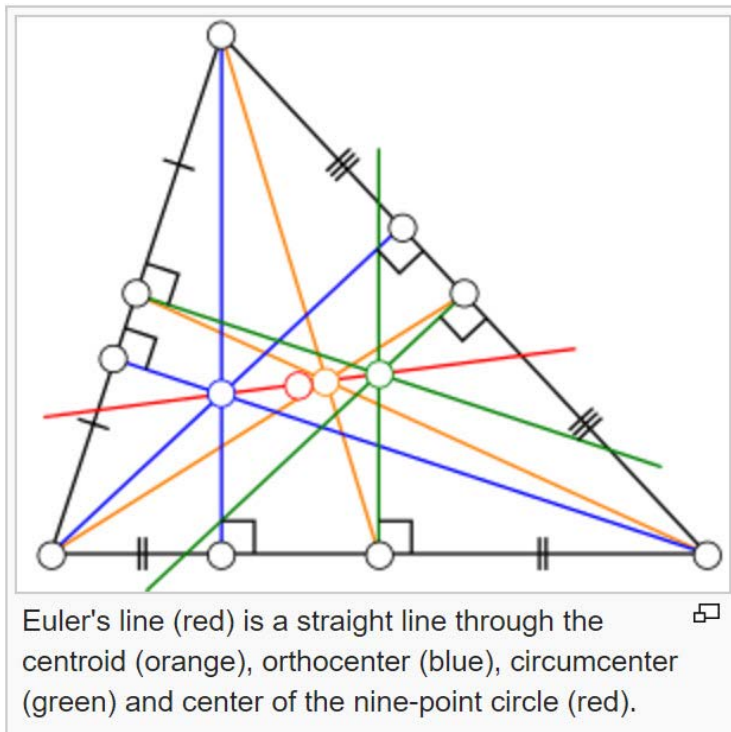
A Geometry Gem the Greeks Overlooked



- **Centroid** = center of mass
- **Circumcenter** = intersection of \perp *side* bisectors
- **Orthocenter** = intersection of the altitudes

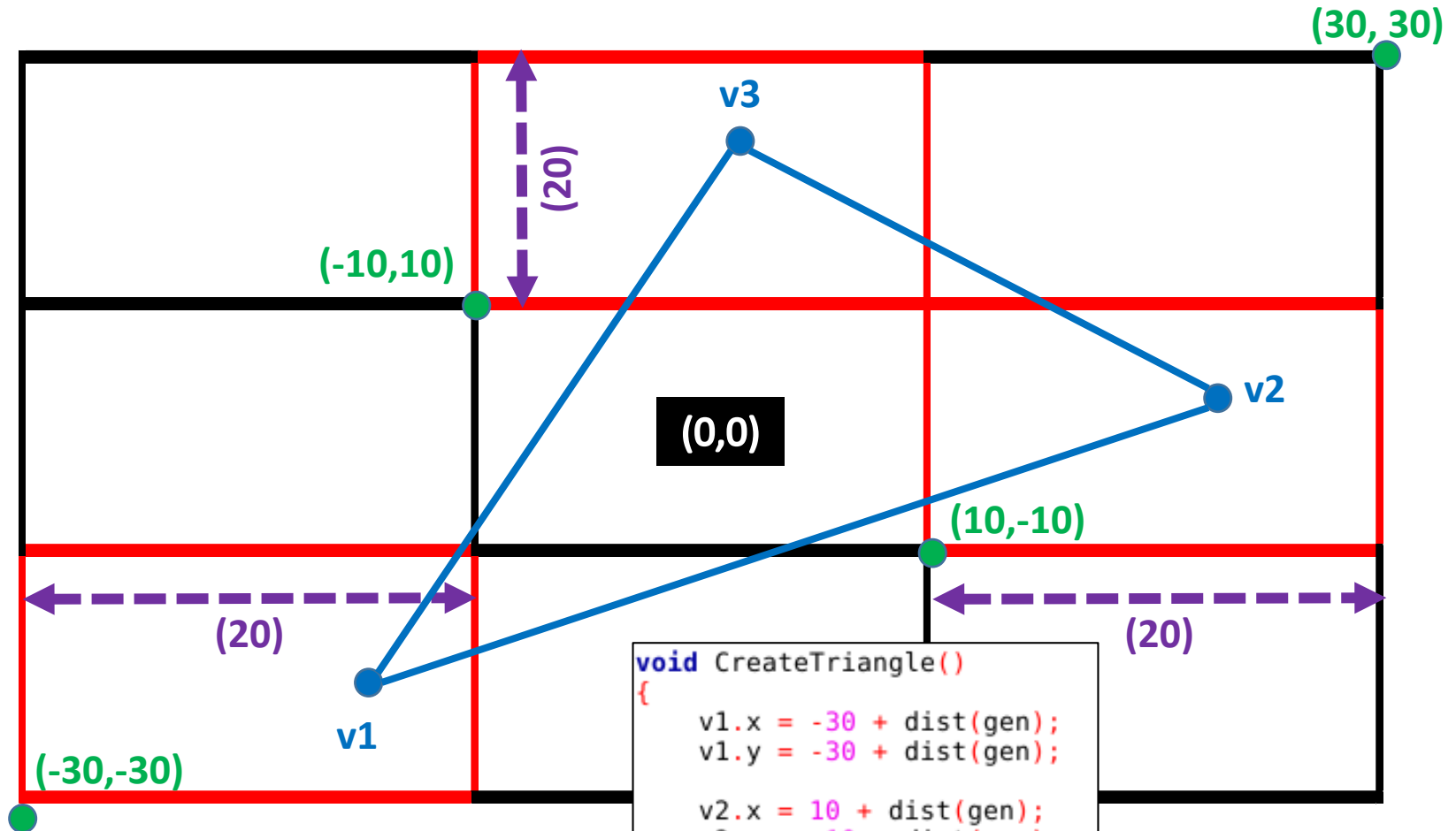
Euler was the first to realize and prove those **three** points are **always colinear** for any given triangle!

The Euler Line



```
seed_seq seed{ 2016 };
default_random_engine gen{ seed };
uniform_int_distribution<int> dist(0, 20);
```

The Euler Line



```
void CreateTriangle()  
{  
    v1.x = -30 + dist(gen);  
    v1.y = -30 + dist(gen);  
  
    v2.x = 10 + dist(gen);  
    v2.y = -10 + dist(gen);  
  
    v3.x = -10 + dist(gen);  
    v3.y = 10 + dist(gen);  
}
```

Open Lab 4 – Euler Line

```
void eventHandler(SimpleScreen& ss, ALLEGRO_EVENT& ev)
{
    if (ev.type == ALLEGRO_EVENT_KEY_CHAR) {
        if (ev.keyboard.keycode == ALLEGRO_KEY_N) {
            CreateTriangle();
            ss.Clear();
            ss.Redraw();
        }
    }
}

int main()
{
    SimpleScreen ss(draw, eventHandler);

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

    CreateTriangle();

    ss.HandleEvents();

    return 0;
}
```

Press the N key to
draw a new
random triangle

View Lab 4 – Euler Line

```
void draw(SimpleScreen& ss)
{
    // Connect the three vertices
    PointSet ps;
    ps.add(&v1);
    ps.add(&v2);
    ps.add(&v3);
    ss.DrawLines(&ps, "black", 2, true);

    // Calculate the slope of each side
    double slope12 = (v2.y - v1.y) / (v2.x - v1.x);
    double slope13 = (v3.y - v1.y) / (v3.x - v1.x);
    double slope23 = (v3.y - v2.y) / (v3.x - v2.x);

    // Calculate perpendicular slopes of each side
    slope12 = -1 / slope12;
    slope13 = -1 / slope13;
    slope23 = -1 / slope23;

    // Calculate the centroid
    Point2D centroid((v1.x + v2.x + v3.x) / 3,
                    (v1.y + v2.y + v3.y) / 3);

    // Connect vertices to centroid
    ss.DrawLine(v1, centroid, "orange", 3);
    ss.DrawLine(v2, centroid, "orange", 3);
    ss.DrawLine(v3, centroid, "orange", 3);
}
```

```
// Calculate side bisector points
Point2D bis12((v1.x + v2.x) / 2, (v1.y + v2.y) / 2);
Point2D bis13((v1.x + v3.x) / 2, (v1.y + v3.y) / 2);
Point2D bis23((v2.x + v3.x) / 2, (v2.y + v3.y) / 2);

// Calculate y-intercept of each perpendicular side bisector
double yint12 = bis12.y - slope12*bis12.x;
double yint13 = bis13.y - slope13*bis13.x;
double yint23 = bis23.y - slope23*bis23.x;

// Calculate the circumcenter
double ccx = (yint13 - yint12) / (slope12 - slope13);
double ccy = slope12*ccx + yint12;
Point2D cc(ccx, ccy);

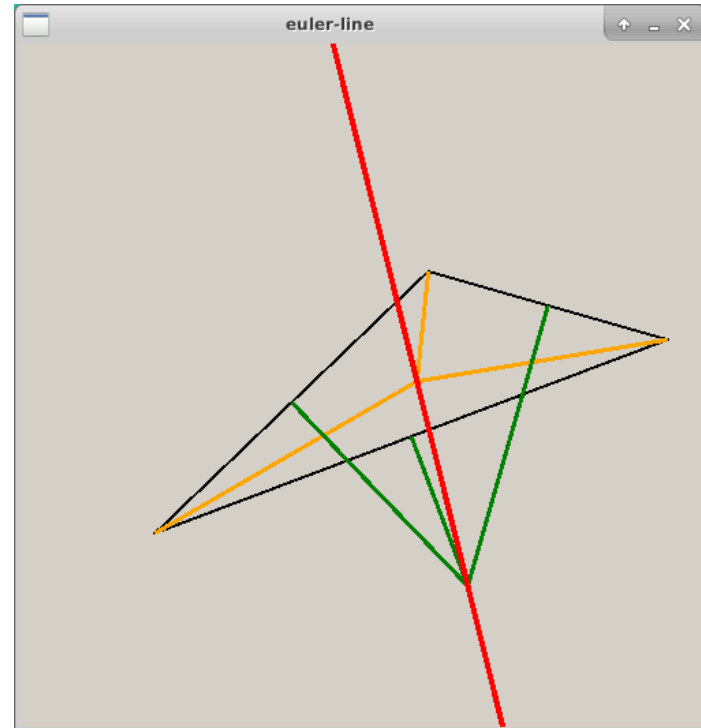
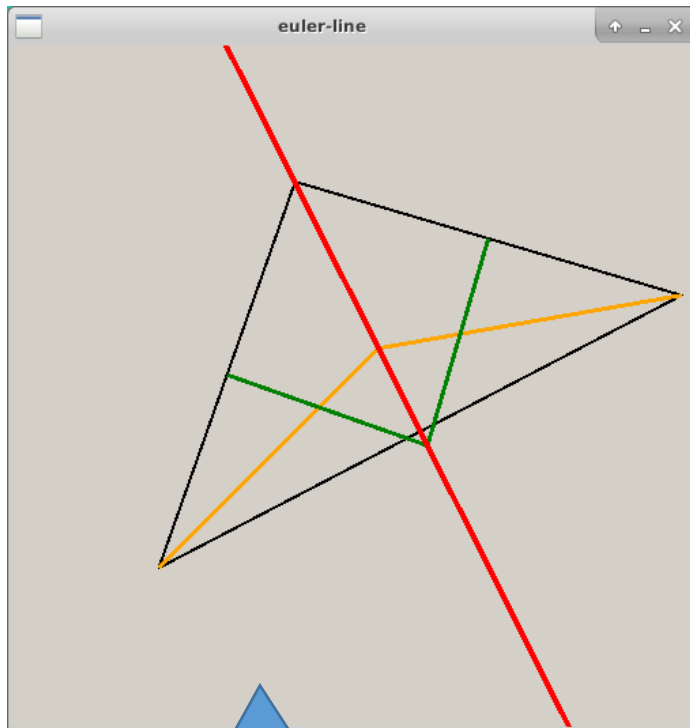
// Connect the side bisectors to the circumcenter
ss.DrawLine(bis12, cc, "green", 3);
ss.DrawLine(bis13, cc, "green", 3);
ss.DrawLine(bis23, cc, "green", 3);
```

```
// Calculate the point-slope of the Euler line
// connecting the centroid with the circumcenter
double slope_el = (centroid.y - cc.y) / (centroid.x - cc.x);
double yint_el = cc.y - slope_el*cc.x;

// Draw the Euler line
Point2D el1(-30, -30 * slope_el + yint_el);
Point2D el2(30, 30 * slope_el + yint_el);
ss.DrawLine(el1, el2, "red", 4);
}
```

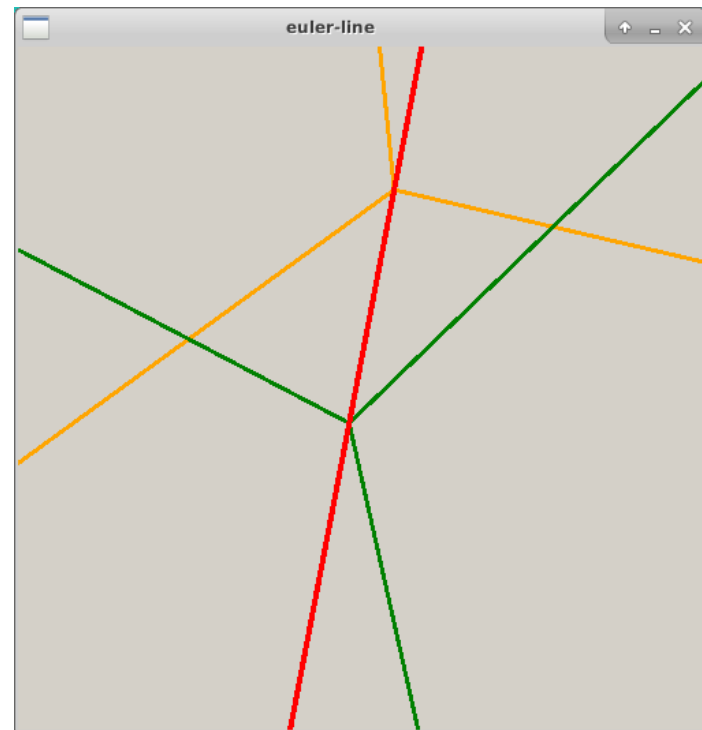
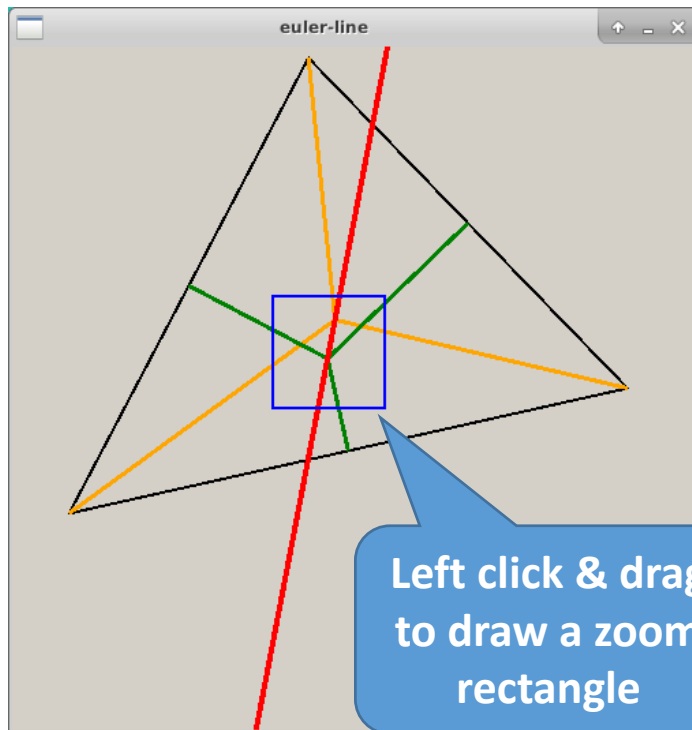
Note: This code only draws
the line connecting the
centroid and **circumcenter**

Run Lab 4 – Euler Line



Press the N key to
draw a new
random triangle

Check Lab 4 – Euler Line



Edit Lab 4 – Euler Line

- Recall Euler proved the **orthocenter** (the intersection of the altitudes) is also on that same **red** line
- The **orthocenter** is similar to the circumcenter, except instead of using the midpoint of each side, **we use the vertex opposite each side**, to find the point-slope form of each altitude
 - Use v_1 and the negative reciprocal of slope v_2v_3
 - Use v_2 and the negative reciprocal of slope v_1v_3
- Add code to **Lab 4** to calculate and draw the **orthocenter**, to visually confirm it falls on the same line formed from the centroid and circumcenter – **for all triangles!**

Now you know...

- How to calculate **Mean, Variance, Standard Deviation**
- Variance is just another average: it is the average distance between each data point and the mean μ of the entire set
- Create and call **custom functions** to organize code into named scopes having specific purposes
- A seemingly random process hides a different universal constant: ***e***
- Some statistics are only meaningful after generating a **very large** sample set – everything in scientific computing is **big**