Numerical Analysis

- Mean/Average
- Variance
- Standard deviation
For a statistical population \( \{x_1, x_2, x_3, x_4, \ldots, x_n\} \)

- **Average:**
  \[
  \overline{X} \equiv \frac{1}{N} \sum_{i=1}^{N} x_i
  \]

- **Variance:**
  \[
  s_x^2 \equiv \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{X})^2
  \]

- **Standard Deviation:**
  \[
  s_x \equiv \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{X})^2}
  \]
Numerical Analysis

- For computation variance is often expressed as

\[ s^2_x \equiv \frac{1}{N-1} \left( \sum_{i=1}^{N} x_i^2 - N\overline{X}^2 \right) \]

OR

\[ s^2_x \equiv \frac{1}{N-1} \left[ \sum_{i=1}^{N} x_i^2 - \frac{\left( \sum_{i=1}^{N} x_i \right)^2}{N} \right] \]
/* Computes the average and the standard deviation of 100 data points. */
#include <stdio.h>
#include <math.h>
#define N 100
int main()
{
  float a[N]={0.974742, 0.0982212, 0.578671, 0.717988, 0.881543, 0.0771773, 0.910513, 0.576627, 0.506879,
  0.629856, 0.71646, 0.454598, 0.312042, 0.47837, 0.719982, 0.676287, 0.201261, 0.0298494, 0.378439,
  0.490109, 0.290748, 0.574289, 0.998625, 0.559518, 0.49549, 0.091864,
  0.792899, 0.0138333, 0.998678, 0.93009, 0.127889, 0.77911, 0.22417, 0.213076, 0.380052, 0.519128,
  0.547883, 0.011815, 0.350202, 0.14069, 0.948774, 0.721067, 0.896979, 0.26913, 0.97952, 0.146778,
  0.898354, 0.709611, 0.48403, 0.0549138, 0.105455, 0.695778, 0.485352, 0.0619048, 0.977566, 0.916668,
  0.261182, 0.848828, 0.597515, 0.39754, 0.713299, 0.837013, 0.247313, 0.25685, 0.764525, 0.115947,
  0.350333, 0.98772, 0.785004, 0.969169, 0.451979, 0.278109, 0.300974, 0.914255, 0.346524, 0.582331,
  0.815621, 0.85235, 0.368957, 0.665663, 0.554439, 0.00352195, 0.771442, 0.268123, 0.841114, 0.166509,
  0.52413};

  float sum=0, average, var=0, sd;
  int i;
  for (i=0;i<N;i++) sum=sum+a[i];
  average=sum/N;
  for (i=0;i<N;i++) var=var+pow( a[i]-average, 2);
  sd=sqrt(var)/(N-1);
  printf("Average= %f S.D.=%fn", average, sd);
  return 0;
}
Random Numbers:

A random number is a number generated by a process, whose outcome is unpredictable, and which cannot be subsequentially reliably reproduced.

Used to perform numerical simulations, that are hard to perform otherwise.
Random Numbers

- rand() returns an integer in the range [0, RAND_MAX] inclusive
- RAND_MAX is system dependent and is defined in stdlib.h
Here is a program to print random numbers 10 times:

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    for (i=0; i< 10; i++) printf("%d\n", rand());
    printf("\nMAX = %d\n", RAND_MAX);
    return 0;
}
```
If you want to generate a random number between 0 and 1

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i;
    for (i=0; i< 10; i++) printf("%fn", 1.0*rand()/RAND_MAX);
    printf("\nMAX = %d\n", RAND_MAX);
    return 0;
}
```

Note the "%f" format and the factor of 1.0.

The problem with these programs is that the same numbers are generated each time the programs are run.
For different set of random number use srand()

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    printf("Enter seed integer = ");
    scanf("%d", &i);
    srand(i);
    printf("%d\n", rand());
    return 0;
}
```

If no seed, then 1 is the default value.

**Problem:**
For the same seed number, the same random number is returned.
Numerical Analysis

- the following program generates a random number every time it is called.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    srand(time(NULL));
    printf("%d\n", rand());
    return 0;
}
```

time(NULL) returns the elapsed time since 00:00:00 GMT, January 1, 1970, measured in seconds
Random numbers

- `rand()` returns an integer between 0 and `RAND_MAX`.
- `1.0*rand()/RAND_MAX` returns a floating number between 0 and 1.
- `5.0*rand()/RAND_MAX` returns a floating number between 0 and 5.
- `10.0*rand()/RAND_MAX-5.0` returns a floating number between -5.0 and 5.0.
- `rand()%7` returns an integer of either 0, 1, 2, 3, 4, 5, or 6.
- `rand()%7+10` returns an integer of either 10, 11, 12, 13, 14, 15, or 16.
To approximate \( \ln 2 \)

From Taylor’s series expansion of \( 1/(1+x) \)

\[
\frac{1}{1+x} = 1 - x + x^2 - x^3 + x^4 - x^5 \ldots
\]

Integrating both the sides yields

\[
\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} \ldots
\]

**X=1 gives:** \( \ln(2) = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \ldots \)
Using Random Numbers to Compute Mathematical Constants

---Monte Carlo Simulation

To approximate \( \ln 2 \)

The area surrounded by the curve \( y = \frac{1}{x} \), \( x = 1 \) and \( x = 2 \) is

\[
\int_{1}^{2} \frac{1}{x} \, dx = \ln x \bigg|_{1}^{2} = \ln 2 - \ln 1 = \ln 2
\]
So if you generate a random number, \( x \), whose range is between 1 and 2, and another random number, \( y \), whose range is between 0 and 1, the probability that \( xy < 1 \) is the shaded area (\( \ln 2 \)).

This can be implemented in the following C program.
```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    float x, y;
    int i,
    count=0;
    int n;
    printf("Enter iteration number = ");
    scanf("%d", &n);
    srand(time(NULL));
    for (i=0; i< n; i++)
    {
        x=1.0*rand()/RAND_MAX+1.0;
        y=1.0*rand()/RAND_MAX;
        if (x*y < 1.0) count=count+1;
    }
    printf("True value = \n", log(2));
    printf("Appx value = %\n", 1.0*count/n);
    return 0;
}
```