# Monte Carlo Method: Simulation

John Burkardt (ARC/ICAM)
Virginia Tech

.....

Math/CS 4414:
"The Monte Carlo Method: SIMULATION"
http://people.sc.fsu.edu/~jburkardt/presentations/
monte_carlo_simulation.pdf

.....

**ARC**: Advanced Research Computing
**ICAM**: Interdisciplinary Center for Applied Mathematics

09-11-13 November 2009

VirginiaTech

# Monte Carlo Method: Simulation

- A Traffic Simulation
- Game Theory: Duels
- Gambler's Ruin
- Brownian Motion
- Random Walks
- Coding a Three-Way Duel
- A Model of Epidemics

VirginiaTech

# Overview

This is the third set of talks on the Monte Carlo Method (MCM).

This talk considers the Monte Carlo Method (MCM) as a way of *simulating* complex processes.

VirginiaTech

# Monte Carlo Method: Simulation

- **A Traffic Simulation**
- Game Theory: Duels
- Gambler's Ruin
- Brownian Motion
- Random Walks
- Coding a Three-Way Duel
- A Model of Epidemics

VirginiaTech

# Traffic Simulation

# Traffic Simulation

Computer simulation is intended to provide a simple model of a complex process. The model is made up of simplifications, arbitrary choices, and rough measurements.

We hope that there is still enough relationship to the real world that we can study the model and get an idea of how the original process works, predict outputs for given inputs, and how we can improve it by changing some of the parameters.

A particularly hard thing for mathematical analysis is to **prove** that a given model will always behave in a certain way (for example, the output won't go to infinity.)

Sampling tries to answer these questions by **practical examples** - we try lots of inputs and see what happens.

VirginiaTech

# Traffic Simulation - How to Set a Traffic Light

We begin with a model of a traffic light. The traffic light has certain settings (length of the red and green cycles), and the intersection has some other properties that can be estimated (typical number of cars arriving, number of cars that can get through the light).

We will suppose the traffic light comes with default settings, and that we've measured a typical traffic density, and want to see whether installing the traffic light will cause problems.

We are ignoring the cross-traffic and the oncoming traffic. We are only going to look at how the line of cars waiting to go is affected by the way the light operates.

VirginiaTech

# Traffic Simulation - Assumptions about Time

Our first simplification is to assume that we can measure time in units **CYCLE_LENGTH** of 10 seconds. Nothing interesting will be allowed to happen faster than this. We will use **CYCLE** to count the number of cycles.

We will assume the red light lasts for **RED_CYCLES** cycles, and the green light for **GREEN_CYCLES** cycles. And we'll ignore the yellow light (like most drivers).

We'll need variables called **RED_TIMER** and **GREEN_TIMER** that will keep track of how long the light has been red or green, so we know when to switch.

VirginiaTech

# Traffic Simulation - Assumptions about Traffic

The variable **CARS** will count the number of cars waiting at the intersection.

We assume the intersection starts out empty.

We assume that at every second, there's a probability **P** that a new car will arrive at the light.

We'll assume that, if the light is green, 8 cars can leave each 10 seconds.

VirginiaTech

# Traffic Simulation - Are the Light Settings Appropriate?

Our simulation will involve letting the traffic light switch back and forth between red and green, applying the rules which determine the cars which come in and the cars which leave, and monitoring the buildup of waiting cars.

If the traffic light settings are well chosen, then over the long run we should see a short average wait per car.

Bad settings might mean a typical car must wait at the intersection while the light changes to green several times (this is the "DC" option).

If the settings are really bad, then the number of waiting cars will grow as time goes on, so that many cars will not get through at all (the "Beltway" option).

VirginiaTech

# Traffic Simulation - Program Outline

```
for cycle = 1 : cycle_num
  r = rand ( cycle_length, 1 );
  cars_new = sum ( r < p );
  cars = cars + cars_new;
  cars_in = cars_in + cars_new;
  if ( light == 'g' )
    [ cars, cars_out, light, green_timer ] = go ( ...
      green_cycles, cars, cars_out, light, green_timer );
  else
    [ cars, light, red_timer ] = stop ( red_cycles, ...
      cars, light, red_timer );
  end
  car_wait_cycles = car_wait_cycles + cars;
end
```

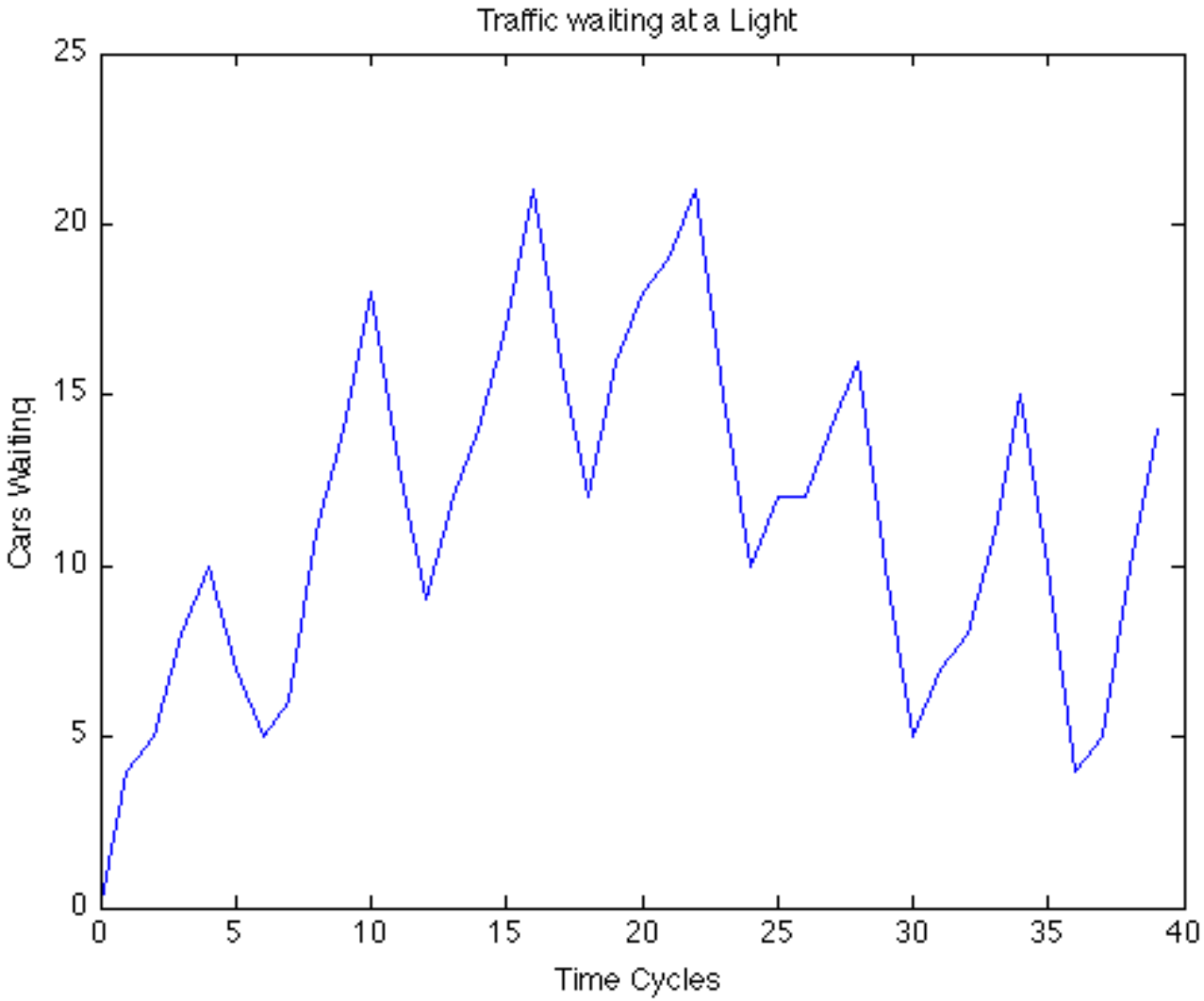
VirginiaTech

# Traffic Simulation - A 40-cycle Simulation

```
Number of cycles =         40
Simulated time =           400 seconds
Number of cars in =        112
Number of cars waiting =   16
Number of cars out =       96
Percentage Out/In =      85.7%
Average wait =     41.96 seconds
Average wait =      0.70 light cycles
```

VirginiaTech

# Traffic Simulation - A 40-cycle Simulation



# Traffic Simulation - A 40-cycle Simulation

Actually, I had to run the program several times before finding a plot in which the number of cars waiting decreases. In most of the runs, you can see from the plot that the light is red too long.

The other indicator of trouble is the "throughput", that is, the ratio between the number of cars that came to the light versus the ones that got through.

As time increases, this number should go to 100of cars is going to grow indefinitely.

Let's get a better idea of what's happening with these settings by taking 8 times as many cycles!

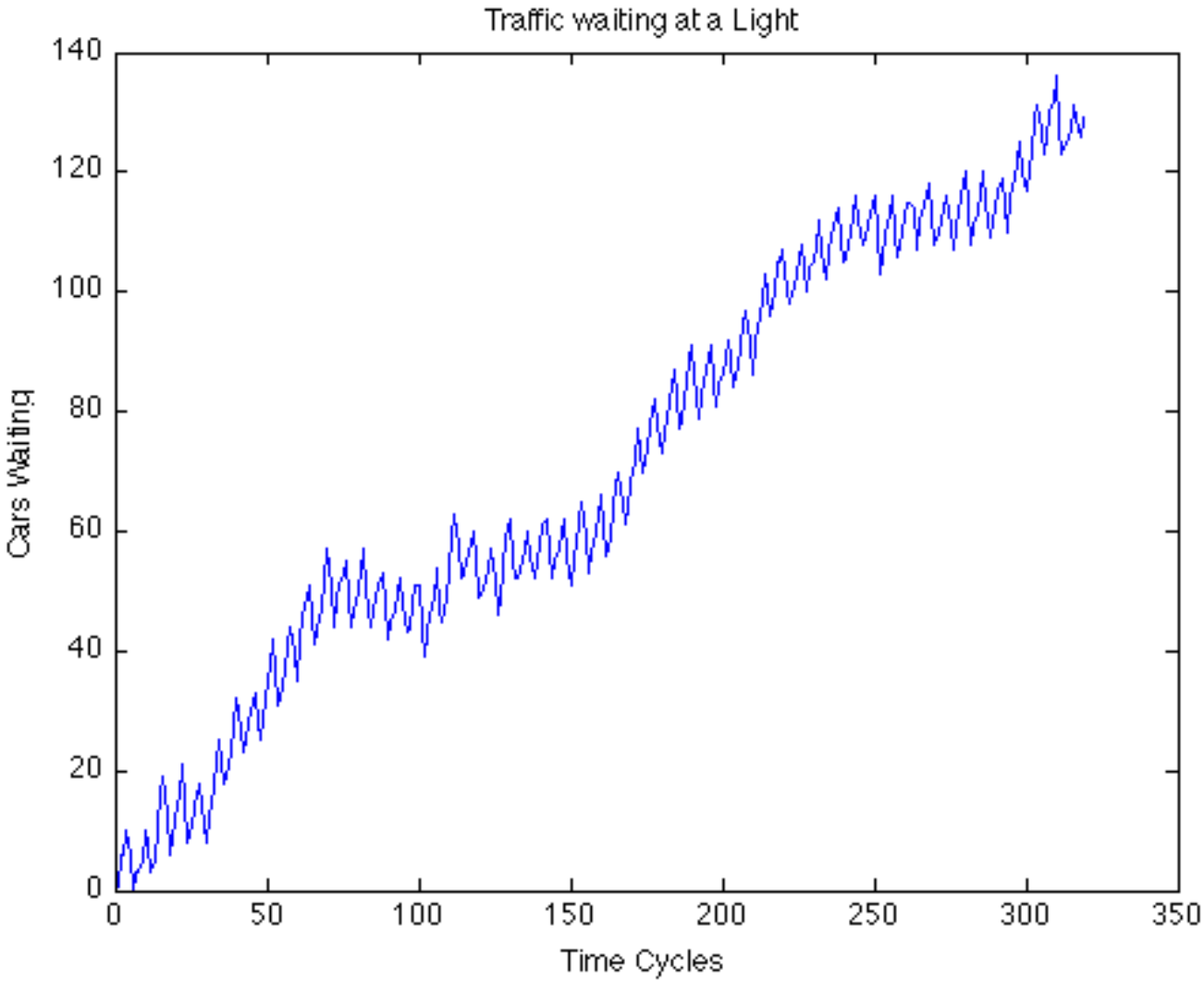
VirginiaTech

# Traffic Simulation - A 320-cycle Simulation

```
Number of cycles =         320
Simulated time =           3200 seconds
Number of cars in =        979
Number of cars waiting =   131
Number of cars out =       848
Percentage Out/In =      86.6%
Average wait =    233.15 seconds
Average wait =      3.89 light cycles
```

VirginiaTech

# Traffic Simulation - A 320-cycle Simulation



# Traffic Simulation - Conclusion

So if our data about the traffic density is reasonably accurate, then the simulation data provided by our model suggests that we must change the settings on the traffic light or else the highway will become a parking lot.

Simulation allows us to make a reasonable adjustment to our traffic lights before we've even set them up.

There is almost no other practical method of trying to estimate the performance of a complex system, with internal settings, and random inputs!

VirginiaTech

# Traffic Simulation

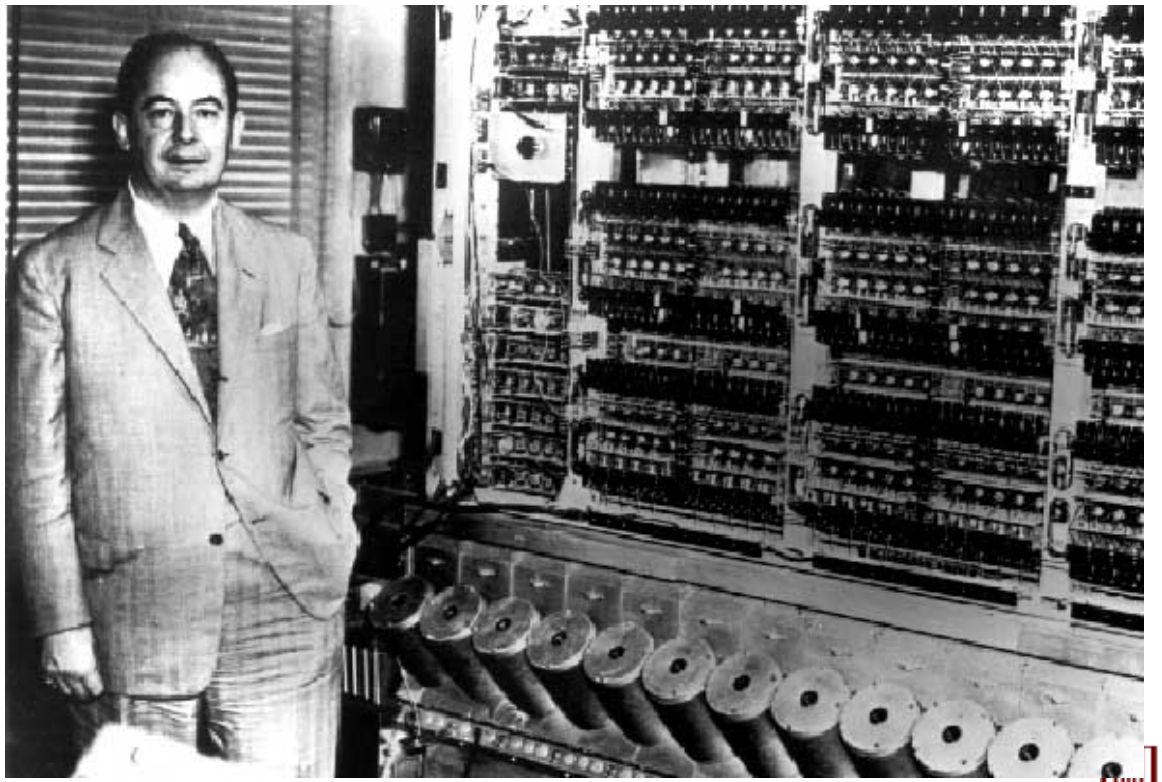
# Monte Carlo Method: Simulation

- A Traffic Simulation
- **Game Theory: Duels**
- Gambler's Ruin
- Brownian Motion
- Random Walks
- Coding a Three-Way Duel
- A Model of Epidemics

VirginiaTech

# Game Theory

In the 1950's, John von Neumann (who wrote the first computer user manual, for the ENIAC), developed **game theory**, which sought strategies in multi-player games. The Monte Carlo method was used to simulate the games, and determine the best strategies.



VirginiaTech

# Game Theory - A Duel

A basic example was a **duel**. Two players take turns firing at each other until one is hit. The accuracy, or probability of hitting the opponent on one shot, is known for both players.

Here there is no strategy (just keep firing), but we want to know the probabilities that either player survives.

If we assume Player A has an accuracy of $\frac{4}{6}$ and Player B an accuracy of $\frac{5}{6}$ then Player A can survive if:

1. A hits on the first shot: (4/6) OR
2. A misses, B misses, A hits (2/6)*(1/6)*(4/6) OR
3. A misses, B misses, A misses, B misses, A hits: (2/6)*(1/6)*(2/6)*(1/6)*(4/6) OR...

VirginiaTech

# Game Theory - A Duel

The probability that A survives is thus an infinite sum:

$$\sum_{i=0}^{\infty}((1-\frac{4}{6})(1-\frac{5}{6}))^i\,\frac{4}{6}$$

This has the form $4/6 * (1 + x + x^2 + x^3 + ...)$ where $0 < x < 1$ so this is equal to $4/6 * 1/(1-x)$ so the probability that A wins is

$$\frac{PA}{PA + PB - PA * PB} = \frac{12}{17}$$

while B's chance is

$$\frac{PB * (1 - PA)}{PA + PB - PA * PB} = \frac{5}{17}$$

# Game Theory - Simulate a Duel

```
turn_num = 0;
while ( 1 )
  turn_num = turn_num + 1;  %  Player 1 fires.
  r = rand ( );
  if ( r <= p(1) )
    survivor = 1;
    break
  end
  turn_num = turn_num + 1;  %  Player 2 fires.
  r = rand ( );
  if ( r <= p(2) )
    survivor = 2;
    break
  end
end
```

VirginiaTech

# Game Theory - Simulating A Duel

To *estimate* the probabilities of A and B surviving (which in this case we can already work out exactly), we would simulate the duel 100 or 1,000 times, counting the frequency with which each player survived, and dividing by the number of duels.

```
s = zeros(2,1);
turn_average = 0;

for duel = 1 : duel_num
  [ survivor, turn_num ] = duel_once ( p );
  s(survivor) = s(survivor) + 1;
  turn_average = turn_average + turn_num;
end

s = s / duel_num;
turn_average = turn_average / duel_num;
```

VirginiaTech

# Game Theory - A Three Person Duel

The Three-Way Duel from "The Good, The Bad, and The Ugly"

VirginiaTech

# Game Theory - A Three Person Duel

I said that in a duel, there's no strategy (except of course to shoot at your opponent on your turn).

But suppose we move to a three person duel involving A, B, and C? And let's suppose that C is a relatively poor shot.

If A and B are "reasonable", they will probably shoot at each other, since each represents the biggest threat to the other. As soon as one gets hit, it will be C's turn, and that's his best chance, to have the first shot at the remaining opponent.

But a disaster occurs if, instead of A or B knocking the other out, poor shot C accidentally hits A or B on his turn. Because then C does not get the first shot; rather the survivor gets first shot at C, and we know whichever survivor it is is a better shot!

VirginiaTech

# Game Theory - A Three Person Duel

Now the coding for this problem is similar to that for the duel, except that, on each shot, the player has a **choice** of who to shoot at. We said A and B should shoot at each other.

It seems that C should shoot at A, so that if he accidentally hits him, he has to face the weaker player B.

Another possibility: C could shoot at no one (if that's allowed). Then, once A or B is out, C is sure to get the first shot at the survivor.

VirginiaTech

# Game Theory - ASSIGNMENT

In a three person duel, players A, B and C have accuracies (probabilities of hitting on a single shot) of $\frac{5}{6}$, $\frac{4}{6}$ and $\frac{2}{6}$.

1) Program a simulation of this duel. Have each player fire at the most accurate (surviving) opponent. A goes first, then B, C, A, ... until only one is left. Estimate the probabilities of A, B and C surviving by simulating 1000 duels.

2) Change C's strategy so that if A and B are both alive, C doesn't shoot at all, (but when one opponent is left, C does shoot.) Estimate the probabilities of A, B and C surviving.

**TURN IN**: The probabilities for case 1 and for case 2 (six numbers). Due on Friday, 13 November.

VirginiaTech

# Monte Carlo Method: Simulation

- A Traffic Simulation
- Game Theory: Duels
- **Gambler's Ruin**
- Brownian Motion
- Random Walks
- Coding a Three-Way Duel
- A Model of Epidemics

VirginiaTech

# Gambler's Ruin

Two gamblers, **A** and **B**, decide to play a game.

**A** has \$3 and **B** has \$7.

They decide to flip a coin. If it comes up heads, **A** wins \$1 from **B**, while tails works the other way.

The game is over when one gambler is bankrupt.

VirginiaTech

# Gambler's Ruin

Here are some questions we can ask:

1. What is the probability that **A** will win?
2. What is the expected value of the game to **A**, that is, how much will he win or lose, on average?
3. How long will a typical game last?
4. What are the chances that the winner will be ahead the entire game?
5. What happens if we change the amount of money that **A** and **B** have?

VirginiaTech

# Gambler's Ruin

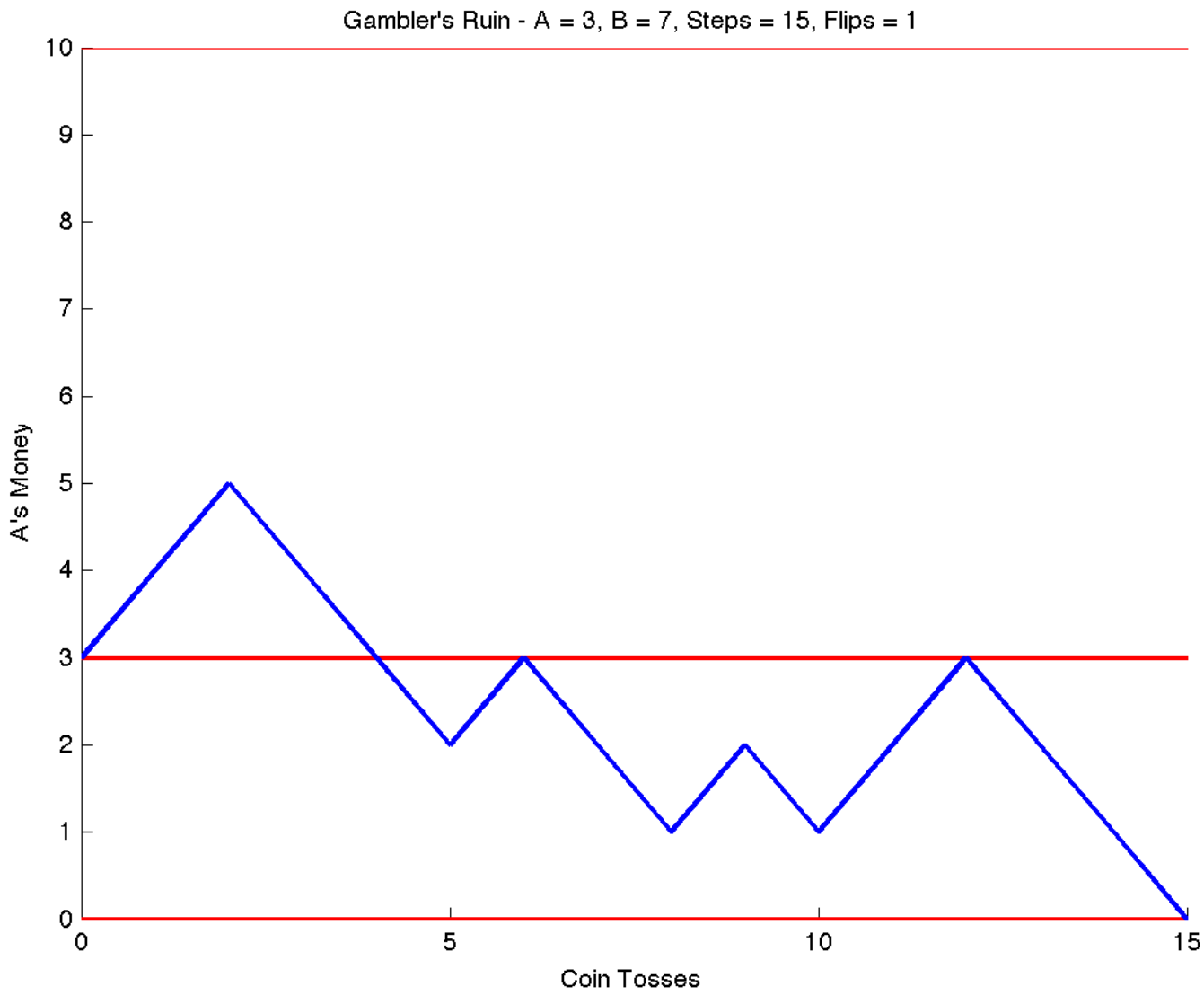
Here's a typical game, in which **A** starts with \$3 and **B** with \$7:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | H | H | T | T | T | H | T | T | H | T | H | H | T | T | T |
| A | 3 | 4 | 5 | 4 | 3 | 2 | 3 | 2 | 1 | 2 | 1 | 2 | 3 | 2 | 1 | 0 |
| B | 7 | 6 | 5 | 6 | 7 | 8 | 7 | 8 | 9 | 8 | 9 | 8 | 7 | 8 | 9 | 10 |

**A** loses after 15 tosses, having tossed 6 heads and 9 tails.

VirginiaTech

# Gambler's Ruin



# Gambler's Ruin

In the duel, we could see that there was a big advantage to going first.

In Gambler's Ruin, the player who starts with more money has an advantage, but how much?

It's easy to see that if **A** starts with \$3 dollars, the game can end in 3 steps. Can we expect that in such a case, a typical game might last 6, or 9 or 12 steps? Does it depend in any way on how much more money **B** has?

Suppose that **A** and **B** start with the same amount of money. In a typical game, is it likely that they will each be in the lead about the same number of tosses?

VirginiaTech

# Gambler's Ruin

A single plot can show you an example of how the game works, but it can't tell you all the things that can happen.

For a simulation, we need to look at a lot of games, and we try different starting bets (the "parameters" of this model).

Let's consider the following cases:

| $A | $B | Remarks |
|---|---|---|
| $3 | $7 | Our starting example. |
| $30 | $70 | 10 times as long? |
| $10 | $10 | Equal stakes, equal odds |
| $1 | $100 | Game will be quick. |

VirginiaTech

# Gambler's Ruin - Program (Initialize)

```
for game = 1 : game_num

  step_num = 0;
  leader = '0';
  flip_num = -1;
  a = a_stakes;
  b = b_stakes;
```

VirginiaTech

# Gambler's Ruin - Program (Play One Game)

```
while ( 0 < a & 0 < b )
  step_num = step_num + 1;
  if ( rand ( ) <= 0.5 )
    a = a + 1;
    b = b - 1;
  else
    a = a - 1;
    b = b + 1;
  end
  if ( a_stakes < a & leader ~= 'A' )
    leader = 'A'; flip_num = flip_num + 1;
  elseif ( a < a_stakes & leader ~= 'B' )
    leader = 'B'; flip_num = flip_num + 1;
  end
end
```

VirginiaTech

# Gambler's Ruin - Program (Record Results)

```
  if ( b == 0 )
    a_wins = a_wins + 1;
  else
    b_wins = b_wins + 1;
  end
  flip(game) = flip_num;
  step(game) = step_num;
end
```

VirginiaTech

# Gambler's Ruin - Results of 1000 Games

| $A | $B | Length | Max | A Prob | Flips | Max Flips |
|---|---|---|---|---|---|---|
| $3 | $7 | 21 | 135 | 0.29 | 1.7 | 14 |
| $30 | $70 | 2,010 | 12,992 | 0.28 | 19.7 | 189 |
| $10 | $10 | 101 | 722 | 0.49 | 4.6 | 59 |
| $1 | $100 | 104 | 10,472 | 0.01 | 0.5 | 1 |

VirginiaTech

# Gambler's Ruin - Some Facts

From this data you might be able to guess the following:

- the expected number of steps is $A * $B;
- The most likely number of flips or changes in the lead is 0! (We'll see this data in a minute);
- **A**'s chance of winning is $A / ( $A + $B ) ;
- The expected value for **A** is $0.

VirginiaTech

# Gambler's Ruin - The Expected Value

What is the expected value of this game?

**A**'s chance of winning \$B is $\frac{\$A}{\$A+\$B}$
the chance of losing \$A is 1 - this probability, or $\frac{\$B}{\$A+\$B}$.

The expected value is **value(win)*p(win) + value(loss)*p(loss)**
or

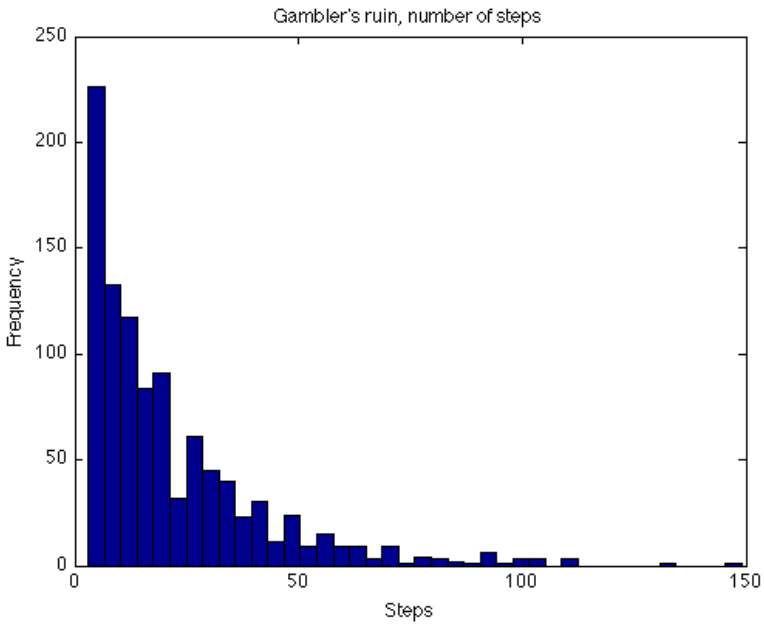
$$\$B\frac{\$A}{\$A+\$B} - \$A\frac{\$B}{\$A+\$B} = 0$$

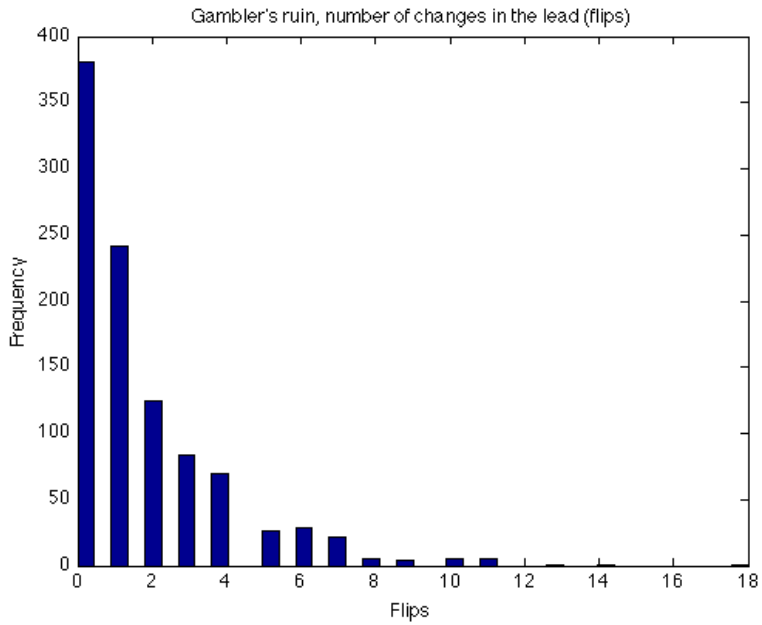
so even when **A** has \$1 against **B**'s \$100, it's a fair game.

(small chance of big win) - ( big chance of small loss).

VirginiaTech

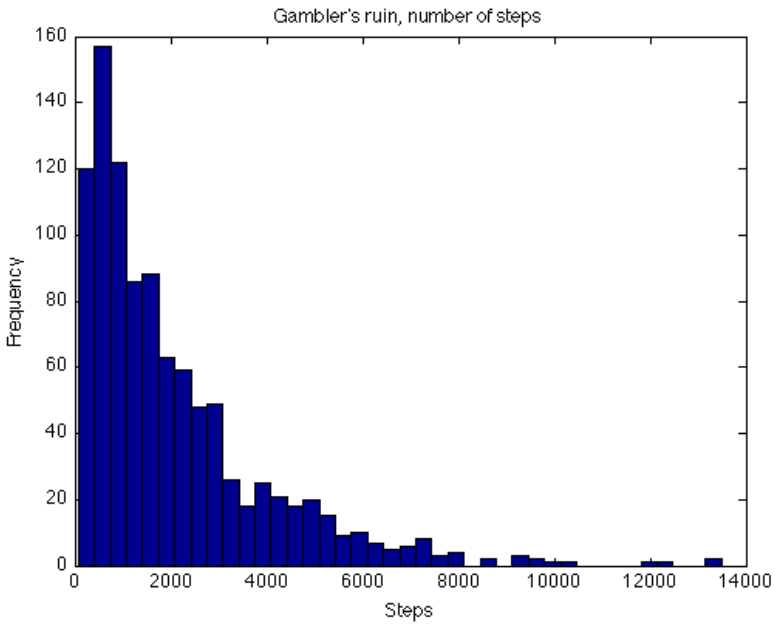
# Gambler's Ruin - (3,7)

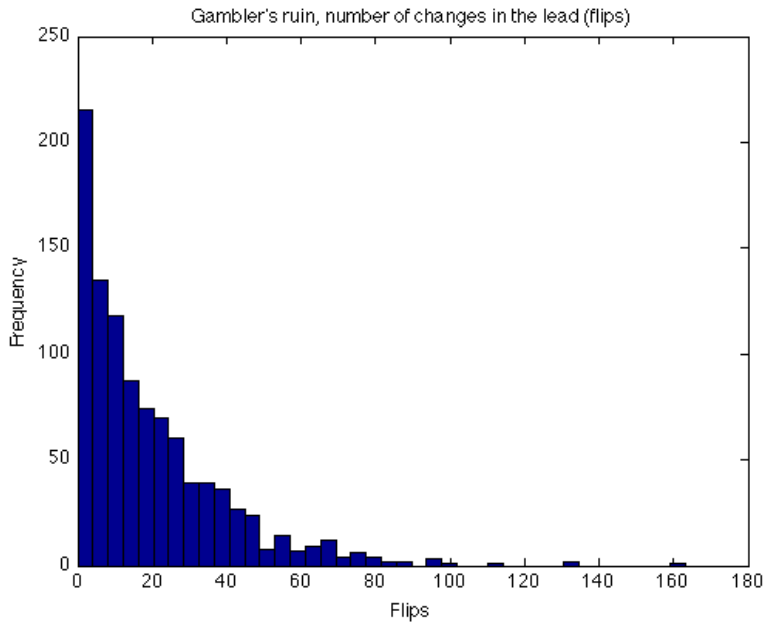



VirginiaTech

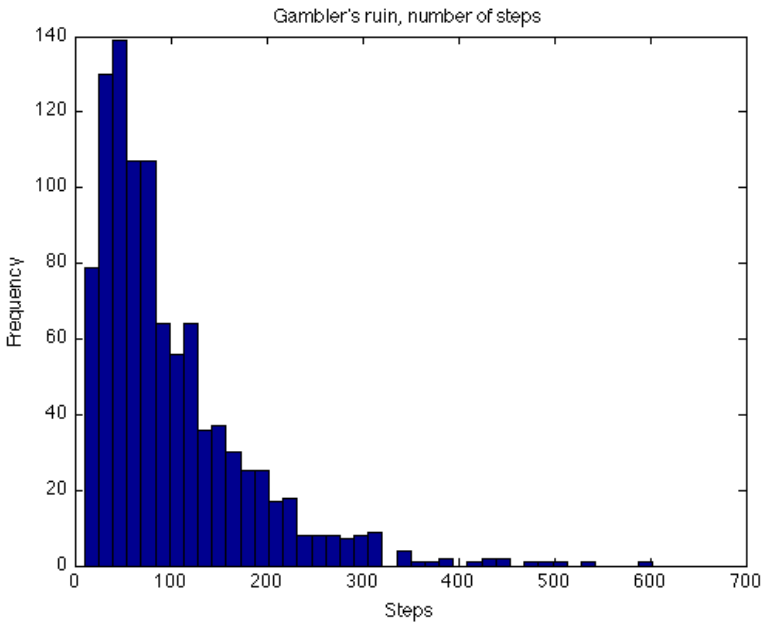
# Gambler's Ruin - (30,70)

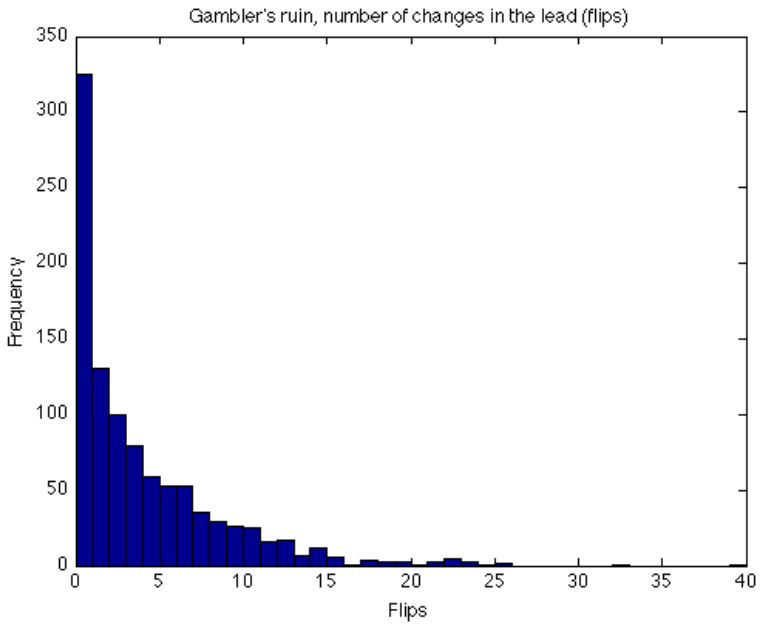



VirginiaTech

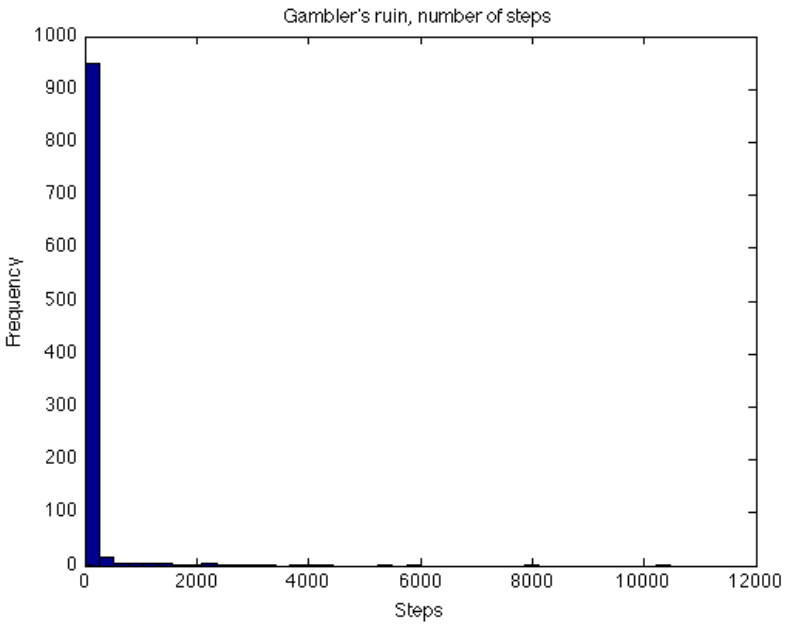
# Gambler's Ruin - (10,10)

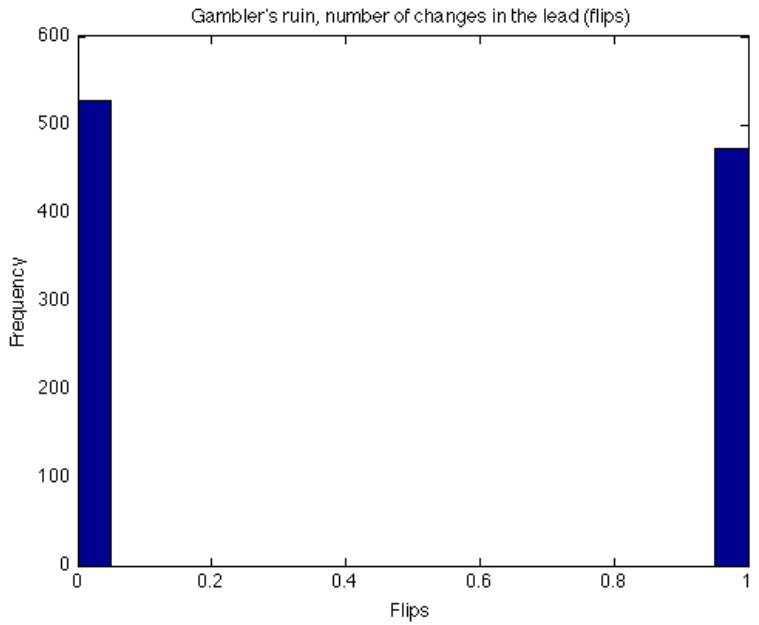



VirginiaTech

# Gambler's Ruin - (1,100)




VirginiaTech

# Gambler's Ruin - Conclusion

This simulation is useful because, although the system is very simple, its behavior is in some ways counter-intuitive.

Moreover, some of the unusual features only show up if we look at "enough" cases.

In part, this is an example of a system that is NOT well modeled by the normal distribution, where the average behavior is dominant.

Instead, we have many games of short duration, but a few of such long duration that they bring the average far up.

The fact that a player with just \$1, playing against a player with \$1000, can expect, on average, to play for 1000 tosses, is almost impossible to believe.

VirginiaTech

# Monte Carlo Method: Simulation

- A Traffic Simulation
- Game Theory: Duels
- Gambler's Ruin
- **Brownian Motion**
- Random Walks
- Coding a Three-Way Duel
- A Model of Epidemics

VirginiaTech

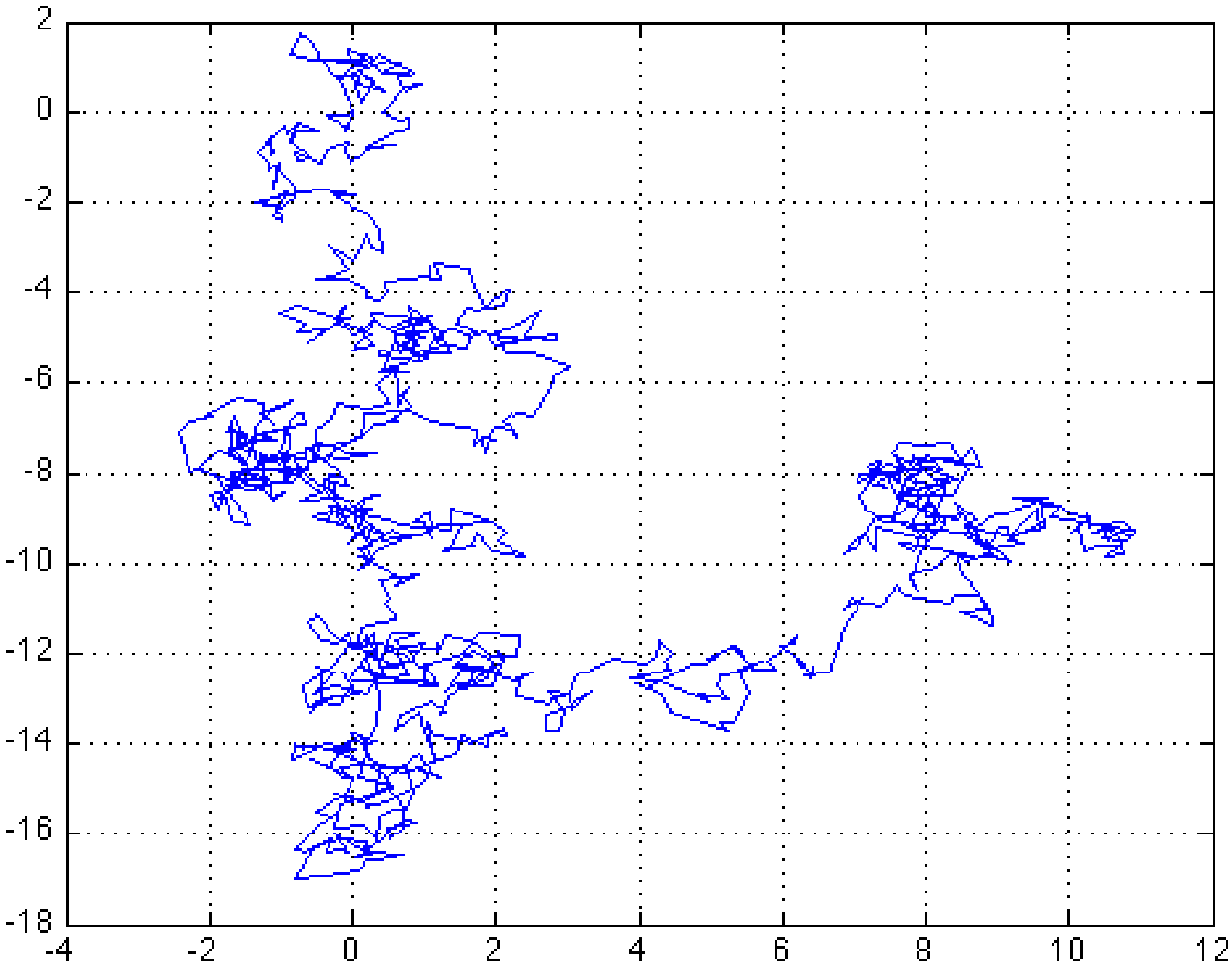
# Brownian Motion

In 1827, Scottish botanist Robert Brown was studying pollen grains which he had mixed in water. Although the water was absolutely still, the pollen grains seemed to quiver and move about randomly. He could not stop the motion, or explain it. He carefully described his observations in a paper.

When other researchers were able to reproduce the same behavior, even using other liquids and other particles, the phenomenon was named **Brownian Motion**, although no one had a good explanation for what they were observing.

VirginiaTech

# Brownian Motion - Simulated Brownian Motion




aTech

# Brownian Motion - Einstein Explains

In 1905, the same year that he published his paper on special relativity, Albert Einstein wrote a paper explaining Brownian motion. Each pollen grain, he said, was constantly being jostled by the motions of the water molecules on all sides of it. Random imbalances in these forces would cause the pollen grains to twitch and shift.

Moreover, if we observed the particle to be at position **(X,Y)** at time **T**=0, then its distance from that point at a later time **T** was a normal random variable with mean 0 and variance **T**. In other words, its typical distance would grown as $\sqrt{T}$.

VirginiaTech

# Brownian Motion - A Simulation

```
T = 10.0;
N = 1000;
h = sqrt ( T / N );
x(1) = 0.0;
y(1) = 0.0;

for i = 1 : N
  x(i+1) = x(i) + h * randn ( );
  y(i+1) = y(i) + h * randn ( );
end
```

Can you suggest another way to write this program which gets rid of the loop?

VirginiaTech

# Brownian Motion - A Simulation

```
T = 10.0;
N = 1000;
h = sqrt ( T / N );
x(1) = 0.0;
y(1) = 0.0;

x(2:N+1) = h * cumsum ( randn(1:N,1) );
y(2:N+1) = h * cumsum ( randn(1:N,1) );
```

VirginiaTech

# Brownian Motion - Diffusion

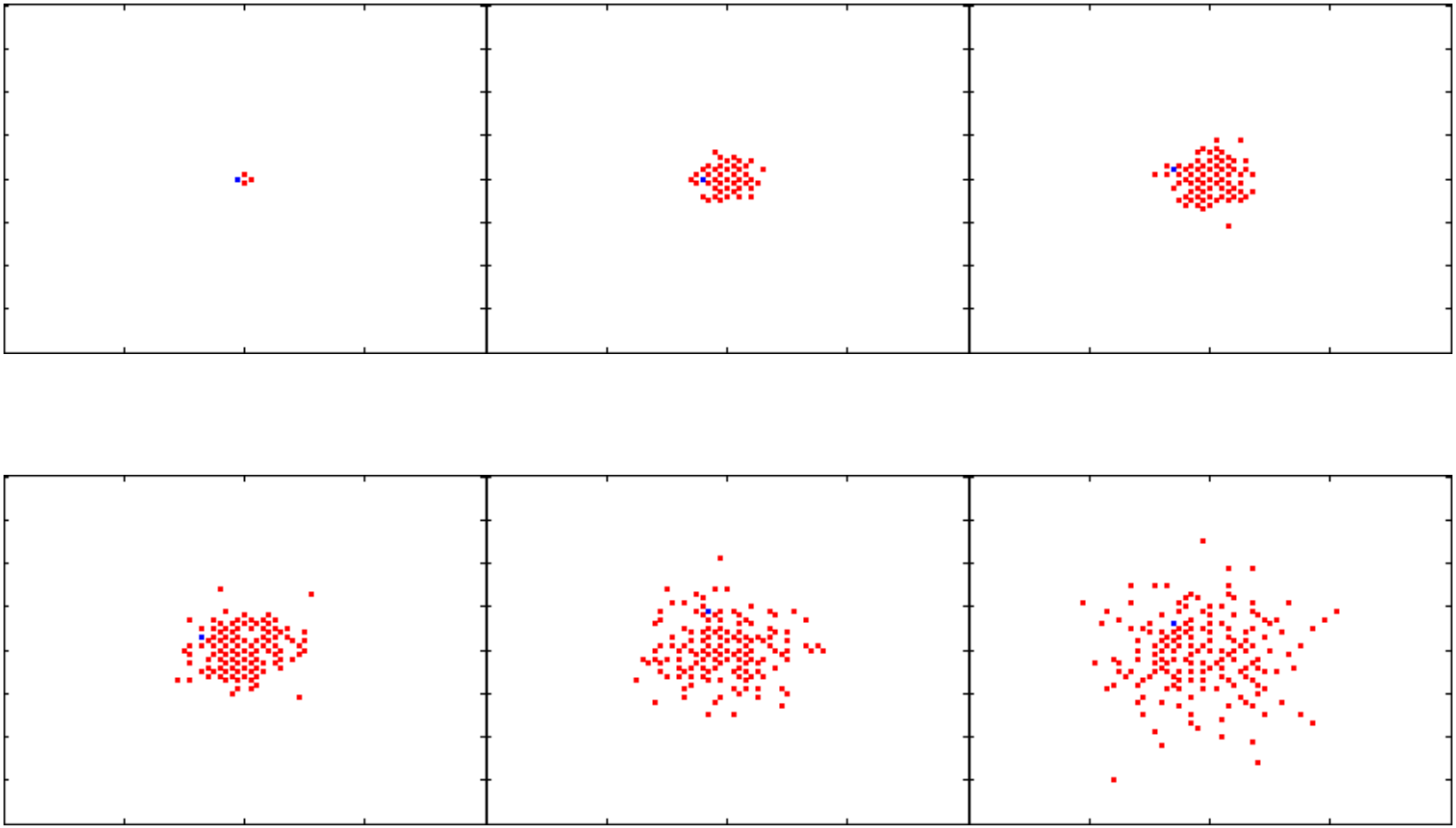
Brownian motion also explained the phenomenon of **diffusion**, in which a drop of ink in water slowly expands and mixes.

As particles of ink randomly collide with water molecules, they spread and mix, without requiring any stirring.

The mixing obeys the $\sqrt{T}$ law, so that, roughly speaking, if the diameter of the ink drop doubles in 10 seconds, it will double again in 40 seconds.

VirginiaTech

# Brownian Motion - Diffusion 1, 10, 20, 40, 80, 160 seconds



VirginiaTech

# Monte Carlo Method: Simulation

- A Traffic Simulation
- Game Theory: Duels
- Gambler's Ruin
- Brownian Motion
- **Random Walks**
- Coding a Three-Way Duel
- A Model of Epidemics

VirginiaTech

# Random Walks

The physical phenomenon of Brownian Motion had been explained by assuming that a pollen grain was subjected to repeated random impulses. This model was intriguing to physicists and mathematicians, and they soon made a series of abstract, simplified versions of it whose properties they were able to analyze, and which often could be applied to new problems.

The simplest version is known as the **Random Walk in 1D**.

VirginiaTech

# Random Walks - 100 Drunken Sailors

We'll introduce the random walk with a story about a captain whose ship was carrying a load of rum. The ship was tied up at the dock, the captain was asleep, and the 100 sailors of the crew broke into the rum, got drunk, and staggered out onto the dock.

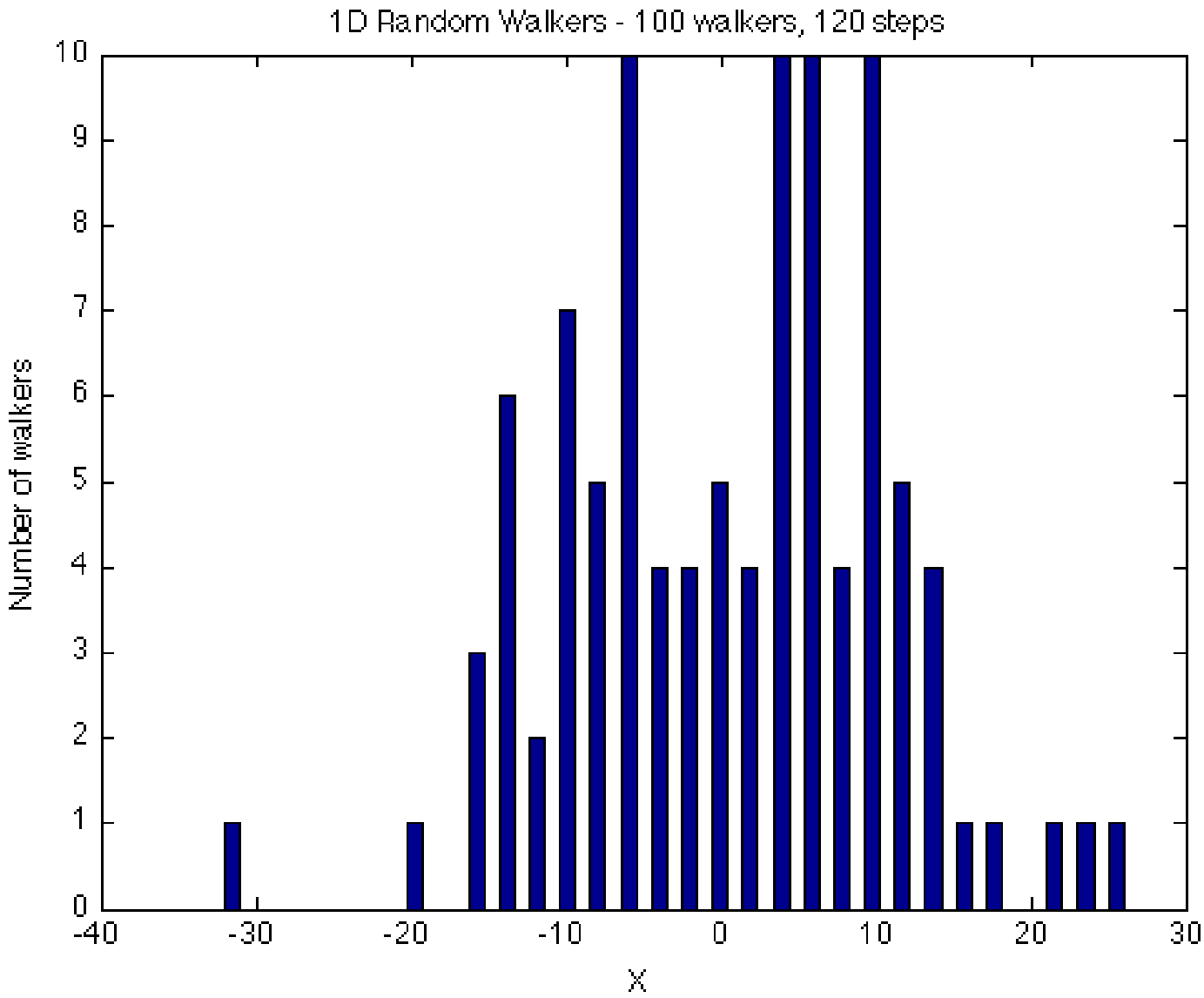
The dock was 100 yards long, and the ship was at the 50 yard mark. The sailors were so drunk that each step they took was in a random direction, to the left or right. They were only able to manage one step a minute. Two hours later, the captain woke up.

"Oh no!" he said, "There's only 50 steps to the left or right and they fall into the sea! And two hours makes 120 steps!"

But he was surprised to see that 70 of the crew were actually within 12 steps to the left or right, and that **all** of the crew was alive and safe, though in sorry shape.

VirginiaTech

# Random Walks - 100 Drunken Sailors




aTech

# Random Walks - 100 Drunken Sailors

What explains the huge disparity between how far the sailors could have gotten, and how far they did get? Taking **N** random steps is like adding up random +1's and -1's, a process that follows the normal distribution; the average of such a sum tends to zero, with an error that is roughly $\frac{1}{\sqrt{N}}$.

$$|\text{average} - 0| \sim \frac{1}{\sqrt{N}}$$

then

$$|\frac{\text{sum}}{N}| \sim \frac{1}{\sqrt{N}}$$

and so

$$|\text{sum}| \sim \sqrt{N}.$$

VirginiaTech

# Random Walks - The Random Walk in 1D

Our model for a random walk in 1D is very simple. We let **X** represent the position of a point on a line, assume that at step **N** $= 0$ the position is $\mathbf{X(0)} = 0$, and on each new step, we move one unit left or right, chosen at random.

From what we just said, we can expect that after **N** steps, the distance of the point from 0 will on average be about $\sqrt{N}$. If we compare **N** to the *square* of the distance, we can hope for a nice straight line.

VirginiaTech

# Random Walks - The Random Walk in 1D

How many random walks of **N** steps are there? There must be $2^N$, because it's like flipping a coin **N** times.

How many of those random walks end up at a given position?

There's only one that can end at **N**.

There are **N** walks that end at **N** - 2; they involve **N**-1 steps of $+1$'s and a single -1 step which can happen at any of **N** places.

So there must be $\binom{N}{2}$ that end up at **N** - 4; that's how many strings of (**N**-2) H's and 2 T's we can have.

VirginiaTech

# Random Walks - The Random Walk in 1D

So in general, there are $\binom{N}{K}$ distinct random walks that will end up at **N - 2*K**.

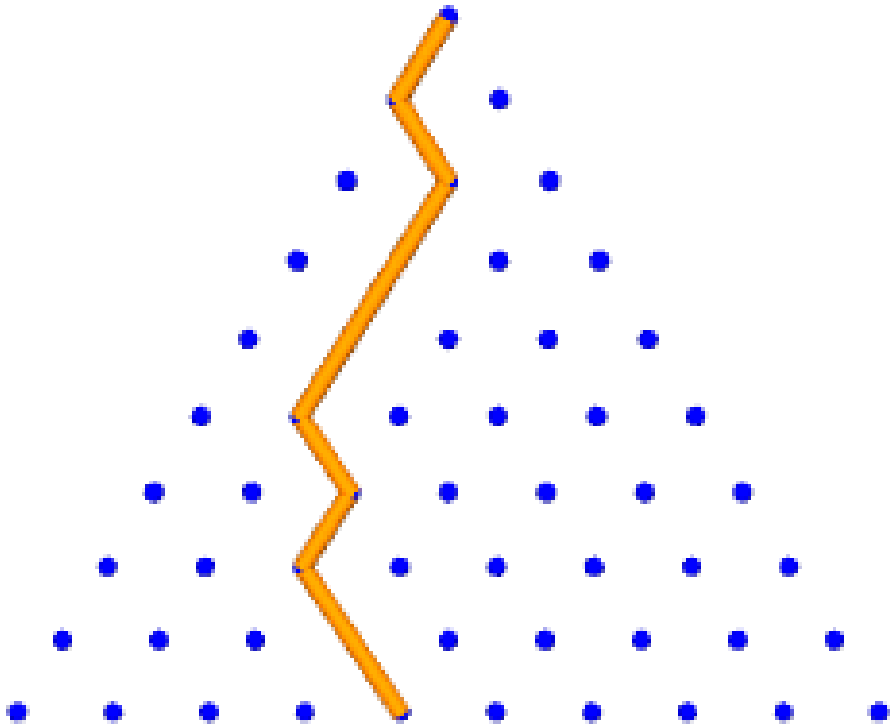
That means the **probability** of ending up at a particular spot is just the corresponding combinatorial coefficient divided by $2^N$.

In other words, row **N** of Pascal's triangle tells you what a random walk can do. But row **N** of Pascal's triangle can be thought of as dropping balls in a pachinko game that can randomly go left or right at each level!

VirginiaTech

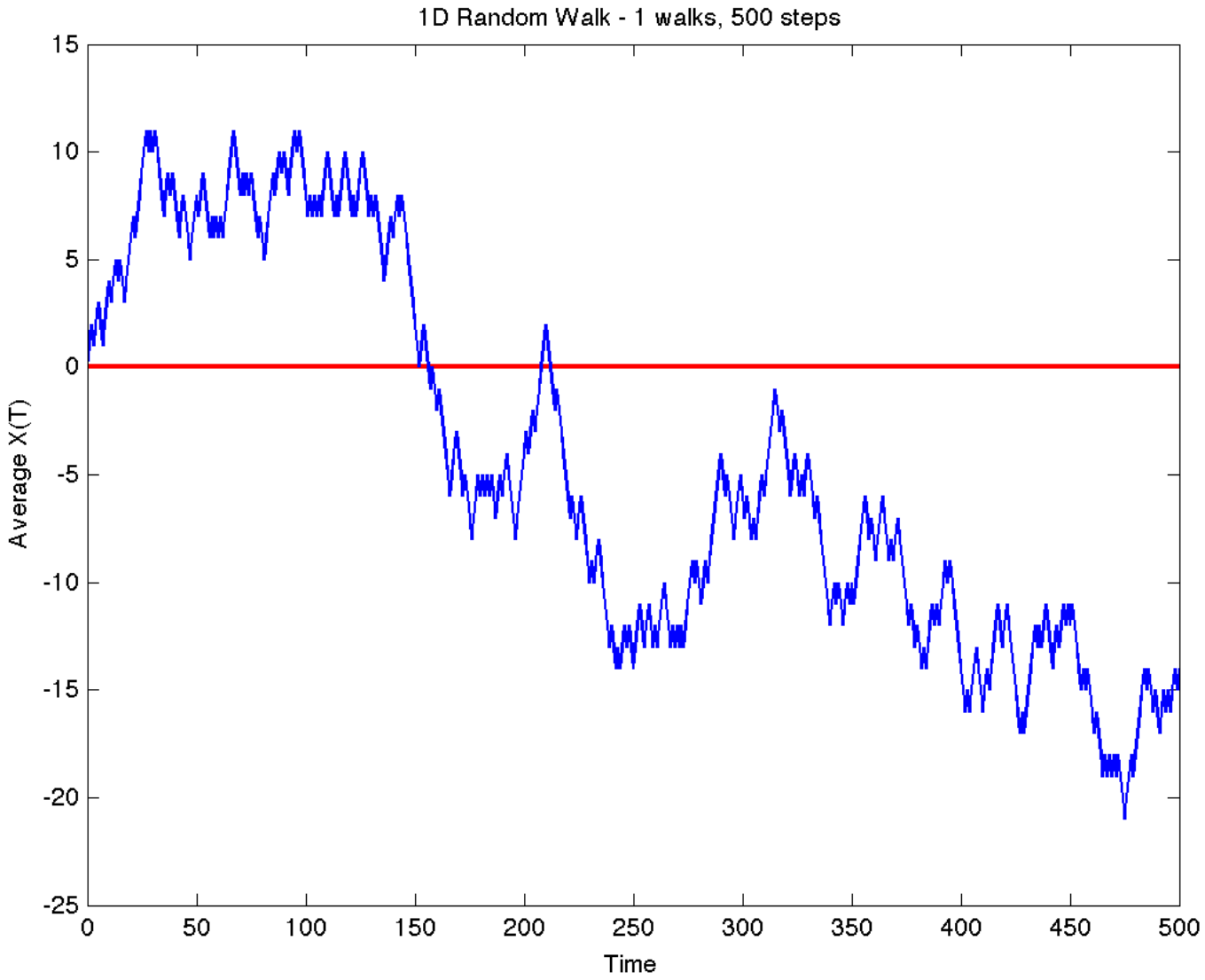
# Random Walks - Pascal's Pachinko

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 9 | 36 | 84 | 126 | 126 | 84 | 36 | 9 | 1 |
| .002 | .017 | .070 | .164 | .246 | .246 | .164 | .070 | .017 | .002 |



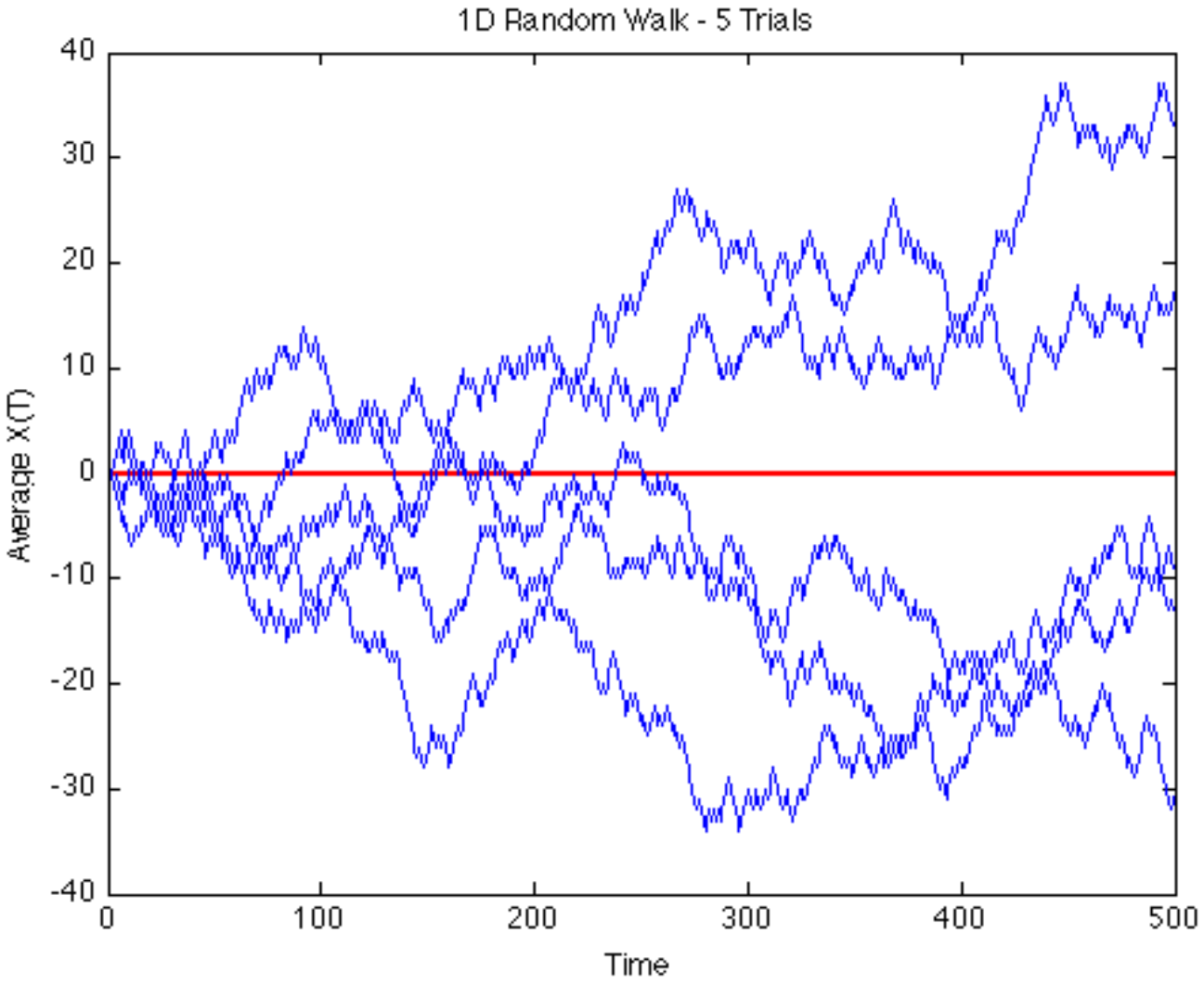
VirginiaTech

# Random Walks - 1 Walk of 500 Steps



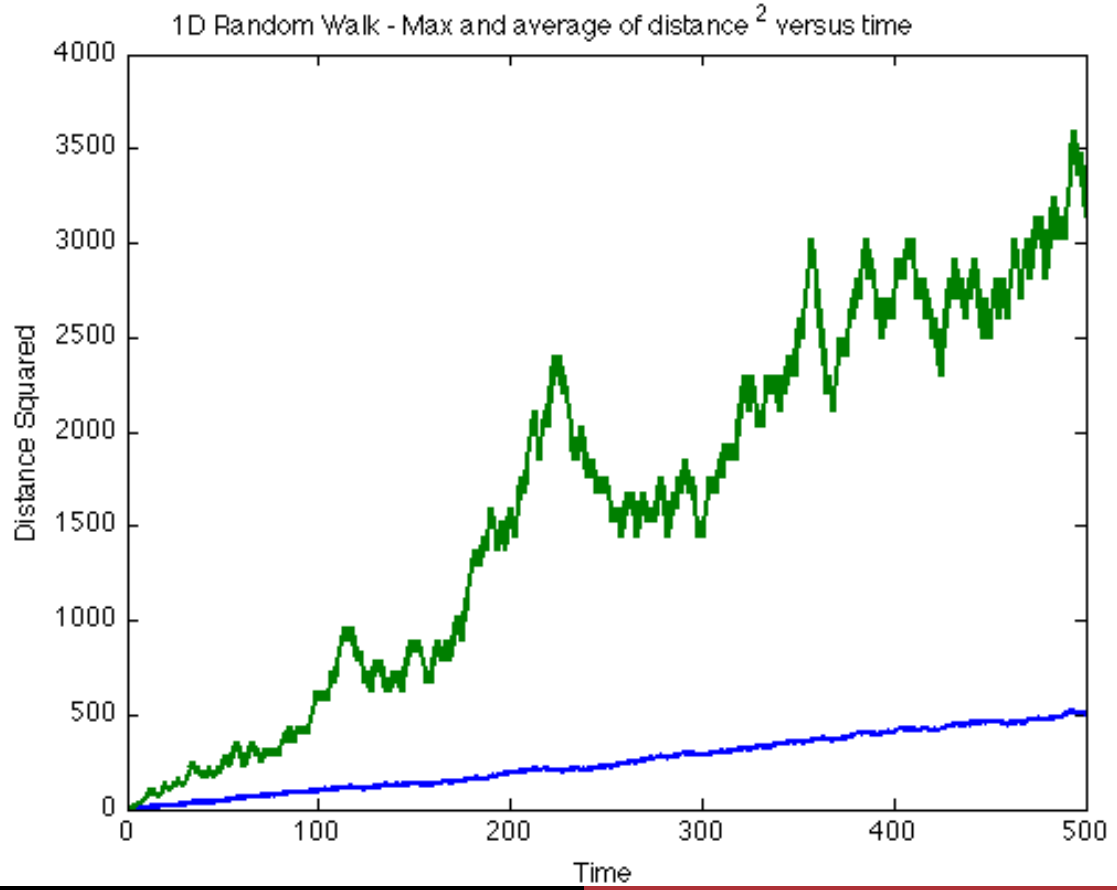

aTech

# Random Walks - 5 Walks of 500 Steps



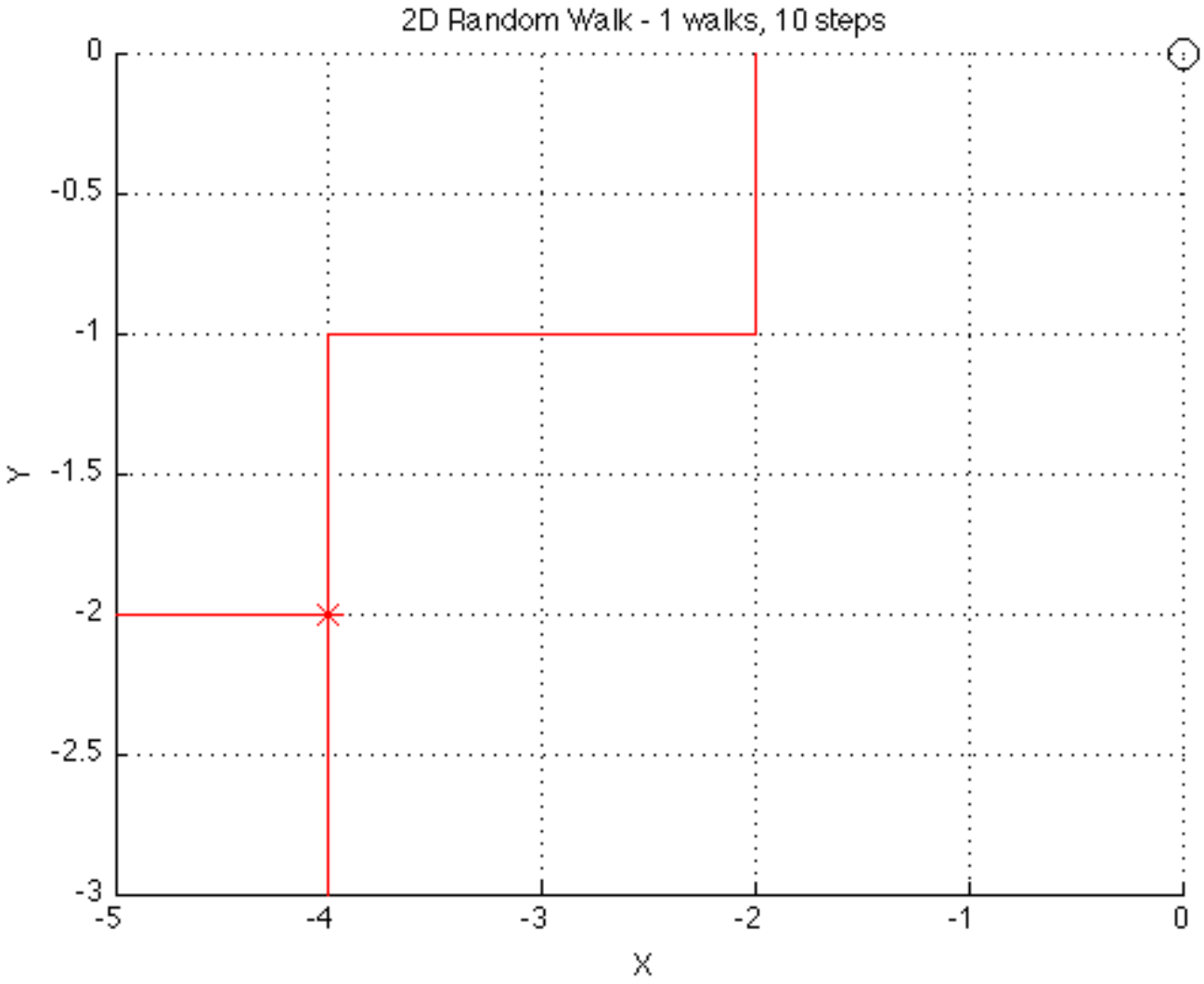
# Random Walks - 100 Walks of 500 Steps

Despite the wild variation in pattern, if we average quantities associated with a random walk, we tend to get well-behaved statistics. Here is a plot of the maximum distance (not well behaved) and the averaged distance (very close to a straight line)

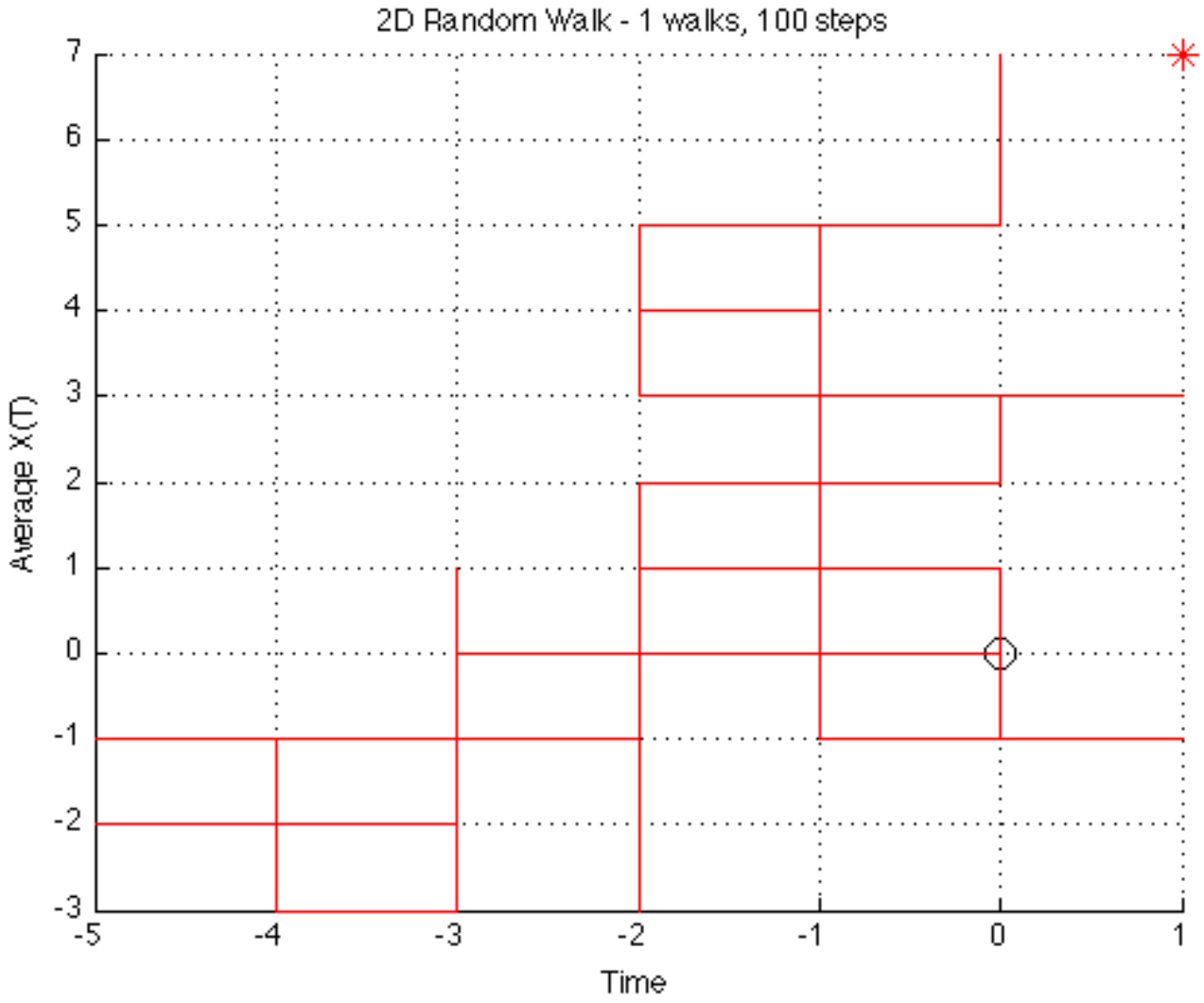



VirginiaTech

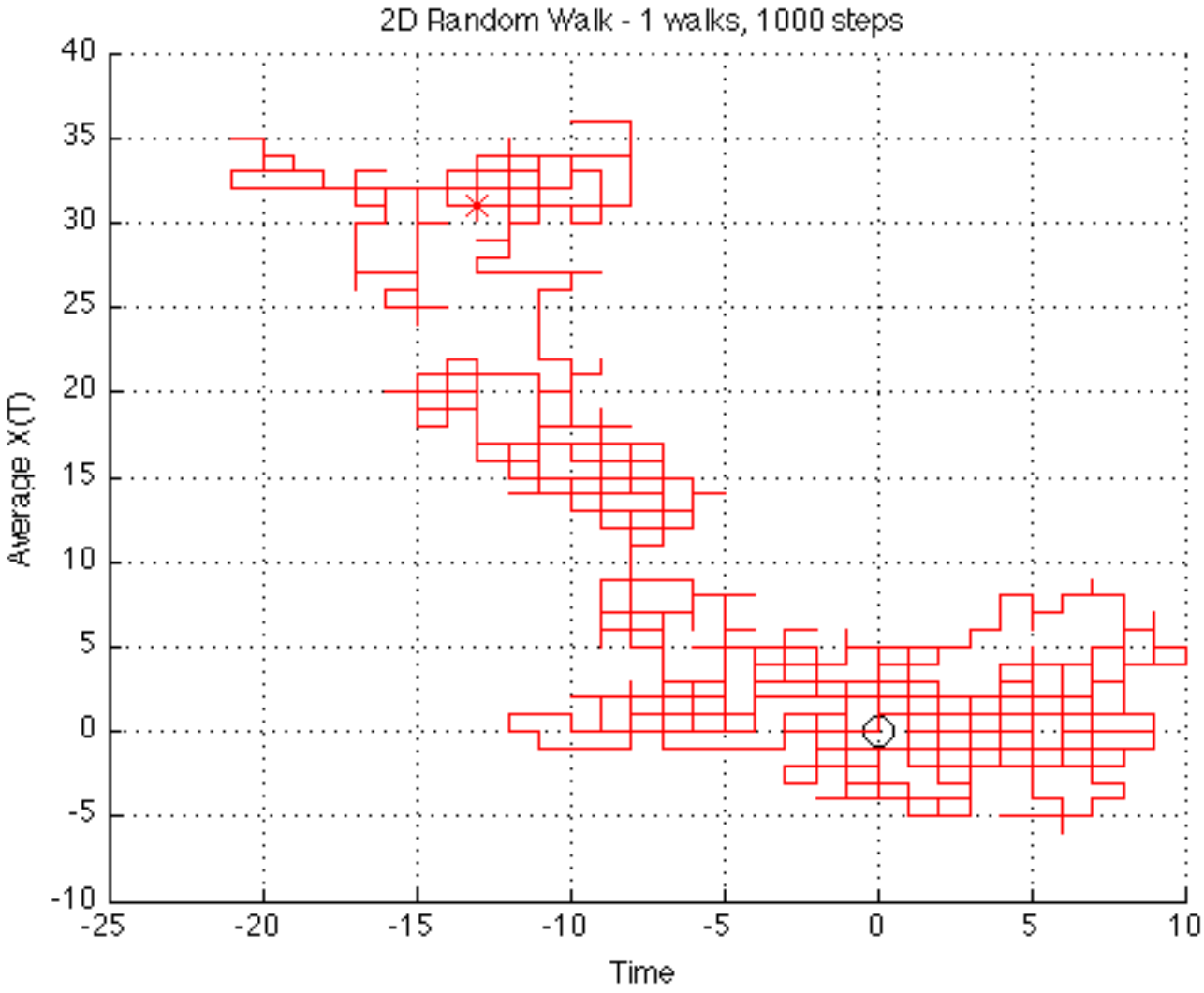
# Random Walks - 10 Steps in 2D



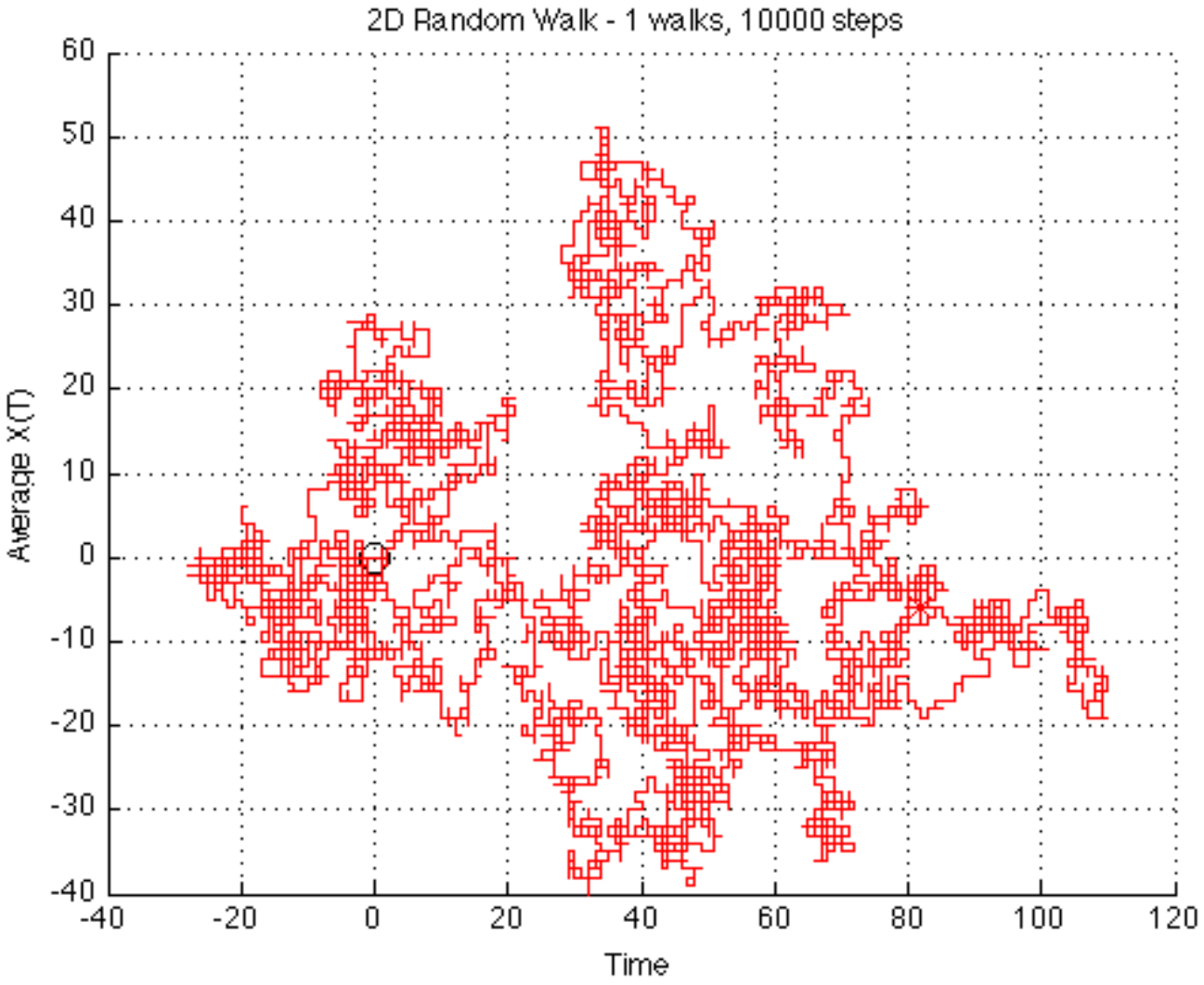
# Random Walks - 100 Steps in 2D



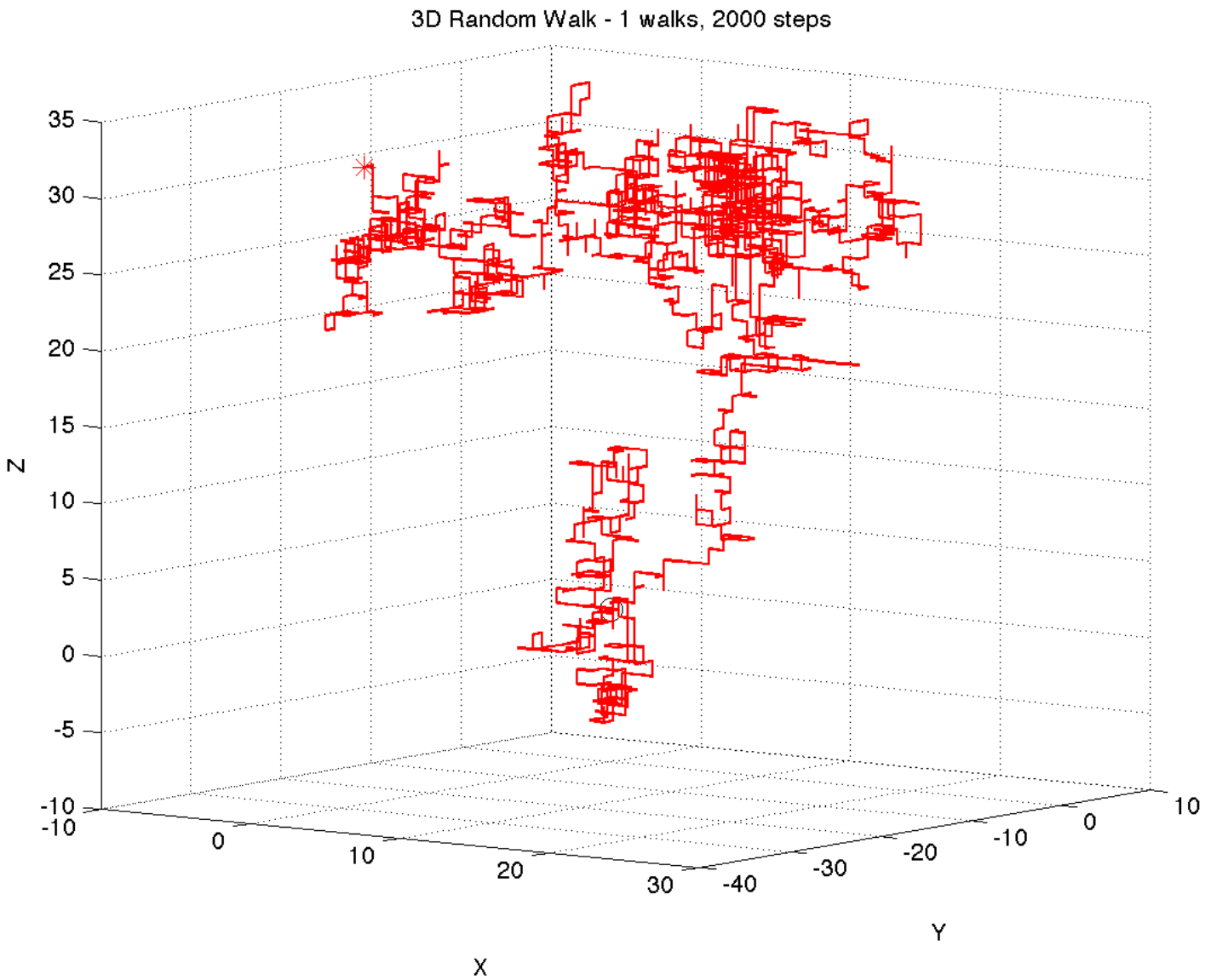
# Random Walks - 1000 Steps in 2D



# Random Walks - 10000 Steps in 2D



# Random Walks - 1 3D Walk of 2000 Steps




aTech

# Random Walks - Self Avoidance

The random walk model produces a squiggle that is a good imitation of how a particle might move through space under random influences.

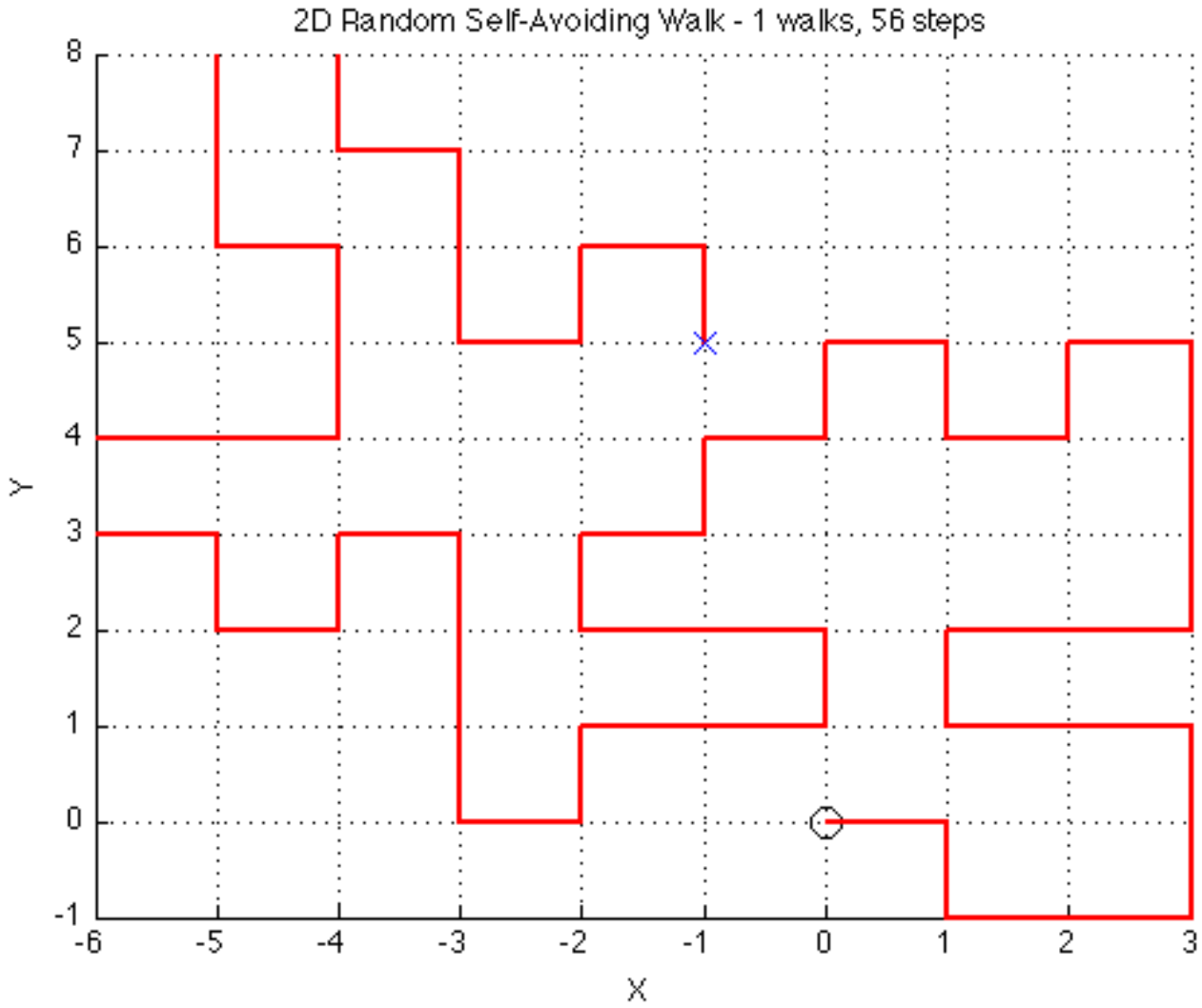
What if we were trying, instead, to model the way a piece of spaghetti arranged itself in space? The biggest problem is that we can't let the spaghetti go through itself.

If we add this requirement to the model, we are talking about **self avoiding random walks**. These are much harder to generate!

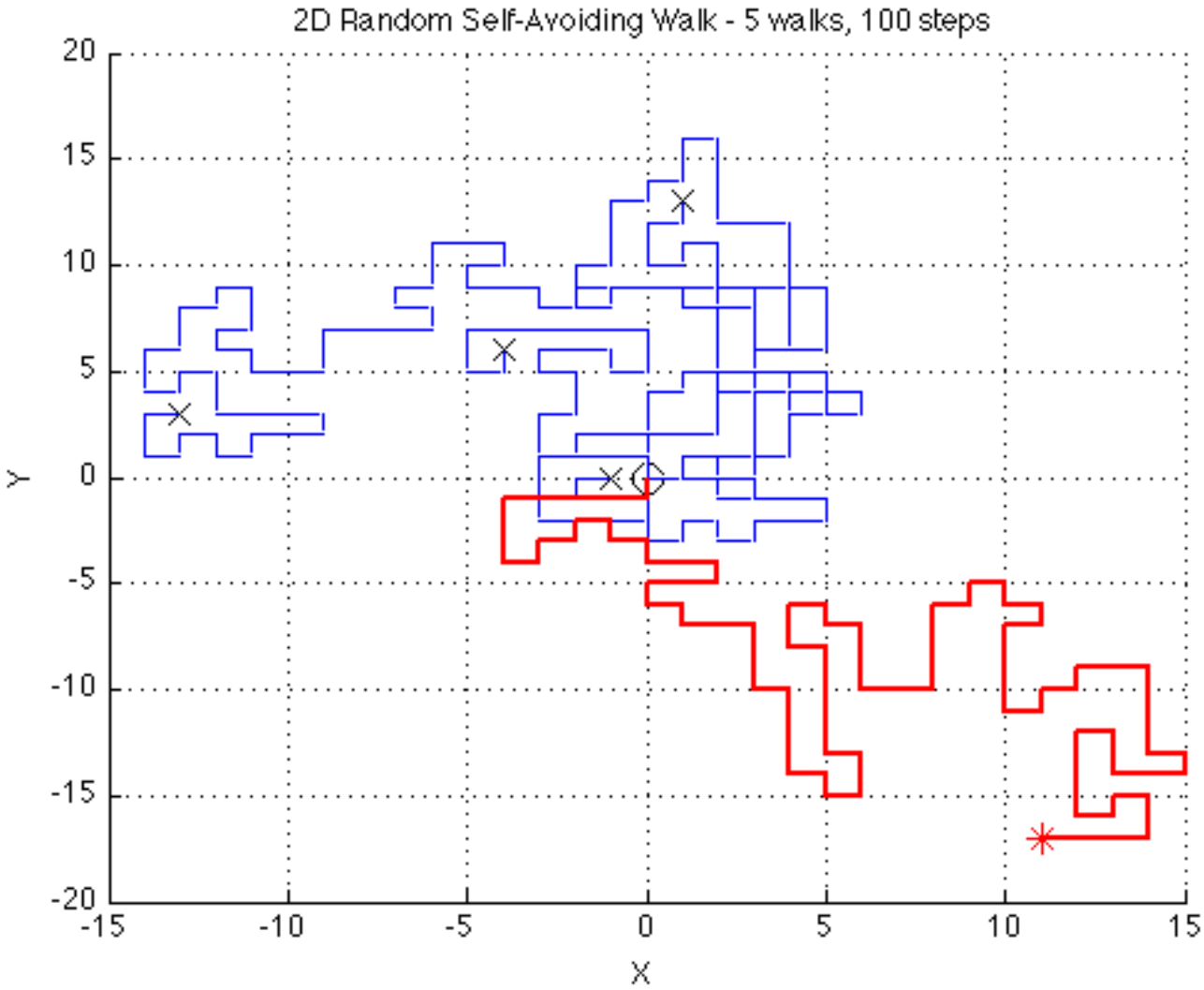
The 1D problem is uninteresting, so we go straight to 2D!

VirginiaTech

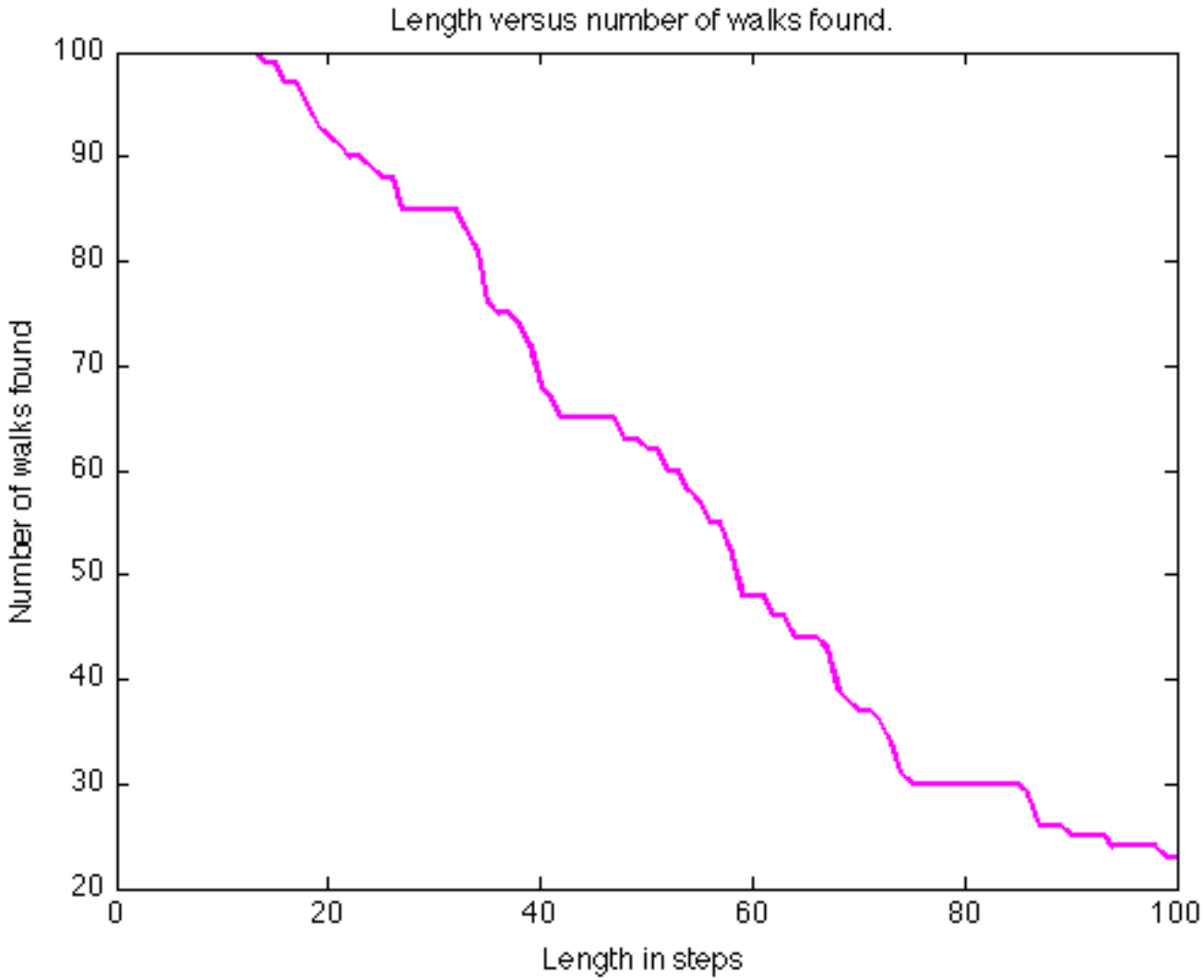
# Random Walks - A Failed Attempt at Self Avoidance



# Random Walks - A Self Avoiding Walk of Length 100



# Random Walks - 100 Tries at Self Avoidance



# Random Walks - Self Avoidance

Proteins are linear strings of amino acids. But in life, when they are generated, they fold up in a very specific way, based on attractions between particular amino acids.

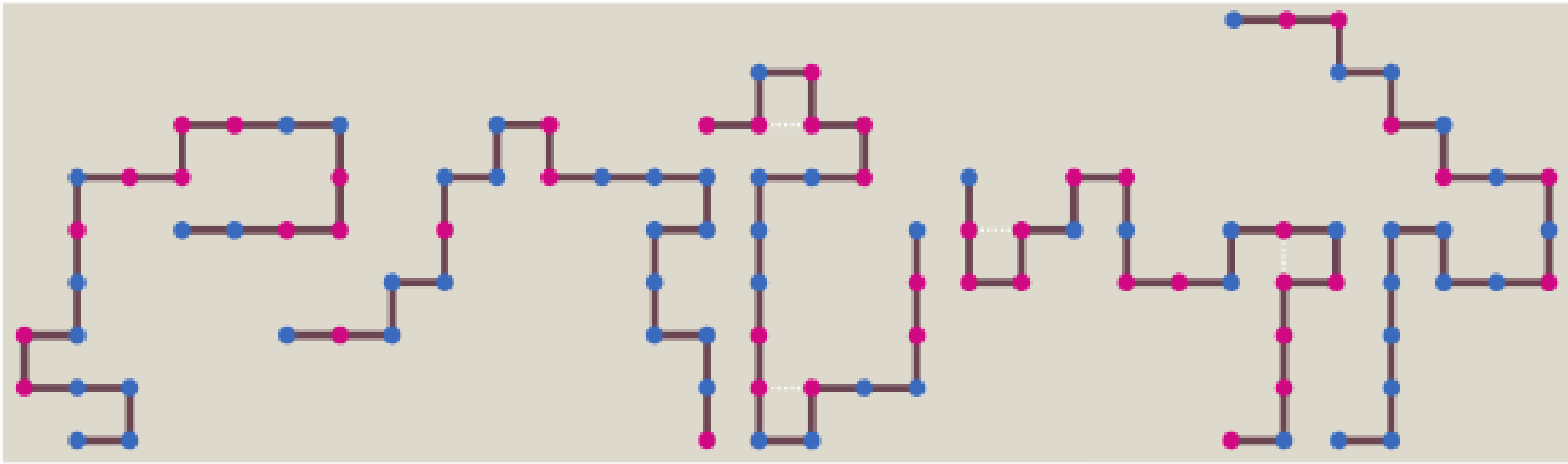
Trying to predict, from the chemical formula, how a protein will fold up, is a huge problem in the biological sciences.

A simplified model, called *prototeins*, considers the way a string of blue and red beads can fold. The red beads attract each other, and the blue ones don't care.

Given a specific string of blues and reds, we must consider all self-avoiding random walks, and seek ones of lowest energy.

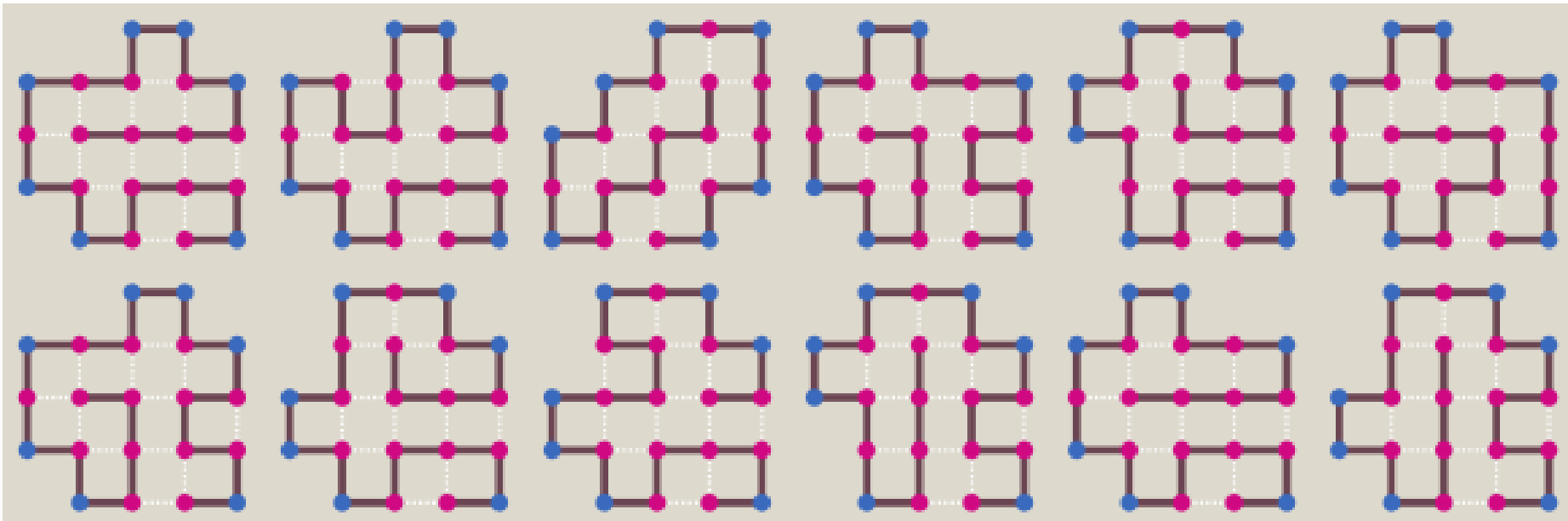
VirginiaTech

# Random Walks - Random Prototeins



VirginiaTech

# Random Walks - Stable Prototeins



VirginiaTech

# Random Walks - Self Avoidance

If we allow a self-avoiding random walk to terminate at its beginning, then we have a (self-avoiding) loop.

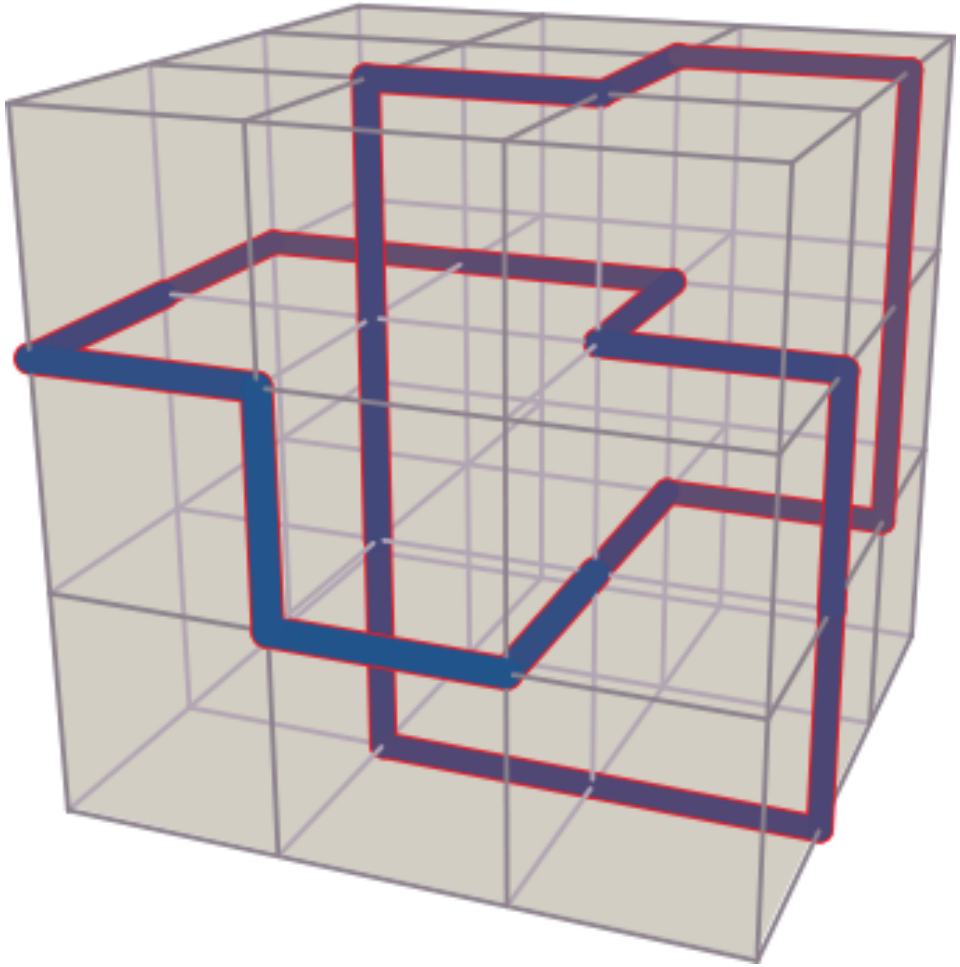
In 2D, all loops are topologically the same.

In 3D, a loop can be *knotted*, perhaps more than once.

By generating random self-avoiding loops, we can search for configurations that represent knots, and try to classify them.

VirginiaTech

# Random Walks - A 3D Knot



VirginiaTech

# Monte Carlo Method: Simulation

- A Traffic Simulation
- Game Theory: Duels
- Gambler's Ruin
- Brownian Motion
- Random Walks
- **Coding a Three-Way Duel**
- A Model of Epidemics

VirginiaTech

# Three-Way Duel

The three-way duel simulation is a nice example; it's much more human than a partial differential equation, and it's easy to understand the results!

However, even this example can be tricky to program, especially when we have to go from a two-way duel to a three-way one.

And I would like you to imagine the possibility that you might have to program an "N-way" duel. Some of the choices we make for the simplest cases won't stretch that far!

VirginiaTech

# Three-Way Duel - How We did a Two-Way Duel

```
turn_num = 0;
while ( 1 )
  turn_num = turn_num + 1;  %  Player 1 fires.
  r = rand ( );
  if ( r <= p(1) )
    survivor = 1;
    break
  end
  turn_num = turn_num + 1;  %  Player 2 fires.
  r = rand ( );
  if ( r <= p(2) )
    survivor = 2;
    break
  end
end
```

VirginiaTech

# Three-Way Duel - Three-Way is Harder

The three-way duel is harder; for one thing, we can't assume that as soon as one person is eliminated, the game is over. So the idea that if the shot is accurate, the game is over and the shooter is the winner has to be revised.

There is also the issue of *who to aim at*; we are basing this on the idea that everyone wants to eliminate the person with the best accuracy (except, of course, that person, who will aim at the second-most-accurate person!).

It's possible to code this by writing out what **A** will do, then what **B** will do and so on. This is straightforward, but cluttered. It might be better to try to **abstract** what is going on.

So let's concentrate on one **arbitrary** player whose index is **I**.

VirginiaTech

# Three-Way Duel - Pseudocode for Player "I"'s Turn

```
IF ( I am alive )
: turn_num = turn_num + 1
: IF ( two other players are alive )
: : TARGET = the better one
: ELSE
: : TARGET = the remaining player.
: END
: r = rand ( );
: IF ( r <= p(I) )
: : TARGET is eliminated.
: : IF ( I am the ONLY remaining player now)
: : : survivor = I; break;
: : END
: END
END
```

# Three-Way Duel - Pseudocode to Code?

To turn our pseudocode into code, we need an efficient way to keep track of who's in the game and who's out!

One way to do this is to play with the **P** array. Once a person's dead, we could set their **P** value to zero.

That also makes it easy to choose the target. Always pick the person with the biggest **P** value.

*But wait...how do I avoid shooting myself?*

VirginiaTech

# Three-Way Duel - MATLAB for Player "I"'s Turn

```
turn_num = turn_num + 1;
p_save = p(i);
p(i) = 0.0;
[ pmax, target ] = max ( p ); <-- TARGET is index of max
r = rand ( );
if ( r <= p_save )
  p(target) = 0.0;
  if ( sum ( p ) == 0.0 )
    survivor = i;
    break;
  end
end
p(i) = p_save;
```

# Three-Way Duel - Conclusion

This is not the best way, or the only way, but it is a way, and it has some real advantages.

There are hardly any extra variables (except for **p_save**).

The code works the same for every player. We never assume that the players were given in a particular order.

The code works the same if we increase the number of players.

Of course, adding the part about C's optimal strategy would make this a little harder...but not too much!

VirginiaTech

# Monte Carlo Method: Simulation

- A Traffic Simulation
- Game Theory: Duels
- Gambler's Ruin
- Brownian Motion
- Random Walks
- Coding a Three-Way Duel
- **A Model of Epidemics**

VirginiaTech

# Disease Transmission - Disease Factors

For an epidemiologist, such as Dr John Snow, there are many aspects to disease.

For a given disease there may be:

- a cause, such as a bacteria;
- a carrier, such as polluted water;
- a means of getting the disease - ingestion of polluted water;
- a means of passing the disease, - bacteria in body fluids;
- a latent phase, when a sick person shows no symptoms;
- varying susceptibility to the disease among the population;
- resistance, from people who have recovered from an attack:
- vaccines, that convey resistance.

VirginiaTech

# Disease Transmission - The SIR Model

To try to control a disease outbreak, it is useful to have a model which embodies our understanding or best guess for how the disease spreads, and which factors might be the best ways to control an outbreak.

An early influential model of disease transmission is known as the **SIR** model. It assumes that a disease outbreak is in progress, and it divides the populuation into three groups:

- **Susceptibles**, well people who could get the disease;
- **Infecteds**, sick people who carry the disease;
- **Recovereds**, people who had the disease, but recovered.

VirginiaTech

# Disease Transmission - The FLEMS

So that we have something to work with, let's make some choices.

- We'll assume our disease is called *the FLEMS*;
- Only a susceptible person **'S'** can get the FLEMS;
- A person who gets the FLEMS immediately becomes an infected person **'I'**;
- An **'I'** person stays sick *('flemish')* for **K** days;
- An **'I'** person can transmit the disease to nearby **'S'** people;
- The probability of transmission of the disease over one day's contact is **TAU**;
- After **K** days, an **'I'** person becomes an **'R'** recovered person, and can never get the disease or transmit it.

VirginiaTech

# Disease Transmission - Adding Geometry

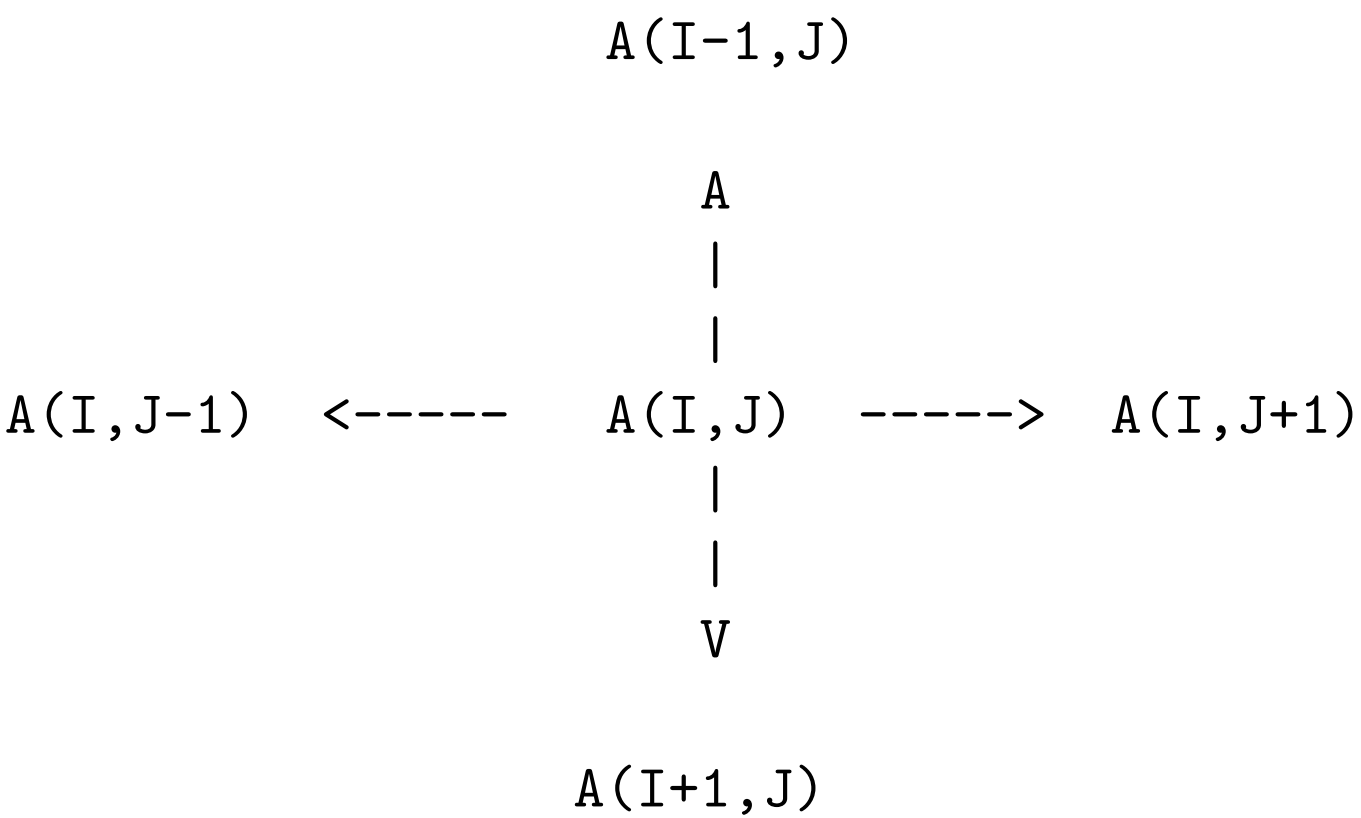
Since our disease is transmitted only between nearby people, we need to add some geometry to our problem. Again, to make things simple, we'll assume our population is a group of patients in a huge open hospital ward, awaiting appendix transplant operations.

So there's initially nothing wrong with any of them. Well, except for one or two people who are actually...*flemish*!

Let's say the hospital ward is an array **A** of **M** by **N** beds (how convenient for MATLAB!), and that a sick person is only in contact with people in the beds to the "north", "south" "east" or "west".

VirginiaTech

# Disease Transmission - North, South, East, West



# Disease Transmission - Parameters

So at this point, the parameters in our model include:

- **K**, the number of days an 'I' person stays sick;
- **TAU**, the probability of transmission from an 'I' person to a nearby 'S' person;
- the values of **M** and **N** for the hospital ward;
- the initial assignment of 'S', 'I' and 'R' people to beds.

and we will also add one more parameter, **T_MAX**, the number of days we want to follow the course of the disease.

VirginiaTech

# Disease Transmission - Parameter Values

Now it's time to pick some specific values for a simulation:

- Let the sickness last **K** = 4 days;
- Let the transmission probability be **TAU** = 0.2;
- Let there be **M** = 10 rows and **N** = 10 columns of beds;
- Initialize the patients to type **'S'**;
- Set patient (5,5) to be type **'I'**;
- Let the simulation run for **T_MAX**=50 days.

VirginiaTech

# Disease Transmission - Patient Status

How will we represent the status of the patients?

We can use the **A** array to keep track of everything:

- **S** patients can have A(I,J) = 0;
- **I** patients have A(I,J) = 1, 2, 3, 4 or 5 (how long they've been sick);
- **R** patients can have A(I,J) = -1.

Each step of the simulation updates **A** for the next day.

VirginiaTech

# Disease Transmission - Simulation Code

```
    a_new = zeros ( m, n );  <-- Update the A array.
    for i = 1 : m
      for j = 1 : n
%  Recovered never change.
        if ( a(i,j) == -1 )
          a_new(i,j) = -1;
%  Infected, and less than K days, increase by 1.
        elseif ( 1 <= a(i,j) & a(i,j) < k )
          a_new(i,j) = a(i,j) + 1;
%  Infected K days becomes recovered.
        elseif ( a(i,j) == k )
          a_new(i,j) = -1;
%  Susceptible, look for infected neighbors.
        else
```

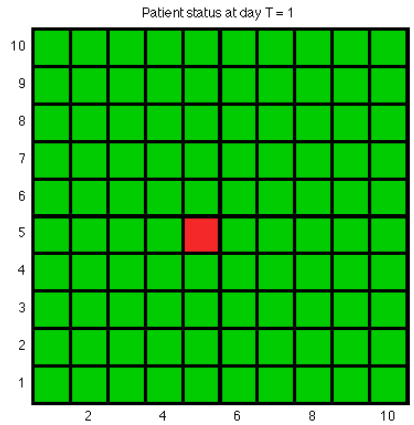
# Disease Transmission - Simulation Code

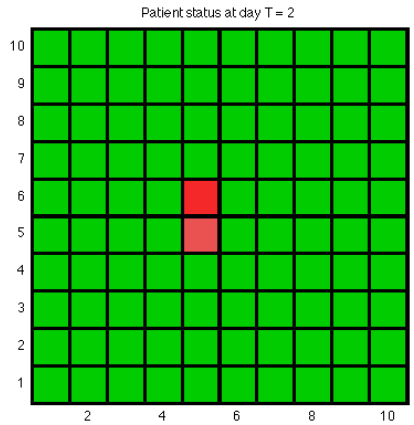
```
%  Susceptible, look for infected neighbors.
  a_new(i,j) = 0;
  if ( 1 < i & 0 < a(i-1,j) & a(i-1,j) <= k )  <-- North
    if ( rand ( 1, 1 ) <= tau )
      a_new(i,j) = 1;
    end
  end

  if ( i < m & 0 < a(i+1,j) & a(i+1,j) <= k )  <-- South
    if ( rand ( 1, 1 ) <= tau )
      a_new(i,j) = 1;
    end
  end
```

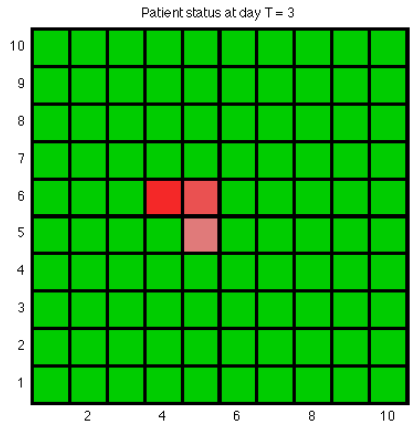
(and also East and West)

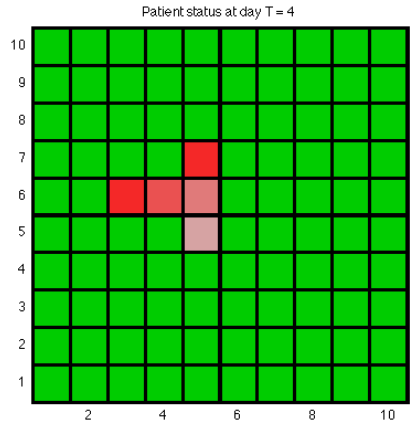
VirginiaTech

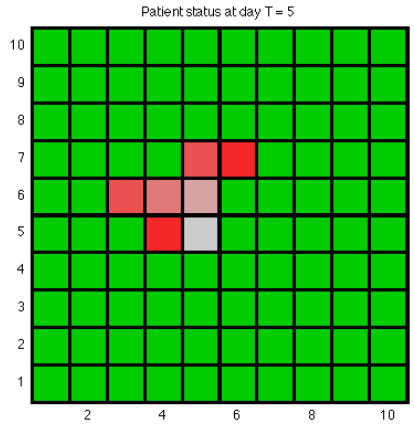
# Disease Transmission - Days 1, 2, 3, 4, 5, 6

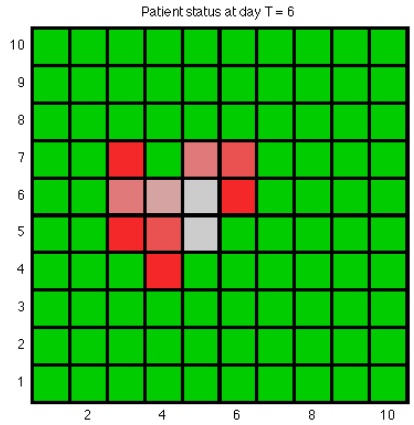


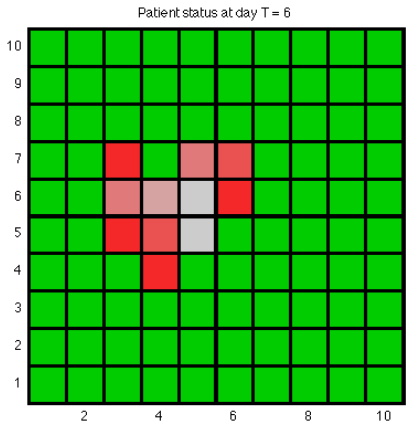



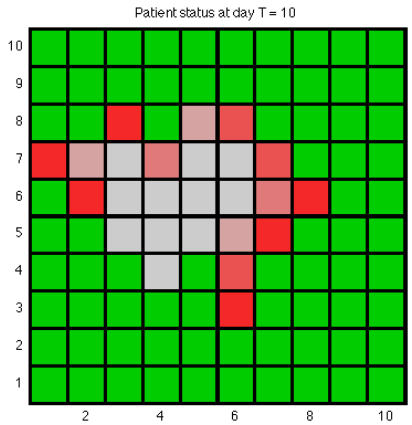



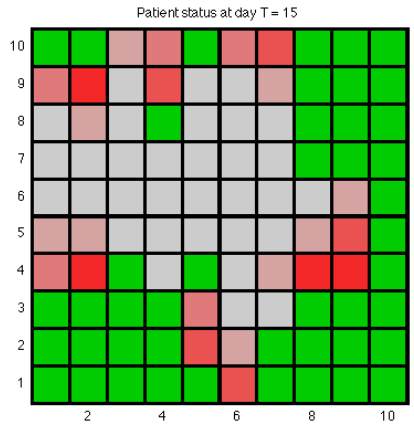



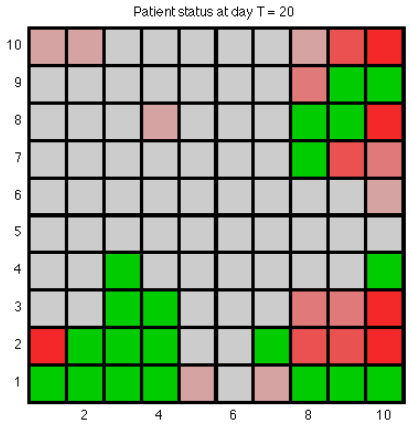



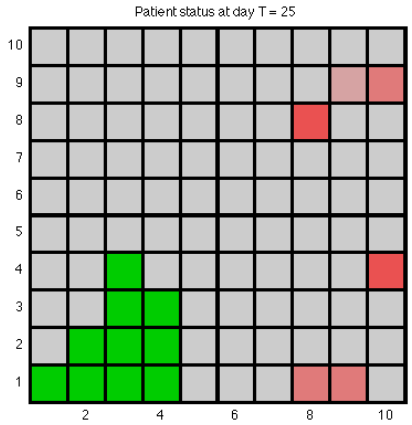
# Disease Transmission - Days 6, 10, 15, 20, 25, 30

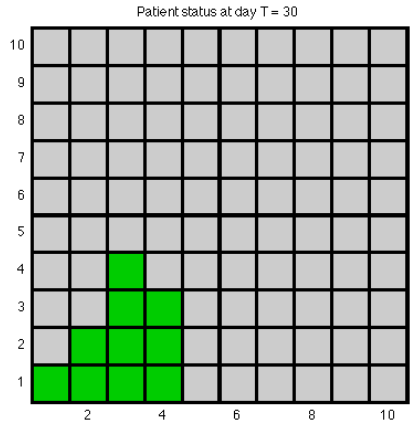


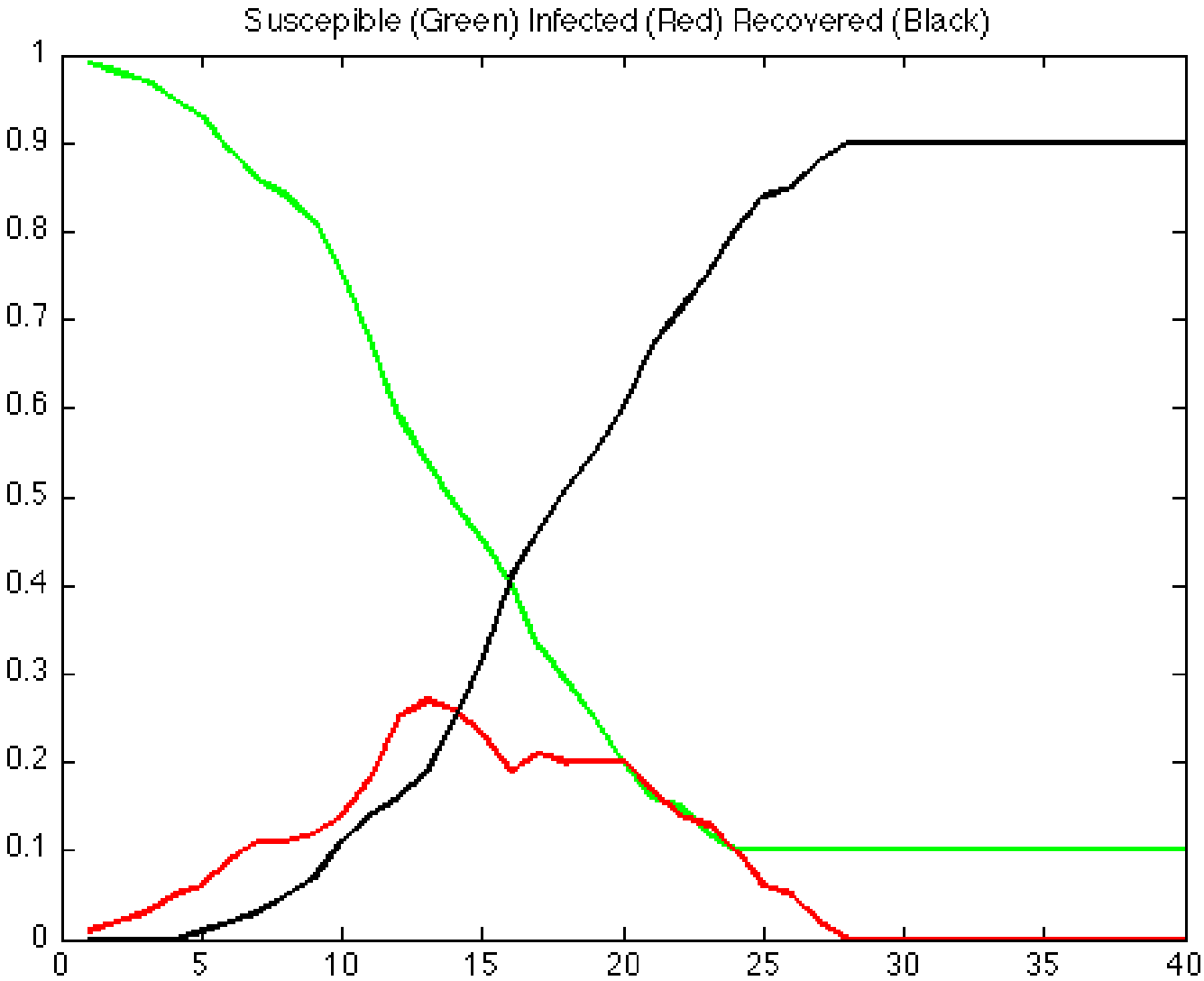



# Disease Transmission - Proportions of S, I and R




aTech

# Disease Transmission - Summary

Our SIR model is a very simplified picture of an epidemic.

It very much includes the effects of randomness. From the same initial condition, I have gotten no new infections, or everyone infected, as well as the typical case in which about 3/4 of the patients get sick.

So as with any randomized simulation, you need to make many runs and then draw conclusions from the average, the minimum and maximum, and the variance.

It's easy to include new features, such as **vaccination**, or the chance that some people could get the disease multiple times.

A good computer simulation makes new experiments easy.

VirginiaTech

# Monte Carlo Method: Simulation

- Overview
- A Traffic Simulation
- Game Theory: Duels
- Gambler's Ruin
- Brownian Motion
- Random Walks
- A Model of Epidemics
- **Conclusion**

VirginiaTech

# Monte Carlo Simulation - Conclusion

A model needs to be simple enough that we can work with it, while allowing us to capture some interesting features of a process.

Often we have to guess for *reasonable parameter values*; we may be able to correct them later by comparing the simulation results to reality.

Extremely simple models, such as the random walk, can nonetheless provide some information about a process, or at least give us a way to think about what is happening.

Especially when *random events* are included, such as in traffic, simulation is the only method that can provide rough answers or a feeling for how a process works.

VirginiaTech