INTRODUCTION TO MONTE CARLO METHODS

Roger Martz

Introduction

The Monte Carlo method solves problems through the use of random sampling. Otherwise know as the random walk method.

A statistical method.

Introduction

Solve mathematical problems:

- Multi-dimensional integrals
- Integral equations
- Differential equations
- Matrix inversion
- Extremum of functions

Introduction

Solve problems which involve intrinsically random or stochastic processes.

- Particle transport through matter
- Operations research

Most useful in solving problems involving many contingencies or multi-dimensional space of high order.

Introduction

Consider particle transport through matter

Three possible methods

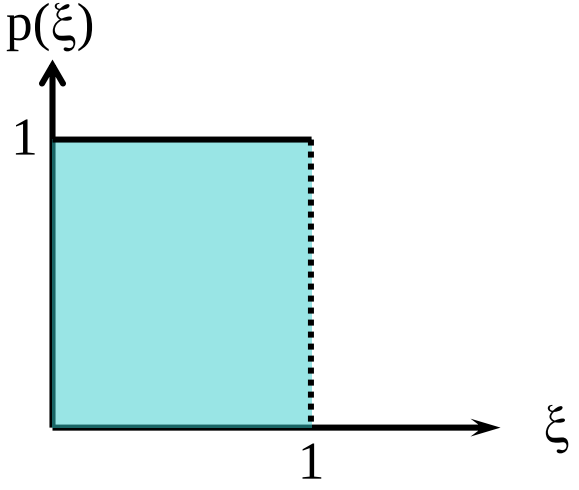
- Direct Experiment
  - Sometime virtually impossible
  - Very expensive / time consuming
- Theoretical
  - Solve complex equations subject to interface & boundary conditions
  - Often need approximations
- Monte Carlo
  - Use basic physics laws and probabilities
  - Minimal amount of approximations
  - Since a statistical method, can require large amounts of computer time

Random Numbers

It is possible to generate a sequence of random numbers, ξ, in which there is no apparent relationship between members of the sequence.

$$0 \le \xi \le 1$$

Such a sequence is said to be uniformly distributed if, in the limit of a very large sequence, the fraction of numbers that fall in a range $\Delta\xi$ about ξ is very closely equal to $\Delta\xi$ and is independent of ξ.



Sample Mean and Variance

- Given a sample of size n of a random variable x

$$x_1, x_2, \ldots, x_n$$

- The sample mean value of x is defined as

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

- The sample variance of the x_i about the mean is defined as

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{1}{n-1}\{\sum_{i=1}^{n} x_i^2 - \frac{1}{n}[\sum_{i=1}^{n} x_i]^2\}$$

Standard Deviaton

The sample standard deviation is the square root of the sample variance.

$$S = \sqrt{S^2}$$

Finding PI

Equation for a Circle:
(centered at the origin)

$$x^2 + y^2 = r^2$$

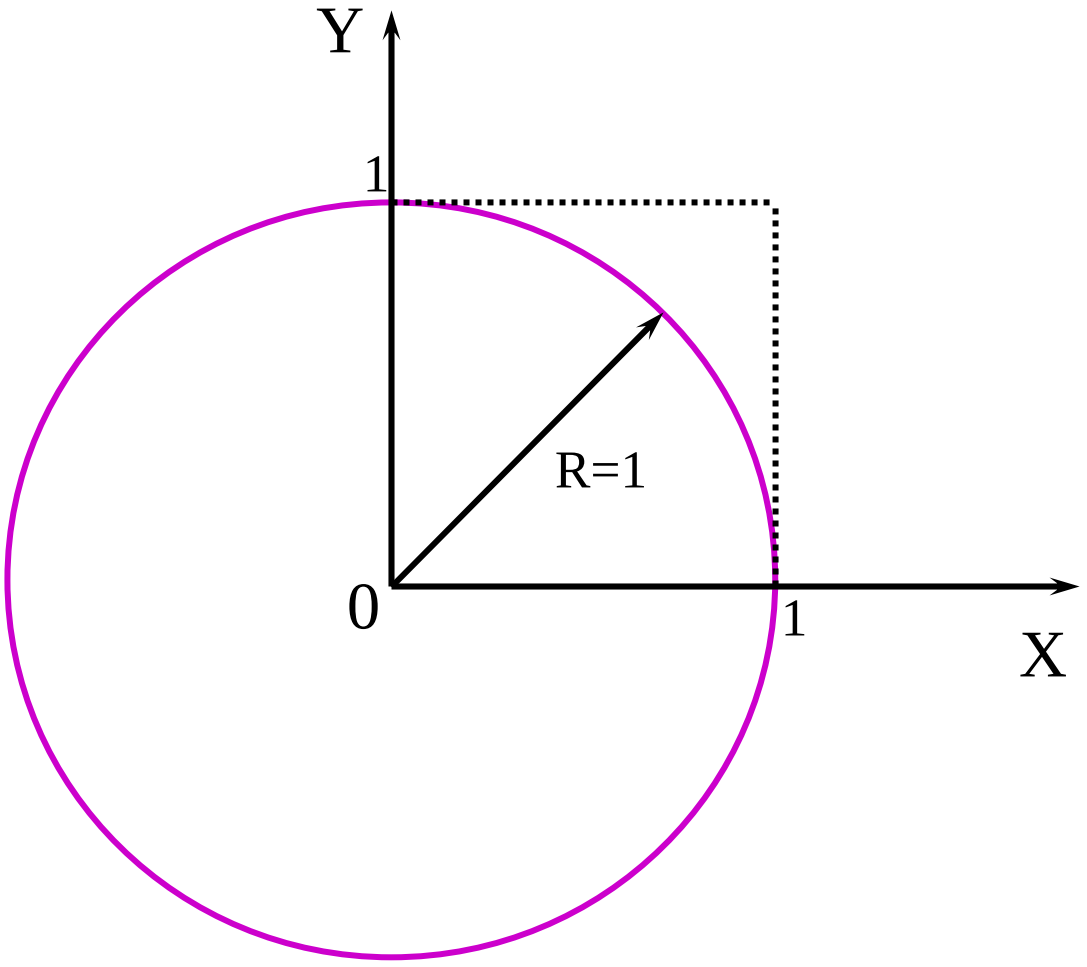
Area of a Circle:

$$A = \pi \cdot r^2$$

PI Equation:

$$\pi = \frac{A}{r^2}$$

Finding PI



Area of circle in 1st quadrant: $\pi = \frac{A}{4 \cdot r^2}$

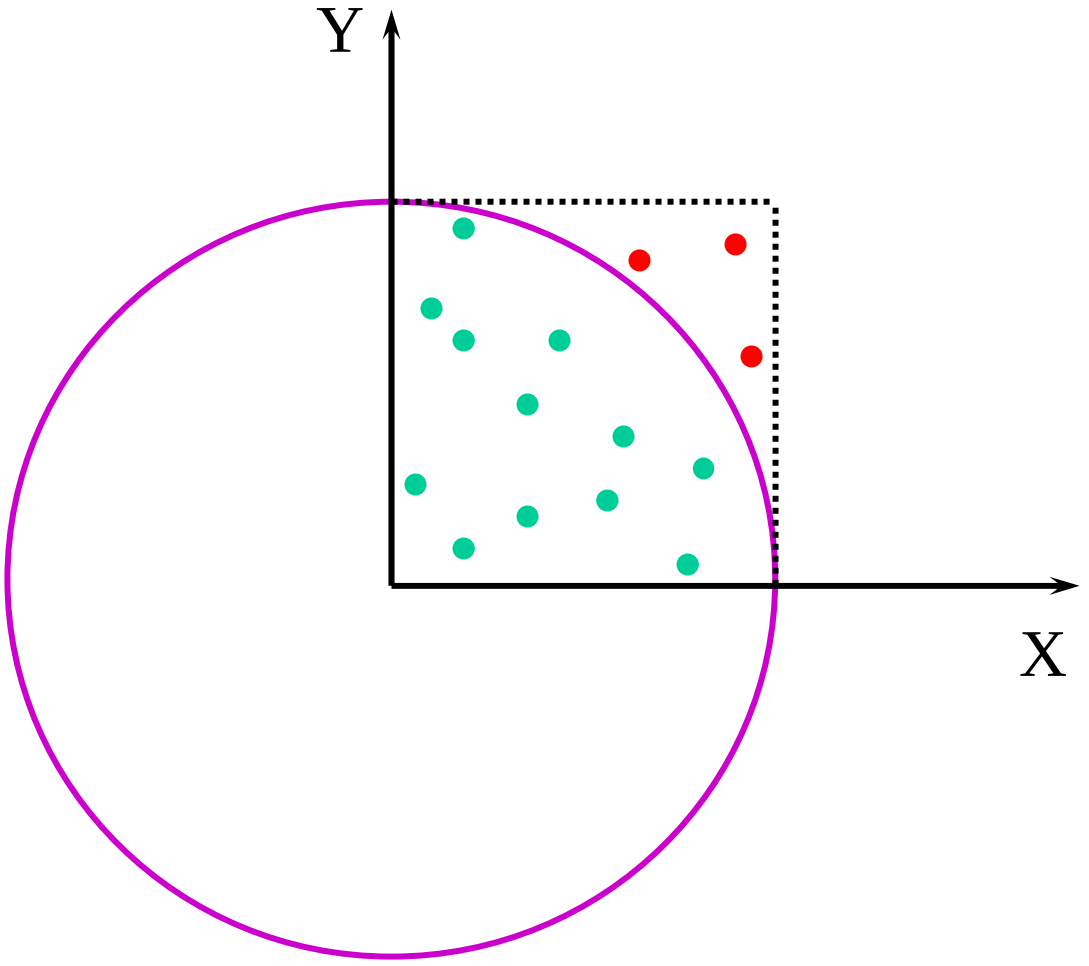
Finding PI

- We can determine PI by finding the area, A, for a given radius, r.
- This is equivalent to integrating the equation of the circle as follows:

$$A = \int_a^b f(x)dx = \int_0^r \sqrt{r^2 + x^2}\,dx$$

Finding PI

Rejection Method



Finding Pi

Algorithm Guts:

- Initialize a counter for the area.
- Select a “x”.
- Select a “y”.
- Calculate $x^2 + y^2$.
- If this value is less than r^2, increment area counter.
- Repeat sampling

Finding PI

Batch – Means Method

- Obtain a sequence of independent B samples (batches) each with N trials (histories) per batch.
- Perform means calculations (determine the mean and standard deviation) on the batches.
- Total histories: H = B * N

Finding PI

Batch – Means Method

… Outer Loop over batches

… Initialize appropriate batch variables

… Inner Loop over batch trials/histories

… Inner loop calculations

… End inner loop

… End outer loop

… Finalize calculations and normalize results

Exercise #1: PI

- Make a copy of file pi_template.c
- Complete the program
  - Add the “guts”
  - Add the code to find the average, variance, & standard deviation
- Run the following calculations:
  - 100 batches, 1000 histories per batch
  - 100 batches, 4000 histories per batch
  - 100 batches, 16000 histories per batch
- Do your results converge to the known value of PI?
- How does the error change as the histories increase?

File: pi_template.c

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

// function prototypes

double myRand(void);

//  main program start

int main(int argc, char *argv[]) {
//
  int             i, j;
  int             ir;
  int             nbatch = 10;
  int             batch_size = 100;
  int             nseed = 1;
  unsigned long int   batch_success;
  double x, y, r;
  double area_batch;
  double area_accum, area_accum2, area;
  double var, std;
//
  printf("\n ***** Program to Calulate PI by Monte Carlo Methods *****\n\n");
//
  if (argc < 4) {
    printf(" ***** ERROR: Not enough command line arguments! \n");
    printf("\n      Please enter in the following order:\n");
    printf("      Random Number Seed\n");
    printf("      Number of Batches\n");
    printf("      Histories per Batch\n");
    exit(0);
  }
//
  printf("Program Name is    : %s\n", argv[0]);
  printf("Random Number seed : %s\n", argv[1]);
  printf("Number of Batches  : %s\n", argv[2]);
  printf("Histories per Batch: %s\n", argv[3]);
//
//  Re-assign command line input
//
  nseed      = atoi(argv[1]);
  nbatch     = atoi(argv[2]);
  batch_size = atoi(argv[3]);
```

File: pi_template.c

```
//
//  Initialize the random number generator
//
  srand(nseed);
//
//  Initialize the variance accumulators for x and x-squared
//
  area_accum  = 0.0;
  area_accum2 = 0.0;
//
  for (j=0; j<nbatch; j++)  {
    batch_success = 0;
    for (i=0; i<batch_size; i++)  {
      x = ****
      y = ****
      r = ****
      if ( **** )  {
        ****
      }
    }
    area_batch = ((double) batch_success) / ((double) batch_size);
//
//  Increment the variance accumulators
//
    area_accum  = ****
    area_accum2 = ****
  }
//
//  Normalize results
//
  area = ****
//
//  Calculate variance and standard deviation
//
  var  = ****
  std  = ****
//
//  Print results
//
  printf("\n *** RESULTS ***\n");
  printf("Area: %f +/- %f (%f)\n", area, std, std/area);
  printf("Pi  : %f +/- %f (%f)\n", area*4.0, std*4.0, std/area);
//
}                           // end main function
```

File: pi_template.c

```
//
double myRand()  {
//
//  Return a random number between 0 and 1
//
  double randx;
  randx = ((double) rand()) / (RAND_MAX + 1.0);
  return randx;
}
```


Optional Exercise 1A

Use the Monte Carlo method to evaluate the following integral:

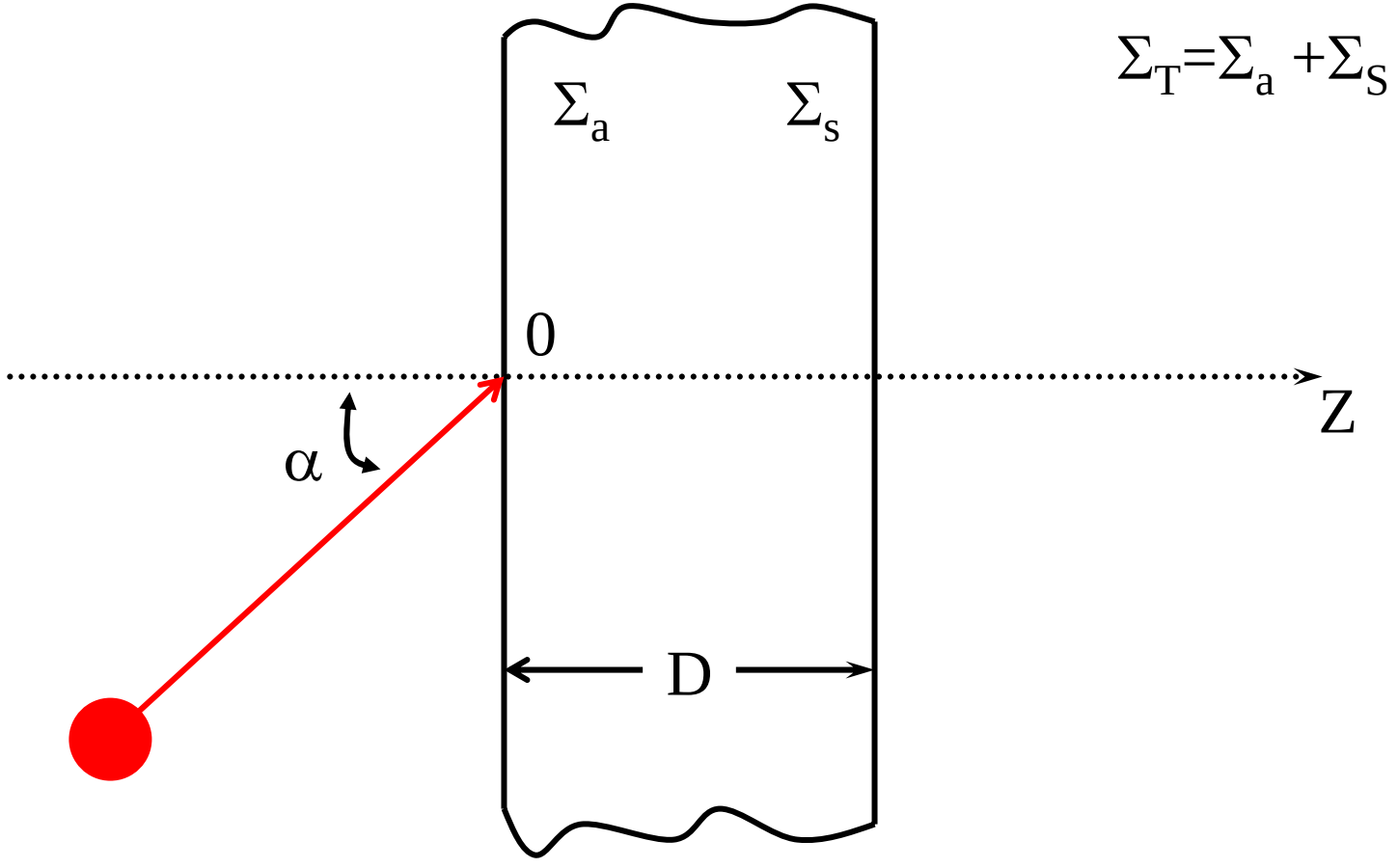
$$\int_{1}^{4} \sqrt{x}\,dx$$

That is, find the area under the following curve from x = 1 to x = 4

$$y = \sqrt{x}$$

Neutral Particle Transport

The problem picture:



Neutral Particle Transport

The problem in words:

- Neutral particles (neutrons, photons) are incident at angle α with respect to the normal on an infinite homogeneous slab of thickness D.
- The slab is characterized by scattering and absorption probabilities, Σ_s and Σ_a.
- Scattering is isotropic in the laboratory reference system and they scatter without loss or gain of energy.
- The total interaction (reaction) probability is
  $\Sigma_T = \Sigma_s + \Sigma_a$

Neutral Particle Transport

Problem Objective:

- Solve for the numerical values of the probabilities of transmission through the slab, reflection from the slab, and absorption in the slab, as well as estimates of the reliability of these probabilities.

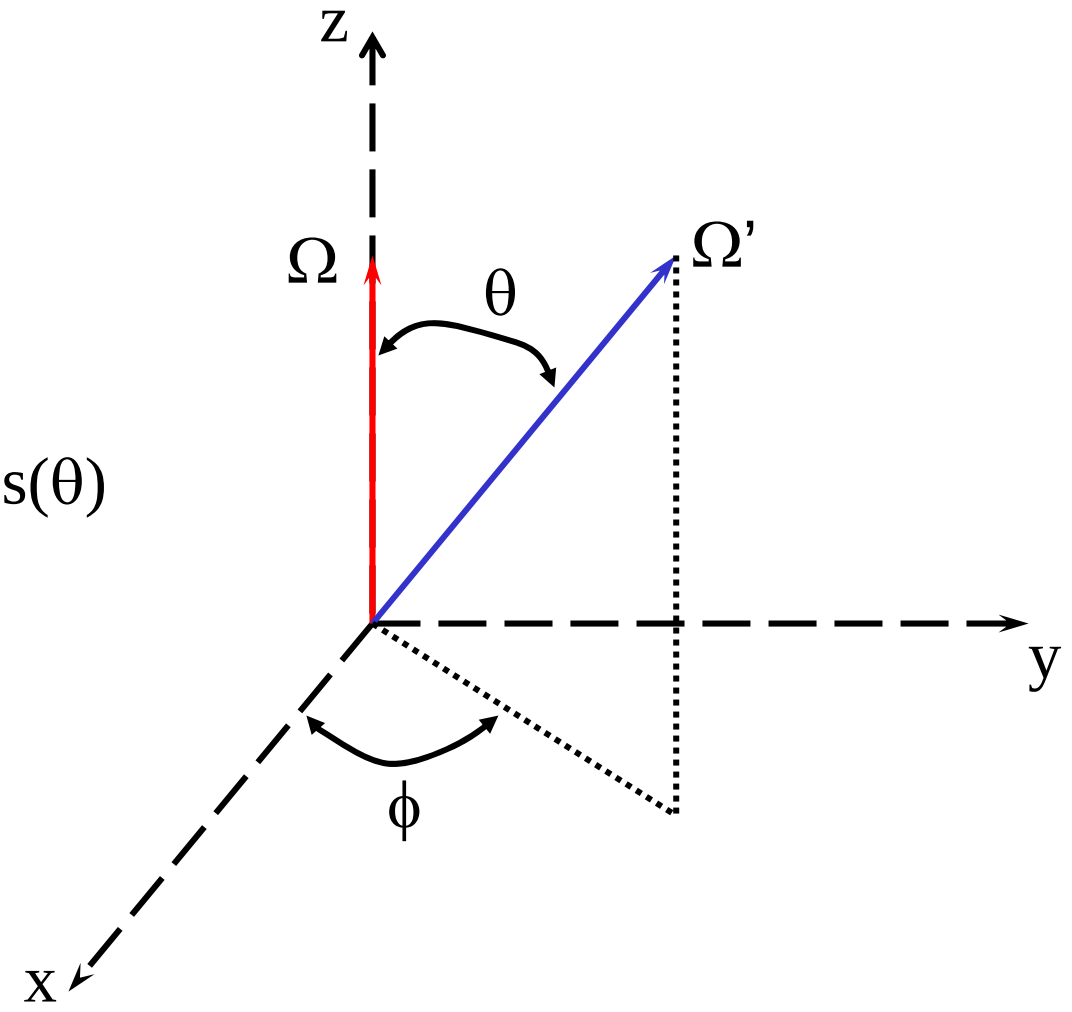
- Probabilities are real numbers between 0.0 and 1.0, inclusive.

  Therefore, $T + A + R = 1.0$

Neutral Particle Transport

Defining the scatter angles:

Define: $\mu=\cos(\theta)$



Monte Carlo Golden Rule

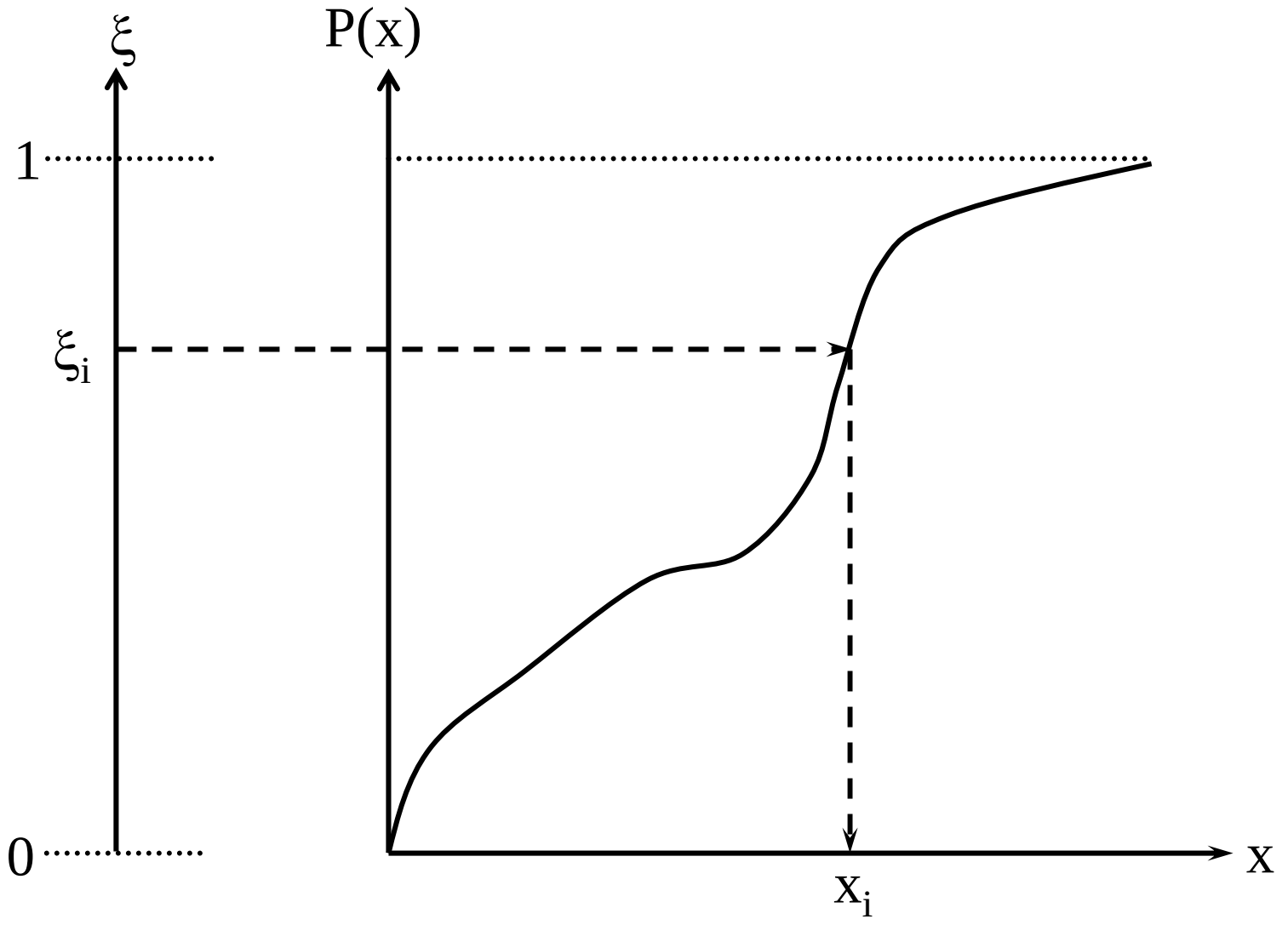
Suppose we have a random variable x defined in the range $a \le x \le b$ with a know probability function p(x). We wish to obtain a random series of "x" values such that in the limit that we select a large number of them, we have the distribution p(x).

We know how to generate a series of random numbers ξ uniformly distributed in the range $0 \le \xi \le 1$

To transform between ξ space and x space we equate the probability of obtaining x in the range dx about x with the probability of obtaining a ξ in the range $d\xi$ about ξ and integrating from the lower limit to some arbitrary point.

$$\int_a^x p(x')dx' = \int_0^\xi d\xi'$$

Monte Carlo Golden Rule



Applying the Golden Rule

Distance to next collision

$$P(r) = \int_0^r e^{-\Sigma r'} \Sigma dr' = 1 - e^{-\Sigma r}$$

$$1 - e^{-\Sigma r} = \xi$$

$$r = -(\ln \xi) / \Sigma$$

For distance to ANY collision, use Σ_T for Σ.

Applying the Golden Rule

Polar Angle for Isotropic Scattering

$$P(\mu) = \int_{-1}^{\mu} \Sigma_s(\mu')2\pi d\mu' / \Sigma_s = \int_{-1}^{\mu} d\mu' = (\mu + 1)/2$$

where $\Sigma_s(\mu)/\Sigma_s = 1/4\pi$

$$(\mu + 1)/2 = \xi$$

$$\boxed{\mu = 2\xi - 1}$$

Applying the Golden Rule

Azimuthal Scattering Angle

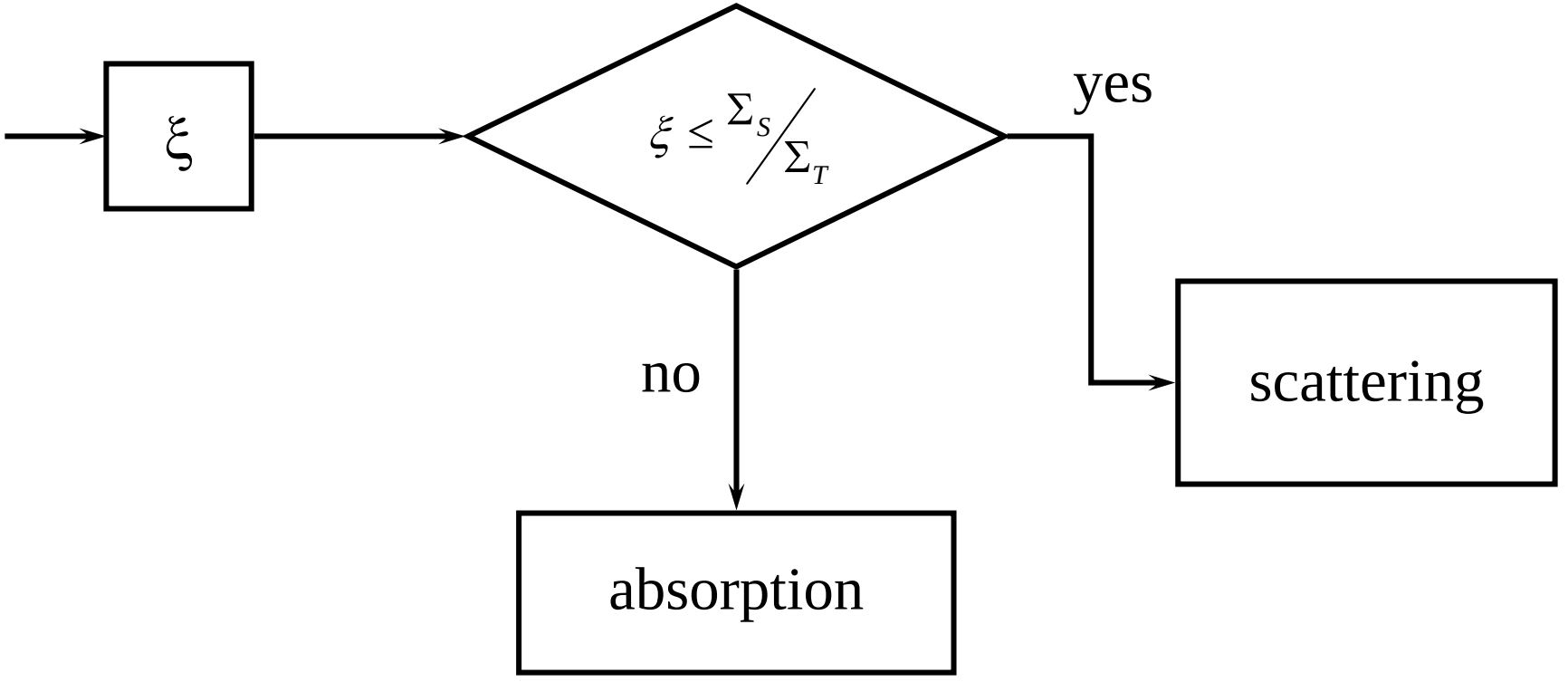
$$P(\phi) = \int_0^{\phi} \frac{1}{2\pi} \cdot d\phi' = \frac{\phi}{2\pi}$$

$$\frac{\phi}{2\pi} = \xi$$

$$\boxed{\phi = 2\pi\xi}$$

Applying the Golden Rule

Interaction (Reaction) Type



Analog Slab Problem

… Outer Loop over batches
  … Initialize Inner Loop counters
  … Inner Loop over histories per batch
    … Initialize particle starting parameters
    … While Loop for single particle
      … distance to next collision
      … does particle transmit? If yes, break While.
      … does particle reflect? If yes, break While.
      … does particle absorb? If yes, break While.
      … particle scatters
    … end While Loop
  … end Inner Loop
  … accumulate results for statistics
… end Outer Loop

Exercise #2: Analog Slab

- Make a copy of file slab1_template.c
- Complete the program
  - Add the “guts”
  - Add the code to find the average, variance, & standard deviation

- Run the following calculations each with 100 batches and 10000 histories per batch:
  - Case #1: Σ_s=0.2, Σ_a=0.8, D=10, α=0
  - Case #2: Σ_s=0.2, Σ_a=0.8, D=10, α=30
  - Case #3: Σ_s=0.2, Σ_a=0.8, D=10, α=60
  - Case #4: Σ_s=0.7, Σ_a=0.3, D=10, α=0
  - Case #5: Σ_s=0.7, Σ_a=0.3, D=10, α=30
  - Case #6: Σ_s=0.7, Σ_a=0.3, D=10, α=60

  Tabulate results on next slide.

- What trends do you observe?

Exercise #2: Analog Slab

Case #	Transmitted	Std. Dev.	Rel. Error	FOM
2-1				
2-2				
2-3				
2-4				
2-5				
2-6				

Exercise #2: Analog Slab

Case #	Reflected	Std. Dev.	Rel. Error	FOM
2-1				
2-2				
2-3				
2-4				
2-5				
2-6				

Exercise #2: Analog Slab

Case #	Absorbed	Std. Dev.	Rel. Error	FOM
2-1				
2-2				
2-3				
2-4				
2-5				
2-6				

File: slab1_template.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <sys/times.h>
#include <sys/resource.h>

// function prototypes

double myRand(void);
double newAngle(double oldAngle, double phi, double cos_theta);
void timers(double *cpu, double *et);

//  main program start

int main(int argc, char *argv[]) {
//
  int   i, j;
  int nbatch, batch_size, nseed;
//
//  A -- Absorption
//  R -- Reflection
//  T -- Transmission
//
//  est -- estimate for
//  std -- standard deviation for
//  var -- variance for
//  fom -- Figure Of Merit
//
//  Counters
//
  unsigned long int   Ac, Rc, Tc;
  double   Acount, Rcount, Tcount ;
//
  double estA, stdA, varA, fomA;
  double estR, stdR, varR, fomR;
  double estT, stdT, varT, fomT;
//
//  Accumulator Variables for Statistics Calculations
//
  double Ay, Ry, Ty;
  double Ax, Ax2, Rx, Rx2, Tx, Tx2;
```

File: slab1_template.c

```
//  Input Parameters
//
  double sigmaT, sigmaA, sigmaS, thicknessD, alpha0;
//
//  Calculated Distances
//
  double r1, z;
//
//  Calculated Angles and Cosines
//
  double phi, cos_alpha1, cos_alpha0, cos_theta;
//
//  Other Variables
//
  double test1;
  double myPI;
  double weight;
//
//  Timing parameters
//
  double dtime;
  double cpu0, cpu1, et0, et1;
//
//  Initialize pi.
//
  myPI = acos(-1.0);
//
  printf("\n ***** Slab Dynamics by Monte Carlo Methods *****\n");
  printf(" ***** Analog Calculation                    *****\n\n");
//
  if (argc < 8) {
    printf(" ***** ERROR: Not enough command line arguments! \n");
    printf("\n      Please enter in the following order:\n");
    printf("      Random Number Seed\n");
    printf("      Number of Batches\n");
    printf("      Histories per Batch\n");
    printf("      Slab Thickness\n");
    printf("      Slab Absorption Cross Section\n");
    printf("      Slab Scattering Cross Section\n");
    printf("      Particle Starting Angle (>=0 & < 90)\n");
    exit(0);
  }
```

File: slab1_template.c

```
  printf("Program Name is              : %s\n", argv[0]);
  printf("Random Number seed           : %s\n", argv[1]);
  printf("Number of Batches            : %s\n", argv[2]);
  printf("Histories per Batch          : %s\n", argv[3]);
  printf("Slab Thickness               : %s\n", argv[4]);
  printf("Slab Absorption Cross Section: %s\n", argv[5]);
  printf("Slab Scattering Cross Section: %s\n", argv[6]);
  printf("Particle Starting Angle      : %s\n", argv[7]);
//
//  Re-assign command line input
//
  nseed      = atoi(argv[1]);
  nbatch     = atoi(argv[2]);
  batch_size = atoi(argv[3]);
  thicknessD = atof(argv[4]);
  sigmaA     = atof(argv[5]);
  sigmaS     = atof(argv[6]);
  alpha0     = atof(argv[7]);
  cos_alpha0 = cos(alpha0 * myPI /180.);
  sigmaT     = sigmaS + sigmaA;
  test1      = sigmaS / sigmaT;
//
  if (cos_alpha0 > 0.9999) cos_alpha0 = 0.9999;
//
//  Initialize the random number generator & accumulator variables.
//
  srand(nseed);
//
  Ax  = 0.0;   Ax2 = 0.0;
  Rx  = 0.0;   Rx2 = 0.0;
  Tx  = 0.0;   Tx2 = 0.0;
  Ac = 0;      Tc = 0;     Rc = 0;
//
  (void) timers (&cpu0, &et0);
```

File: slab1_template.c

```
 for (j=0; j<nbatch; j++)  {
    Acount = 0;  Rcount = 0;  Tcount = 0;
//
    for (i=0; i<batch_size; i++)  {
//
//  Start a particles with position and direction
//
      weight = 1.0;  z = 0.0;  cos_alpha1 = cos_alpha0;
//
      while (1)  {
//
//  Distance to next collision & location
//
     r1 =  ****            z =  ****
//
//  Particle transmits
//
     if ( **** )  {
       ****
       break;
     }
//
//  Particle reflects
//
     if ( **** )  {
       ****
       break;
     }
//
//  Particle absorbs
//
     if ( **** )  {
       ****
       break;
     }
//
//  Particle scatters
//
     cos_theta = ****
     if (cos_theta > 0.9999)  cos_theta = 0.9999;
     phi = ****
     cos_alpha1 = newAngle( **** );
     if (cos_alpha1 > 0.9999)  cos_alpha1 = 0.9999;
//
      }
    }                    // end i-loop for histories/batches
```


File: slab1_template.c

```
//
//  Accumulate results for statistics.
//
    Ay   = ((double) Acount) / ((double) batch_size);
    Ax  ****
    Ax2 ****
//
    Ry   = ((double) Rcount) / ((double) batch_size);
    Rx  ****
    Rx2 ****
//
    Ty   = ((double) Tcount) / ((double) batch_size);
    Tx  ****
    Tx2 ****
//
  }           // end j-loop for batches
//
  (void) timers (&cpu1, &et1);
//
//  Calculate cpu time for the main loop process.
//
  dtime = cpu1 - cpu0;
//
//  Normalize results and calculate errors
//
  estR = ****
  varR = ****
  stdR = ****
  fomR = estR / (varR * dtime * 3600.);
//
  estA = ****
  varA = ****
  stdA = ****
  fomA = estA / (varA * dtime * 3600.);
//
  estT = ****
  varT = ****
  stdT = ****
  fomT = estT / (varT * dtime * 3600.);
//
```


File: slab1_template.c

```
//
double myRand()  {
//
//  Return a random number between 0 and 1
//
  double randx;
  randx = ((double) rand()) / (RAND_MAX + 1.0);
  return randx;
}
//
double newAngle(double oldAngle, double phi, double cos_theta)  {
//
//  Return the new scattering angle (as a cosine)
//
  double sinlt;
  double sinth;
  double newCosth;
//
  sinlt = sqrt(1.0 - cos_theta * cos_theta);
  sinth = sqrt(1.0 - oldAngle * oldAngle);
  newCosth = oldAngle * cos_theta + sinth * sinlt * cos(phi);
  return newCosth;
}
//
void timers(double *cpu, double *et)  {
//
//  Retrun cpu and elasped time through pointers.
//
  struct rusage r;
  struct timeval t;
//
  getrusage( RUSAGE_SELF, &r );
  *cpu = r.ru_utime.tv_sec + r.ru_utime.tv_usec*1.0e-6;
//
  gettimeofday( &t, (struct timezone *)0 );
  *et = t.tv_sec + t.tv_usec*1.0e-6;
}
```

Analog vs. Biased

- ANALOG MONTE CARLO
  Sample events according to their natural physical probabilities.

- BIASED MONTE CARLO
  Do not directly simulate nature's laws.
  Bias the distribution function in a fair way to produce a more efficient random walk.

Survival Biasing

- Nature says that the probability of a particle absorbing in a collision is the ratio Σ_a/Σ_T.
- Suppose the objective of the Monte Carlo calculation is to determine the number of particles transmitted through a thick shield.
- The number transmitted would increase if there was no absorption.
- So we cheat. Do NOT allow particles to absorb. Distort nature's laws.

Survival Biasing

- Since we distorted nature's laws, we must make a correction in order to preserve the Monte Carlo "fair game".

- Use the concept of statistical weight
  - When a history begins, $w_0 = 1$
  - Force all collisions to be scattering
  - The particle's statistical weight must be reduced by the probability that it survives: Σ_s/Σ_T
  - So, on each collision: $w_{new} = w_{old} * \Sigma_s/\Sigma_T$

- Other Implementation Notes
  - $T = T + w_{old} * \Sigma_s/\Sigma_T$
  - $R = R + w_{old} * \Sigma_s/\Sigma_T$
  - $A = A + w_{old} * \Sigma_a/\Sigma_T$

Russian Roulette

- If the material has a high scatter fraction / probability, it is possible for a particle's weight to be multiplied by the Σ_s/Σ_T ratio many times. This can result in extremely low weights.
- It may not always be efficient use of the computer time to follow all of these low weight particles.
- When a particle's weight falls below a certain level, play the Russian roulette game.
  - Choose a random number and compare it against weight/cutoff
  - If ξ < weight/cutoff, set the weight equal to cutoff
  - Else, terminate the particle.

Figure Of Merit

The more efficient a Monte Carlo calculation is, the larger the FOM will be because less computer time is required to reach a given value of R.

$$FOM \equiv \frac{1}{R^2 T}$$

$$T \propto N \qquad R^2 \propto 1/N$$

Exercise #3: Biased Slab

- Make a copy of your previous file. You want both an analog and biased file to work with.
- Change your program to add the appropriate weight formulas.
  - See the previous slides.
  - Keep the integer counters so you can print out the number of particles transmitted, absorbed, and reflected in both the analog and biased programs.
  - Look at slab2_template.c for guidance.

- Run the following calculations each with 25 batches and 5000 histories per batch:
  - Use: Σ_s=0.2, Σ_a=0.8, D=8, α=0
  - Make 3 runs with the analog code with different RN seeds.
  - Make 3 runs with the biased code with the same 3 RN seed values. Use weight cutoff value of 0.1

Tabulate results on next slides.

File: slab2_template.c

```
 for (j=0; j<nbatch; j++)  {
    Acount = 0.;   Rcount = 0.;   Tcount = 0.;
//
    for (i=0; i<batch_size; i++)  {
//
//  Start a particle with position, direction, and intial weight.
//
      weight = 1.0;  z = 0.0;   cos_alpha1 = cos_alpha0;
//
      while (1)  {
//
//  Distance to next collision
//
    r1 =  ****
    z = ****
//
//  Particle transmits
//
    if ( **** )  {
      ****
      break;
    }
//
//  Particle reflects
//
    if ( **** )  {
      ****
      break;
    }
//
//  Particle always scatters; adjust the weight; tabulate absorption
//
    ****
```


File: slab2_template.c

```
//
//  Play the weight cutoff game.
//
      if (weight < cutoff)  {
        if ( **** ))  {
//
//  Particle survives; re-set its weight.
//
           ****
        }
        else {
//
//  Particle dies; terminate it.
//
          break;
        }
      }             // end if for particle below weight cutoff

//
//  Particle scatter mechanics
//
    cos_theta = ****
    if (cos_theta > 0.9999)  cos_theta = 0.9999;
    phi = ****
    cos_alpha1 = newAngle(**, **, **);
    if (cos_alpha1 > 0.9999)  cos_alpha1 = 0.9999;
//
      }
    }               // end i-loop for histories/batches
```


Exercise #3: Analog / Biased Comparison

Case # / Seed #	Transmitted	Std. Dev.	Rel. Error	FOM	Counts
3A-1					
3A-2					
3A-3					
3B-1					
3B-2					
3B-3					

Exercise #3: Analog / Biased Comparison

Case # / Seed #	Reflected	Std. Dev.	Rel. Error	FOM	Counts
3A-1					
3A-2					
3A-3					
3B-1					
3B-2					
3B-3					

Exercise #3: Analog / Biased Comparison

Case # / Seed #	Absorbed	Std. Dev.	Rel. Error	FOM	Counts
3A-1					
3A-2					
3A-3					
3B-1					
3B-2					
3B-3					

Exercise #4: Cutoff Values

No programming for this one. Objective: see how cutoff parameter values affect the variance reduction performance.

- Run the following calculations each with 25 batches and 50000 histories per batch:
  - Use: Σ_s=0.2, Σ_a=0.8, D=8, α=0
  - Make runs with the baised code with the same RN seed.
  - Use weight cutoff value of 0.1, 0.01, 0.001, 0.0001

Tabulate results on next slides.

Exercise #4: Cutoff Value Comparison

Case # / Seed #	Transmitted	Rel. Error	FOM	Counts	Time
4B-1 0.1					
4B-2 0.01					
4B-3 0.001					
4B-4 0.0001					

Concluding Remarks

What did you learn?

Concluding Remarks

- Introduced to the Monte Carlo Method
- Used the MC method to integrate functions
- Fundamentals of neutral particle transport & how to simulate
- Difference between analog & biased calculations (sometimes cheating is a good thing)
- Monte Carlo Golden Rule