• HW08 due **Friday 30 March** electronically by midnight
• HW grades nearly caught up!

• Join Piazza!
  – NB: It is easiest to join Piazza by first authenticating through NetBadge, accessing our UVaCollab site, and then connecting to Piazza from the link on the left-hand sidebar

• Office hours: all held in our computer lab, room 022-C of this bldg
  – Me: After lecture 3:30-4:30 every Tuesday
  – TAs:
    • Mondays: 3-5pm and 6-8pm
    • Wednesdays: 5-9pm
Mid Term Exam

• Exam had 44 questions, out of 63 points.
• One 3-pointer question was poorly written – I have thrown it out
  – Hence the exam is now out of 60 points

• Your grades for this exam are not yet ready 😞
Mid Term Exam

- Exam had 44 questions, out of 63 points.
- One 3-pointer question was poorly written – I have thrown it out
  - Hence the exam is now out of 60 points

- Your grades for this exam are not yet ready 😞
Review and Outline

• Last time:
  – More on arrays
  – Char arrays
  – More on passing arguments to main(..)
  – Random numbers

• Today:
  – More on random numbers
  – Structures
  – Making code reusable
  – Code libraries
  – Histograms
  – Plot making tools
An Example: Integration Using Random Numbers

Integrating Over an Arbitrary Area:

Say we have some horrible function in two dimensions, like the one shown below, and we want to calculate the area enclosed within the curve.

Define a box as shown, where
Y2 >= Max Y value of f(x,y) in the range X1 <= x <= X2
Y1 <= Min Y value of f(x,y) in the range Y1 <= y <= Y2

Area of the box = (X2-X1)*(Y2-Y1) > Area enclosed by the function
An Example: Integration Using Random Numbers

Using Random Numbers to Estimate the Area:

If we generate random points within the box, we can use them to get a better estimate of the area:

\[ r = \frac{\text{Points inside shape}}{\text{Total points}} \]

As we generate more and more points we asymptotically approach the exact answer.

```c
n_in = 0;
for (i=0 ; i< num_trials <i++) {
    x = (double)rand() / RAND_MAX * (X2-X1) + X1;
    y = (double)rand() / RAND_MAX * (Y2-Y1) + Y1;
    if (in_fcn(x, y)) n_in ++;
}

area = (X2-X1)*(Y2-Y1) * n_in / num_trials;
```
Monte Carlo Techniques: Integration

Programs that use random numbers are often referred to as “Monte Carlo” programs. The Monte Carlo method of integration isn't the best for every problem, but it's easy to implement. Even a chicken can do it:

Consider a circle in the dirt. Draw a box around the circle. Now allow a chicken to peck at will in and around your drawing.

By counting the total pecks and the pecks within the circle, we can estimate the circle’s area.
Monte Carlo Integration

The Whale Example:

$m = \text{Number of points in whale shape}$

$M = \text{Total number of points}$

$m / M \xrightarrow{M \to \infty} \frac{A_{\text{whale}}}{A_{\text{box}}}$

$A_{\text{whale}} \approx A_{\text{box}} \times \frac{m}{M}$

If we repeat this experiment with a different set of random points, our resulting estimate of the whale's area will be slightly different. By coming up with several estimates of the area, and looking at how much variation they display, we can get an idea of how accurate our estimate is.
Multiple Measurements

The Sample Mean:

Say we repeat the experiment N times. From our N experiments, we get a set of independent estimates of the area: \( A_1, A_2, A_3, \ldots, A_N \).

Each experiment contains M points. \( N \) = Number of experiments.

The mean (average) value will be:

\[
\overline{A} = \frac{1}{N} \sum_{i=1}^{N} A_i
\]
Advantages of Monte Carlo Integration:

- The Monte Carlo method can be trivially extended to higher dimensional problems.

- If it is at least possible to determine whether something is inside or outside of your area you can do the integral, even if the shape is beyond hope of integrating analytically.

- In general, it's more efficient to use other techniques to calculate integrals of 1D or 2D functions. But, for higher dimensional integrals, this technique is quite efficient compared to others.
Structures!
Let's say we wanted to store some census information about each of the fifty states.

There are several interesting facts about each state, but we can only store one fact in each variable. So we might choose to store the data in a bunch of parallel arrays, like this:

```c
int population[50];
double income[50];
double area[50];
double birthrate[50];
double deathrate[50];
```

This will work, but it's a little awkward. It would be nicer if we could bundle together all of the facts about a given state into one package.
In addition to the regular variable types like “int” and “double”, C lets us define our own custom-made types for variables, and pack multiple pieces of data into them.

For example, we could define a 50-element array called “state” that would hold all of our census data:

```
struct {
    int population;
    double income;
    double area;
    double birthrate;
    double deathrate;
} state[50];
```

“state” is of type “struct” (a data structure), and each element of the array will contain several pieces of related data.
Accessing Structure Elements: The "." Operator

You can refer to a particular piece of data in a struct by using the dot operator ("."): 

```c
struct {
    int population;
    double income; // Average/person/year.
    double area;    // In sq. miles.
    double birthrate; // Per year.
    double deathrate; // Per year.
} state[50];

state[0].population = 1234567;
state[0].income = 40280.0;
state[0].birthrate = 1280.5;
state[0].deathrate = 1280.1;
```
Using Multiple Equivalent Structures

What if we wanted to use the same data structure for other variables? Say, for example, we wanted to store census data for a group of 100 countries. We could just re-type the struct definition:

```c
struct {
    int population;
    double income; // Average/person/year.
    double area;   // In sq. miles.
    double birthrate; // Per year.
    double deathrate; // Per year.
} state[50];

struct {
    int population;
    double income; // Average/person/year.
    double area;   // In sq. miles.
    double birthrate; // Per year.
    double deathrate; // Per year.
} country[100];
```
Using Multiple Equivalent Structures

What if we wanted to use the same data structure for other variables? Say, for example, we wanted to store census data for a group of 100 countries. We could just re-type the struct definition:

```c
struct {
    int population;
    double income; // Average/person/year.
    double area;    // In sq. miles.
    double birthrate; // Per year.
    double deathrate; // Per year.
} state[5];

struct {
    int population;
    double income; // Average/person/year.
    double area;    // In sq. miles.
    double birthrate; // Per year.
    double deathrate; // Per year.
} country[100];
```

Tedious!
Instead of re-typing the struct, we could use `typedef` to define an alias for this struct:

```c
typedef struct {
    int population;
    double income; // Average/person/year.
    double area;  // In sq. miles.
    double birthrate; // Per year.
    double deathrate; // Per year.
} census;

census state[50];
census country[100];
```

With `typedef` we've created a new variable type called “census” and now we can use this to define variables, just like “int” or “double”.
More Examples of **typedef**

You don't need to use struct to use typedef. You can use typedef to define aliases for **any variable type**:

```c
//Define aliases for some types:
typedef double funds;
typedef double weight;
typedef int days;

//Use these aliases to define some variables:
funds bank_balance;
weight fish_per_month[12];
days til_christmas;
```

This may make it easier for you to **re-define your variables** later on. Say, for example, that you've made so much money that you now need to use a “long double” to count your fortune! If your program uses the “funds” type for all of your accounting variables, then you'll only need to change one line: the typedef statement that defines “funds”.

Pointers to Structures

We can have pointers to structs just like pointers to any other type of variable:

```c
typedef struct {
    int population;
    double income; // Average/person/year.
    double area;   // In sq. miles.
    double birthrate; // Per year.
    double deathrate; // Per year.
} census;

census states[50];
census *sptr;
sptr = states;

printf ("Pop = %d\n", states[0].population);
printf ("Pop = %d\n", (*sptr).population);
printf ("Pop = %d\n", sptr->population);
```

When using pointers, C gives us two ways to get data from a struct.
typedef struct {
    ...
} census;

census states[50];

for (i=0; i<50; i++) {
    clear_data( &states[i] );
}
...

void clear_data( census *s ) {
    s->population = 0;
    s->income = 0;
    s->area = 0;
    s->birthrate = 0;
    s->deathrate = 0;
}
typedef struct{
  double re, im;
} Complex;

double magnitude(Complex z) {
  return sqrt( z.re*z.re + z.im*z.im );
}

void conjugate(Complex *z) {
  z->im = -1.0*z->im;
}

int main() {
  Complex q;
  q.re = 12.;
  q.im = 23.;
  conjugate(&q);
  printf("q*=%f, %f; |q|=%f \n",
         q.re, q.im, magnitude(q));
}
Here are two different ways we could write the “magnitude” function:

**Pass a copy** of the complex structure to the function. Do calculation from the copied data.

```c
double magnitude(complex z) {
    return sqrt( z.re*z.re + z.im*z.im );
}
```

**Pass a pointer** to the complex structure to the function. Do calculation from the original data.

```c
double magnitude(complex *z) {
    return sqrt( z->re*z->re + z->im*z->im );
}
```

Generally more efficient: less data to move around.
Example: Structs and Functions

The gravity problem in HW09
Example: The Gravity Problem

The Problem:
We want to read in the position vectors, initial velocities, and masses of a bunch of objects. Then, using this data, we want to calculate the gravitational force on each object, due to the others. (For the first part, we'll ignore the initial velocities.)

Here's one of our objects. It has mass “m”, and it's located at position X. The calculated force on it is F.
Example: The Gravity Problem

Adding the Forces:

To find the total force on one mass, we just add the force vectors due to each of the other forces.

\[
\vec{F} = \vec{F}_b + \vec{F}_c + \vec{F}_d
\]
Example: The Gravity Problem

Finding Distance and Direction:

We'll need to know the distance and direction to each other object.

This is the vector from a to b:

$$ \vec{r} = \vec{X}_b - \vec{X}_a $$

The magnitude of this vector gives us the distance:

$$ r = |\vec{r}| $$

Once we know these, we can make a unit vector pointing from a to b:

$$ \vec{u} = \frac{\vec{r}}{r} $$
Example: The Gravity Problem

Calculating a Single Force:

Newton tells us that the magnitude of the gravitational force between two objects is:

\[ F = G \frac{m_a m_b}{r^2} \]

The force will point toward the other object, so the force vector will just be:

\[ \vec{F} = F \hat{u} \]
Example: The Gravity Problem

Data Structure:

To solve this problem programmatically, we'll first need a data structure to store information about each body:

```c
typedef struct{
    double s_vec[3];    // space(position) vector
    double v_vec[3];    // velocity vector
    double f_vec[3];    // force vector
    double mass;
} body;

const int MAX_BODIES = 100;
body bodies[MAX_BODIES];   // array of bodies
```
Example: The Gravity Problem

Reading Data from a File:

```c
int read_data(char* file, body *bodies){
    int num=0; // number of entries read from file
    int status;
    FILE *file_p = fopen(file,"r");

    while(num<MAX_BODIES) {
        status=fscanf(file_p,"%lf %lf %lf %lf %lf %lf %lf", 
                        &bodies[num].s_vec[0],
                        &bodies[num].s_vec[1],
                        &bodies[num].s_vec[2],
                        &bodies[num].v_vec[0],
                        &bodies[num].v_vec[1],
                        &bodies[num].v_vec[2],
                        &bodies[num].mass);
        if (status==EOF) break;
        num++;
    }
    return num;
}
```
You will also need to write functions for:
- calculating the distance between objects
- the difference between objects’ positions, using vectors
- calculating the total force on each object
- calculating the center of mass for the array of objects
Making Pieces of Code that Are Reusable
Reusable Code

• Many tasks in computer programming for scientific purposes are confronted over and over again
  – imagine calculating the sine of some angle or the distance between two points in some linear algebra project
  – imagine every time you needed that value, having to write a piece of new code to do the calculation

• There is point in reinventing the wheel every time

• Utilities such as these – and many others we can think of – are best coded once and reused over and over again
  – reduction of chance for bugs
  – easier debugging
  – portability of code to other users
Reusable Code: Option I

Using `#include` to Re-use Code:

File “sqrtn.cpp”:
```c
double sqrtn(double x) {
    double guess = x/2.0;
    while (fabs(guess*guess - x) > 1e-6)
        guess = (guess + x/guess)/2;
    return guess;
}
```

File “main.cpp”:
```c
#include <stdio.h>
#include <math.h>
#include "sqrtn.cpp"
int main() {
    double x;
    printf(“enter a number \n”);
    scanf("%lf", &x);
    printf("sqrt(\%lf) = \%lf\n", x, 
sqrtn(x));
}
```

Note the use of quotes around the file name. Angle brackets (<> are reserved for files in “standard” system directories. In general your personal includes must give the full directory path to the file, or be in the current directory.
Reusable Code: Option I

Using \#include to Re-use Code:

File “sqrtn.cpp”:

double sqrttn(double x) {
    double guess = x/2.0;
    while
        return
    }

File “main.cpp”:

#include
#include
#include
#include
int main
    double
    printf("enter a number\n");
    scanf("%lf", &x);
    printf("sqrt(%lf) = %lf\n", x,
        sqrttn(x));

Drawback of this Option:

You have to compile the extra code every time!
Creating Object Files:

File “sqrtn.cpp”:
```c
#include <math.h>
double sqrtn(double x) {
  double guess = x/2.0;
  while (fabs(guess*guess-x) > 1e-6) {
    guess = (guess + x/guess)/2;
  }
  return guess;
}
```

An object file is compiled code that hasn't been fully processed into a program. The above code isn't a complete program.

We can compile the code into an object module as follows, using the “-c” flag of g++:

```
g++ -O -Wall -c sqrtn.cpp
```

This creates the file sqrtn.o.

sqrtn.o contains the function's code translated into CPU instructions. The -c flag causes g++ to stop after compiling, without continuing to the “linking” step that produces a runnable program.
Linking Object Files with Your Program:

If we have a pre-compiled object file, we only need to `#include` a header file containing the prototype for the function:

File “sqrtn.hpp”:

```cpp
double sqrtn(double x);
```

File “main.cpp”:

```cpp
#include <stdio.h>
#include <math.h>
#include “sqrtn.hpp”
int main(){
  double x;
  printf(“enter a number\n”);
  scanf(“%lf”, &x);
  printf(“sqrt(%lf) = %lf\n”, x, 
  sqrtn(x));
}
```

We can then compile our main program by typing:

```bash
g++ -o main main.cpp sqrtn.o
```
Code Libraries
What is a Code Library?

• A code library is a collection of pre-compiled functions that one can use as needed plugging into newly developed pieces of code
  – Typically contain oft-used utilities that are convenient to just simply re-use rather than re-code

• We have been using code libraries all the time!
  – functions contained in stdio.h, stdlib.h, math.h, etc.

• We will learn how to make our own code library here!
To build a library, first make object files (as we have done before):

```
    g++ -O -Wall -c hist.cpp
    g++ -O -Wall -c random.cpp
```

Next combine the object files into a library:

```
    ar -csr libp2660.a hist.o random.o
```

The `archive` command is used to create your library, the syntax we will use is:

```
    ar -csr lib<name>.a file1.o file2.o ...
```

You can list the contents of your library with a command like:

```
    ar -t libp2660.a
```

```
    hist.o
    random.o
```
Example Using Code from a Library

```c
// Header files for your library
#include "random.hpp"
#include "hist.hpp"

int main(){
    hl myHist;
    // Set range for histogram's x-axis
    hlinit( &myHist, 0, 100);
    for (int i=0; i<1000; i++) {
        // Fill the histogram w/ 100 data points
        // from the function randn:
        hlfill(&myHist, randn(50,10));
    }
    // Plot the histogram to the screen
    hlplot(&myHist, "");
    return 0;
}
```
To use your library with a program:

1) make sure your program includes header files defining the functions you use

2) tell the linker how to find your library

Let's say your program file is called test_hist.cpp. You would build the program as follows:

```
g++ -O -Wall test_hist.cpp -o test_hist -L. -lp254
```

- `-L.` specifies a new directory to search for library files (here we add `.` for the current directory, to the library search path)

- `-l` (small “L”) gives the name of a library (libp254.a) to search for object files needed to complete your program. Note that the “lib”/“.a” prefix/suffix is omitted from the command
In general the header files and libraries will not be located in your current working directory, so for more complex programs the build command could be of the form:

```
g++ -O -Wall \ 
-I<include_dir1> -I<include_dir2> \ test_hist.cpp 
-o test_hist \ 
-L<lib_dir1> -L<lib_dir2> \ 
-l<lib1> -l<lib2> -l<lib3>
```
Data Visualization: Histograms

and

An example of a code library, including structures!
What is a Histogram?

A histogram shows the distribution of data by dividing it up into discrete intervals and counting the data points that fall within each interval.
Elements of a Histogram

Age of Laptop Computer (Years)
Elements of a Histogram

- range
- number of bins
- number of total entries
- number of entries in each bin
- number outside range
Why Are Histograms So Useful?

Reducing Data:

Histograms can reduce large quantities of data into a manageable summary.

Here we've divided the range from 0 to 100 into 50 bins.

We then generated 1000 random numbers and counted how many fell into each bin.

So, with just 50 numbers (the counts in each bin) we have a summary that shows us the distribution of our 1000 random numbers. We could go on and generate millions of numbers, and we'd still be able to summarize them in the same histogram, using only 50 counters.
Histograms in C as a Structure

Here's a structure we could use to store histogram data. At the heart of it is an array of counts.

```c
#define HBINS 50
typedef struct {
    double h_array[HBINS];
    double xmin, xmax;
    int entries;
    int under_flow, over_flow;
    double sumx, sumx2;
} h1;
```

- These histograms will have a fixed number of bins.
- Array of bin counts.
- Min/max of x range.
- Total # of counts.
- Counts outside range.
- Sums of values and squares of values, for calculating statistics.
Operations With / On Histograms

- **create / initialize:** Set range for histogram, etc.
- **reset:** Clear bin contents to 0.
- **fill:** Add a data value to the histogram.
- **dump:** Print contents of histogram.
- **plot:** Graphically display the histogram.
Histogram Operations: Function Prototypes

/* initialize hist. Set min/max limits for the histogram */
void hlinit(hl *hist, double xmin, double xmax);

/* add a data point to a histogram */
void hlfill(hl *hist, double x);

/* dumps hist to screen (filename="") or to a file "filename". Returns 0 for success, 1 for error */
int hldump(hl *hist, char *filename);

/* calculate and return statistics for a histogram
 input: hl *hist
 output: int *entries, double *mean, double *std_dev */
void hlstats(hl *hist, int *entries, double *mean, double *std_dev);

/* plot a histogram to the screen (filename="") or a graphics file "filename" */
void hlplot(hl *hist, char *filename);

We will use these functions in lab this week!
Reset and initialize:

```c
void hlreset(h1 *hist){
    int i;
    hist->entries=0;
    hist->sumx=0;
    hist->sumx2=0;
    hist->over_flow=0;
    hist->under_flow=0;
    for (i=0; i<HBINS; i++) hist->h_array[i]=0;
}

void hlinit(h1 *hist, double min, double max){
    hist->xmax = max;
    hist->xmin = min;
    hlreset(hist);  // clear all storage variables
}
```
void h1fill(h1 *hist, double x){
    int bin=0;
    double binsize, lowedge;

    if (x < hist->min) hist->under_flow++;
    else if (x >= hist->max) hist->over_flow++;
    else {
        binsize = (hist->max - hist->min) / HBINS;
        lowedge = hist->min; // low edge of 1st bin
        while (fabs(x-lowedge) > binsize) {
            bin++;
            lowedge += binsize; // move to next bin
        }
        hist->h_array[bin]++; //increment the appropriate bin
    }
    hist->entries++;
    hist->sumx += x;
    hist->sumx2 += x*x;
}
How to Make a Histogram

• Biggest mistake made by young scientists in my experience:
  – They have taken some nice data, they maybe have some model that they are testing, done a good job
  – They want to convey the message from their experiment through some visualization of their data, a plot
  – And then they make a plot like this:
How to Make a Histogram
How to Make a Histogram
How to Make a Histogram
How to Make a Histogram
How to Make a Histogram
How to Make a Histogram
How to Make a Histogram
How to Make a Histogram
Statistics: Our First Entrypoint
Making a Measurement: Truth

• Let’s say I want to measure some physical observable.
• Examples:
  – the acceleration due to gravity near the Earth’s surface
  – the speed of light
  – the mass of the Higgs boson
• Each such measureable has a single value – the truth:

\[ \text{N repeated measurements of some quantity } x. \]

A perfect apparatus and data collection scheme would yield the true value every time!
Why do we see a range of measured values?
- imperfections in our instruments
- limitations in our measurements
How do we go from to arrive at?

Use Statistics:

The best we can achieve is the center distribution. Its shape depends on the measurement’s **uncertainty**.
The Story of Measurement

• A measurement is not about the central value one finds
• A measurement is truly about its **uncertainty**
  – The central value in fact is MEANINGLESS and OF NO USE without understanding and reporting the associated uncertainty

• Two types of uncertainty:
  – **Systematic uncertainty**: Features of the measurement device or technique that shift (aka “bias”) the measured result away from the true value
    • Biases can often be corrected if discovered
    • Eg: Imagine you are measuring athletes running the 40-yard dash. You discover however the length they run is in fact 42 yards long. This can in principle be corrected
  – **Statistical or random uncertainty**: Features of the measurement device or technique that shift the measured result by a different amount in each attempt (hence, random)
“Uncertainty” and “error” are used interchangeably, often inappropriately.

I will try to use “uncertainty” universally – and you should as well.
The Gaussian Distribution

Measures of random processes, in the limit of many samples often tend to produce a characteristic Gaussian or Normal distribution. (The classic “Bell Curve”.)

$\mu =$ “mean”, average value of the distributions

$\sigma^2 =$ “variance”, characterizes the width

$\sigma =$ “standard deviation”

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$
The Gaussian Distribution

Measures of random processes, in the limit of many samples often tend to produce a characteristic Gaussian or Normal distribution. (The classic “Bell Curve”.)

Central Limit Theorem:
Sufficiently large number of randomly distributed measurements always approaches a Gaussian distribution.

$\mu$ = “mean”, average value of the distributions
$\sigma^2$ = “variance”, characterizes the width
$\sigma$ = “standard deviation”
The red curve represents the probability of observing a particular value. This is the Parent Distribution from which our measurements will be drawn. Our experiment draws measurements, at random, from this. These measurements form our Sample Distribution.

In the limit of many samples our Sample Distribution can approach the Parent Distribution.
What Can We Get From the Sample Distribution?

Our best estimates of the mean and variance of the Parent Distribution come from the mean and variance of the Sample Distribution. This is something we can study with appropriate programs.

We can determine the parameters of the Sample Distribution, such as its mean ($\bar{x}$) and standard deviation ($s$).

Note: variance = $s^2$

From these parameters we can make statements about the parent distribution.
To clearly distinguish between parent and sample distributions, we use $\mu$ and $\sigma$ for the Parent Distribution and $\bar{x}$ and $s$ for the Sample Distribution.

Most probable value = $\mu$ (the mean)

$P(x) = \text{Probability of seeing a value between } x \text{ and } x + dx.$

68% of Area

68% probability of seeing a value between +/- $\sigma$ of the mean.

This is a probability distribution, so the total area under the curve is equal to 1. (There's a 100% probability that something happened.)

$$\int_{-\infty}^{\infty} P(x) \, dx = 1$$
The Normal distribution is a **Probability Density Function** (PDF), so the total area under the curve must be equal to 1. One way to see this is to look at the integral of the Normal distribution. The integral of a PDF is called a **Cumulative Distribution Function** (CDF).

The CDF tells us the probability of observing a value less than or equal to $x$. 

The CDF approaches 1 as $x$ increases.
Statistical vs Systematic Uncertainties

Systematic errors (aka “biases”) must be corrected with expert knowledge of your apparatus and procedures. Statistics will not improve a mis-calibrated detector or a flawed design for your experiment!

- **True Value**
- Systematic errors might shift our results away from the true values we want to measure.

- **Distribution of Observed Values**

Perhaps our clock is running fast? Maybe our yardstick is worn down?
Recall: Monte Carlo Integration

The Whale Example:

- **m**: Number of points in whale shape
- **M**: Total number of points

\[
\frac{m}{M} \xrightarrow{M \to \infty} \frac{A_{\text{whale}}}{A_{\text{box}}}
\]

\[
A_{\text{whale}} \approx A_{\text{box}} \times \frac{m}{M}
\]

If we repeat this experiment with a different set of random points, our resulting estimate of the whale's area will be slightly different. By coming up with several estimates of the area, and looking at how much variation they display, we can get an idea of how accurate our estimate is.
Recall: Multiple Measurements

The Sample Mean:

Experiment 1  Experiment 2

Experiment 3  Experiment 4

Say we repeat the experiment N times. From our N experiments, we get a set of independent estimates of the area:

$A_1, A_2, A_3, ..., A_N$

Each experiment contains M points. $N$ = Number of experiments.

If we made a histogram of our estimates, it might look like this:

The mean (average) value will be:

$$
\overline{A} = \frac{1}{N} \sum_{i=1}^{N} A_i
$$
The Standard Deviation:

Each of our \(A_i\) values will deviate from the true value \((A_{\text{whale}})\) by some amount, \(d_i\).

Since these deviations may be positive or negative, the average deviation will tend toward zero.

How, then, can we come up with an estimate for a “typical” deviation? We can sum the squares of the \(d_i\) values, instead!

We define the “variance” \((\sigma^2)\) as the average squared deviation from the true value:

\[
\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} d_i^2 = \frac{1}{N} \sum_{i=1}^{N} (A_i - A_{\text{whale}})^2
\]

Its square root, \(\sigma\), is called the “Standard Deviation” or “Standard Error”. It’s also sometimes called the Root-Mean-Square (RMS) deviation, for obvious reasons.
Multiple Measurements

The Standard Deviation:

Important vocabulary:
- mean
- variance
- standard deviation or RMS

The standard deviation has an intimate relationship to the definition of the Gaussian (aka normal) distribution.

Here the standard deviation of the sample of measurements is usually the best estimate of the standard deviation of the parent distribution.
The standard deviation tells us about the distribution of our $A_i$ values. In particular, if the values follow a “normal” (gaussian) distribution, it tells us that we should expect that about 68% of our $A_i$ values will fall within +/- $\sigma$ of the true value of the area.
The Importance of the Gaussian Distribution

• If plotted, our measurements will look approximately like a Gaussian Distribution.
  – Is this magic?
  – No!

• Recall the Central Limit Theorem, paraphrased
  – Any collection of random measurements of some variable will resemble a Gaussian
The Importance of the Gaussian Distribution

• If plotted, our measurements will look approximately like a Gaussian Distribution.
  – Is this magic?
  – No!

• Recall the Central Limit Theorem, paraphrased
  – Any collection of random measurements of some variable will resemble a Gaussian

• Back to the whale:
  – Our random measurements here are the results of MC integration of the whale shape
  – An infinite number of such measurements would be a perfect Gaussian
But…What about the True Value?

The Sample Standard Deviation:

The problem with computing $\sigma$ is that we don't know the true value, $A_{\text{whale}}$. Our best approximation of this value is the average of all of our $A_i$ measurements, $\bar{A}$.

But, it can be demonstrated that just plugging $\bar{A}$ in place of $A_{\text{whale}}$ will systematically underestimate the value of sigma.

We correct for this by dividing by $N-1$ instead of $N$, and arrive at the following approximation:

$$s^2 = \frac{1}{N-1} \sum_{i=1}^{N} (A_i - \bar{A})^2 \approx \sigma^2$$

We call $s$ the Sample Standard Deviation.
The Sample Standard Deviation:

The problem with computing $\sigma$ is that we don't know the true value, $\bar{A}$. The change to using $N-1$ is called Bessel's Correction. Read more here.

What happens below as $N$ gets very large?

$$s^2 = \frac{1}{N-1} \sum_{i=1}^{N} (A_i - \bar{A})^2 \approx \sigma^2$$

We call $s$ the Sample Standard Deviation.
For purposes of writing computer programs, it's useful to rearrange our expression for $s^2$ a little. Here are our expressions for $s^2$ and the average, $\bar{A}$:

$$s^2 = \frac{1}{N-1} \sum_{i=1}^{N} (A_i - \bar{A})^2$$

$$\bar{A} = \frac{1}{N} \sum_{i=1}^{N} A_i$$

Substituting the expression for $\bar{A}$ into the first equation, we can collect terms and arrive at this:

$$s^2 = \frac{1}{N-1} \left[ \sum_{i=1}^{N} A_i^2 - \frac{1}{N} \left( \sum_{i=1}^{N} A_i \right)^2 \right]$$

This is easy to code, because our program just needs to do one loop, and keep two sums: the sum of the values and the sum of their squares.
Uncertainty on the Mean

We said that our best guess of the true value, $A_{\text{whale}}$, is the mean of our sample values, $\bar{A}$. Now let's get back to the problem of quantifying just how good that guess is. Earlier, we defined the mean as:

$$\bar{A} = \frac{1}{N} \sum_{i=1}^{N} A_i = \frac{1}{N} (A_1 + A_2 + A_3 \ldots)$$

If we know the standard deviations of the $A_i$ values, we can use them to calculate the standard deviation of the mean, through the regular propagation of errors process:

$$\sigma_f^2(x, y, z \ldots) = \sigma_x^2 \left( \frac{\partial f}{\partial x} \right)^2 + \sigma_y^2 \left( \frac{\partial f}{\partial y} \right)^2 + \sigma_z^2 \left( \frac{\partial f}{\partial z} \right)^2 \ldots$$

Where we treat $\bar{A}$ as a function of the variables $A_1, A_2, A_3 \ldots A_N$. 


Uncertainty on the Mean

Applying this to our expression for $\bar{A}$, we get:

$$\sigma_{\bar{A}}^2 = \frac{\sigma_1^2}{N^2} + \frac{\sigma_2^2}{N^2} + \frac{\sigma_3^2}{N^2} + \ldots$$

What are the $\sigma_i$ values? They're the estimates of how far each $A_i$ value typically deviates from the true value. In our case, the values for $\sigma_i$ will all be equal to $\sigma$, the standard deviation of our collection of $A_i$ values. So, we can simplify the equation above and write:

$$\sigma_{\bar{A}}^2 = \frac{1}{N} \sigma^2$$

or

$$\sigma_{\bar{A}} = \frac{\sigma}{\sqrt{N}}$$

Our uncertainty in $\bar{A}$ decreases like the $\sqrt{N}$ as $N$ increases. For example, to get a 10-times better value for $\bar{A}$, we need to do 100 times as many measurements. Also, because we don’t know the exact value of $\sigma$, we say that $\sigma_{\bar{A}} \approx s/\sqrt{N}$. 
Confidence Levels

When we cite an experimental result, we also state the uncertainty in the result. So, in our whale-shape example, we might give our final estimate of the area as $\bar{A} +/\sigma_{\bar{A}}$ (for example “1.12 +/- 0.01”).

**What we mean:**

“There's a 68% probability that our average value, $\bar{A}$, lies within a distance +/- $\sigma_{\bar{A}}$ from the true value, $A_{\text{whale}}$.”

Probability of getting a given value of $\bar{A}$.

---

68% of Area

$\bar{A} - \sigma_{\bar{A}}$  \hspace{1cm} $\bar{A} + \sigma_{\bar{A}}$
Back to the Whale!

Number of Points per Experiment:

Experiment 1  Experiment 2

Experiment 3  Experiment 4

Each experiment contains $M$ points.

$N = \text{Number of experiments.}$

What about "$M$", the number of points in each experiment? So far, everything we've said has been independent of $M$.

Fewer points will increase the uncertainty in each $A_i$ value ($\sigma$). But, since $\sigma_{\bar{A}} = \sigma / \sqrt{N}$, we can compensate for a factor of ten increase in $\sigma$ by just increasing $N$ (the number of experiments) by a factor of 100.

$M = 10$
$\sigma = 0.2$

$M = 1000$
$\sigma = 0.02$
The thing that matters is the TOTAL number of points that are thrown!

It does not matter if they are in 100, 10 or even 1 experiment – it is the number of points.

What if we treated each single randomly-thrown point as a single measurement?
Back to the Whale!

Treating Each Point as an Experiment:

What would happen if we set $M$ to 1? Then the number of points within the whale for each experiment ($m_i$) would always either be 0 or 1.

Each experiment's estimate of $A_{\text{whale}}$ is given by:

$$A_i = \frac{m_i}{M} A_{\text{box}}$$

So, our mean value for the area could be written:

$$\overline{A} = \frac{1}{N} \sum_{i=1}^{N} A_i = \frac{1}{N} \sum_{i=1}^{N} \frac{m_i}{M} A_{\text{box}} = \frac{A_{\text{box}}}{NM} \sum_{i=1}^{N} m_i$$

If $M=1$, then $m_i$ must be either 0 or 1, and:

$$\overline{A} = \frac{A_{\text{box}}}{N} \sum_{i=1}^{N} m_i = \frac{n}{N} A_{\text{box}}$$

where "$n$" is just the sum of all of the $m_i$ values (in other words, the total number of points within the whale).
Standard Deviation when M=1:

So, if we now define the mean value of A like this:

\[ \overline{A} = \frac{A_{\text{box}}}{N} \sum_{i=1}^{N} m_i \]

And (by analogy with what we’ve done before) the standard deviation of m like this:

\[ s_m^2 = \frac{1}{N-1} \left[ \sum_{i=1}^{N} m_i^2 - \frac{1}{N} \left( \sum_{i=1}^{N} m_i \right)^2 \right] \]

We arrive at the following expression for the standard error of the mean, in the case where we have only one experiment with many points:

\[ \sigma_{\overline{A}} \approx \frac{A_{\text{box}}}{\sqrt{N}} s_m \]
We’ll pick up from here next time.

See you Thursday at labs!