Errors inherent to the modeling process

Models always have errors $\rightsquigarrow$

- Be aware of them.
- Understand and account for them!
- Include in model discussion.

Errors inherent to the modeling process

Models always have errors $\rightsquigarrow$

- Be aware of them.
- Understand and account for them!
- <u>Include in model discussion.</u>

Types of Errors

1. **Formulation Errors** occur when simplifications or assumptions are made. **($\star$)**
2. **Observation Errors** occur during data collection. **($\star$)**

Errors inherent to the modeling process

Models always have errors $\rightsquigarrow$

- Be aware of them.
- Understand and account for them!
- <u>Include in model discussion.</u>

Types of Errors

1. **Formulation Errors** occur when simplifications or assumptions are made. **($\star$)**
2. **Observation Errors** occur during data collection. **($\star$)**
3. **Truncation Errors** occur when you approximate an incalculable function.
4. **Rounding Errors** occur during calculations when your computing device can't keep track of exact numbers.

Errors inherent to the modeling process

1. **Formulation Errors** occur when simplifications or assumptions are made.

Example from the book, pp. 70–73: Seismology.

Set off an explosion at one place and measure it at another (dist. D). Create a model to determine the depth of a layer in the crust based on the time for the initial explosion to arrive T, and the second shock T'.

$$d = \frac{D}{2}\sqrt{(T'/T)^2 - 1}$$

Assumptions: The earth is flat, and the layer is parallel to the surface.

Errors inherent to the modeling process

1. **Formulation Errors** occur when simplifications or assumptions are made.

Example from the book, pp. 70–73: Seismology.

Set off an explosion at one place and measure it at another (dist. D). Create a model to determine the depth of a layer in the crust based on the time for the initial explosion to arrive T, and the second shock T'.

$$d = \frac{D}{2}\sqrt{(T'/T)^2 - 1}$$

Assumptions: The earth is flat, and the layer is parallel to the surface.

If layers are not parallel (off by α°), the percent errors can be large!

α	1	5	10	30
% error in d	3.4	18	37	105

Errors inherent to the modeling process

2. **Observation Errors** occur during data collection.

Continuation of the previous example:

Even if the layers are parallel, perhaps our timing is inaccurate.
Let's say that T is 1 second and T' is 1.2 seconds,
but that our timer is off by at most 1%.

Then T might be ____ seconds or ____ seconds,
and T' might be ____ seconds or ____ seconds.

Errors inherent to the modeling process

2. **Observation Errors** occur during data collection.

Continuation of the previous example:

Even if the layers are parallel, perhaps our timing is inaccurate.
Let's say that T is 1 second and T' is 1.2 seconds,
but that our timer is off by at most 1%.

Then T might be ____ seconds or ____ seconds,
and T' might be ____ seconds or ____ seconds.

T	over	over	under	under
T'	over	under	over	under
% error in d	-0.5%	-5%	+6%	0%

Errors inherent to the modeling process

2 **Observation Errors** occur during data collection.

Continuation of the previous example:

Even if the layers are parallel, perhaps our timing is inaccurate.
Let's say that T is 1 second and T' is 1.2 seconds,
but that our timer is off by at most 1%.

Then T might be ____ seconds or ____ seconds,
and T' might be ____ seconds or ____ seconds.

T	over	over	under	under
T'	over	under	over	under
% error in d	-0.5%	-5%	+6%	0%

One way to decrease influence: measure many times, take average.

Errors inherent to the modeling process

3. **Truncation Errors** occur when you approximate an incalculable function.

Question: When is $x^5 + x - 1 = 0$? What is $\sin 1$? Numerically?

Answer: Use a Taylor series approximation:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots .$$

Errors inherent to the modeling process

3. **Truncation Errors** occur when you approximate an incalculable function.

Question: When is $x^5 + x - 1 = 0$? What is $\sin 1$? Numerically?

Answer: Use a Taylor series approximation:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots .$$

4. **Rounding Errors** occur during calculations when your computing device can't keep track of exact numbers.

Question: What is 1.2300001^{10}?

Answer: If we only have three-digit accuracy, then

$1.23 \cdot 1.23 = 1.51, \qquad 1.23 \cdot 1.51 = 1.86 \qquad \ldots \qquad 1.23^{10} = \mathbf{7.95}$

Errors inherent to the modeling process

3. **Truncation Errors** occur when you approximate an incalculable function.

Question: When is $x^5 + x - 1 = 0$? What is $\sin 1$? Numerically?

Answer: Use a Taylor series approximation:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots .$$

4. **Rounding Errors** occur during calculations when your computing device can't keep track of exact numbers.

Question: What is 1.2300001^{10}?

Answer: If we only have three-digit accuracy, then

$1.23 \cdot 1.23 = 1.51, \qquad 1.23 \cdot 1.51 = 1.86 \qquad \ldots \qquad 1.23^{10} = \mathbf{7.95}$

$1.2300001 \cdot 1.2300001 = 1.5129002,$

$1.2300001 \cdot 1.5129002 = 1.8608674,$

$1.2300001^{10} = \mathbf{7.9259523}$

True answer: $7.925952539912863452584748018737649320039805 \cdots$

Simulation Modeling

Goal: Use probabilistic methods to analyze deterministic and probabilistic models.

Example. Determine the best elevator delivery scheme.

- The wait is too long, too many stops along the way.

Simulation Modeling

Goal: Use probabilistic methods to analyze deterministic and probabilistic models.

Example. Determine the best elevator delivery scheme.

- The wait is too long, too many stops along the way.
- *Inconvenient* to experiment with alternate delivery schemes.
    - Disrupt normal service
    - Take surveys of customers
    - Confuse regular customers

Simulation Modeling

Goal: Use probabilistic methods to analyze deterministic and probabilistic models.

Example. Determine the best elevator delivery scheme.

- The wait is too long, too many stops along the way.
- *Inconvenient* to experiment with alternate delivery schemes.
  - Disrupt normal service
  - Take surveys of customers
  - Confuse regular customers
- Alternatively, run a computer **simulation**. Write a computer program that models the system of elevators, including:
  - Time of arrival of passengers (a random event)
  - Passenger destination (a random event)
  - Capacity of elevator (fixed by system)
  - Speed of elevator (fixed by system)
  - Current delivery scheme

Simulation Modeling

Once you have written the computer program,

Verify that the simulation models the current real-world situation

- Run the model many times.
- Have the computer keep track of data, such as average wait time, number of stops it takes, longest queue, etc.

Then, modify various parameters in order to simulate a new delivery scheme.

- How do the data change?
- Is the alternate scheme better or worse?
- Determine how to implement to cause minimal disruption.

Monte Carlo Simulations

Definition: A simulation that incorporates an element of randomness is called a **Monte Carlo** simulation.

PROS:

CONS:

Monte Carlo Simulations

Definition: A simulation that incorporates an element of randomness is called a **Monte Carlo** simulation.

PROS:

- It is a relatively easy method to approximate complex systems.

CONS:

Monte Carlo Simulations

Definition: A simulation that incorporates an element of randomness is called a **Monte Carlo** simulation.

PROS:

- It is a relatively easy method to approximate complex systems.
- Once built, it allows for tinkering—easy to do sensitivity analysis.

CONS:

Monte Carlo Simulations

Definition: A simulation that incorporates an element of randomness is called a **Monte Carlo** simulation.

PROS:

- It is a relatively easy method to approximate complex systems.
- Once built, it allows for tinkering—easy to do sensitivity analysis.
- It can model systems over difficult-to-measure time frames.

CONS:

Monte Carlo Simulations

Definition: A simulation that incorporates an element of randomness is called a **Monte Carlo** simulation.

PROS:

- It is a relatively easy method to approximate complex systems.
- Once built, it allows for tinkering—easy to do sensitivity analysis.
- It can model systems over difficult-to-measure time frames.

CONS:

- You have to build it. (Expensive to develop!)
- Requires computing power and time.

Monte Carlo Simulations

Definition: A simulation that incorporates an element of randomness is called a **Monte Carlo** simulation.

PROS:

- It is a relatively easy method to approximate complex systems.
- Once built, it allows for tinkering—easy to do sensitivity analysis.
- It can model systems over difficult-to-measure time frames.

CONS:

- You have to build it. (Expensive to develop!)
- Requires computing power and time.
- Makes you over-confident in the results.

Monte Carlo Simulations

Definition: A simulation that incorporates an element of randomness is called a **Monte Carlo** simulation.

PROS:

- It is a relatively easy method to approximate complex systems.
- Once built, it allows for tinkering—easy to do sensitivity analysis.
- It can model systems over difficult-to-measure time frames.

CONS:

- You have to build it. (Expensive to develop!)
- Requires computing power and time.
- Makes you over-confident in the results.
- Dealing with probability, so results will always be of the form: "With 95% probability, the wait time will be less than 2 minutes."

Simulating flipping a coin

Example. Get a computer to simulate flipping a fair coin 20 times.

To simulate a random event, use one of the *Mathematica* commands:

- `RandomInteger` gives a pseudo-random *integer*.

- `RandomReal` gives a pseudo-random *real number*.

Simulating flipping a coin

Example. Get a computer to simulate flipping a fair coin 20 times.

To simulate a random event, use one of the *Mathematica* commands:

- `RandomInteger` gives a pseudo-random *integer*.
  - `RandomInteger[]` (no input) gives either 0 or 1.
  - `RandomInteger[5]` gives an integer from 0 to 5.
  - `RandomInteger[{1, 10}]` gives an integer from 1 to 10.
- `RandomReal` gives a pseudo-random *real number*.

Simulating flipping a coin

Example. Get a computer to simulate flipping a fair coin 20 times.

To simulate a random event, use one of the *Mathematica* commands:

- `RandomInteger` gives a pseudo-random *integer*.
  - `RandomInteger[]` (no input) gives either 0 or 1.
  - `RandomInteger[5]` gives an integer from 0 to 5.
  - `RandomInteger[{1, 10}]` gives an integer from 1 to 10.
  - `RandomInteger[{1, 10},20]` gives a list of 20 such integers.
- `RandomReal` gives a pseudo-random *real number*.

The first input gives the range; a second input tells how many to make.

Simulating flipping a coin

Example. Get a computer to simulate flipping a fair coin 20 times.

To simulate a random event, use one of the *Mathematica* commands:

- `RandomInteger` gives a pseudo-random *integer*.
  - `RandomInteger[]` (no input) gives either 0 or 1.
  - `RandomInteger[5]` gives an integer from 0 to 5.
  - `RandomInteger[{1, 10}]` gives an integer from 1 to 10.
  - `RandomInteger[{1, 10},20]` gives a list of 20 such integers.
- `RandomReal` gives a pseudo-random *real number*.
  - `RandomReal[]` (no input) gives a real number between 0 or 1.
  - `RandomReal[{0.1, 0.2}]` gives a real number from 0.1 to 0.2.

The first input gives the range; a second input tells how many to make.

Simulating flipping a coin

Example. Get a computer to simulate flipping a fair coin 20 times.

To simulate a random event, use one of the *Mathematica* commands:

- `RandomInteger` gives a pseudo-random *integer*.
  - `RandomInteger[]` (no input) gives either 0 or 1.
  - `RandomInteger[5]` gives an integer from 0 to 5.
  - `RandomInteger[{1, 10}]` gives an integer from 1 to 10.
  - `RandomInteger[{1, 10},20]` gives a list of 20 such integers.
- `RandomReal` gives a pseudo-random *real number*.
  - `RandomReal[]` (no input) gives a real number between 0 or 1.
  - `RandomReal[{0.1, 0.2}]` gives a real number from 0.1 to 0.2.
  - `RandomReal[{0.1, 0.2},15]` gives a list of 15 such numbers.

The first input gives the range; a second input tells how many to make.

Simulating flipping a coin

Example. Get a computer to simulate flipping a fair coin 20 times.

To simulate a random event, use one of the *Mathematica* commands:

- `RandomInteger` gives a pseudo-random *integer*.
  - `RandomInteger[]` (no input) gives either 0 or 1.
  - `RandomInteger[5]` gives an integer from 0 to 5.
  - `RandomInteger[{1, 10}]` gives an integer from 1 to 10.
  - `RandomInteger[{1, 10},20]` gives a list of 20 such integers.
- `RandomReal` gives a pseudo-random *real number*.
  - `RandomReal[]` (no input) gives a real number between 0 or 1.
  - `RandomReal[{0.1, 0.2}]` gives a real number from 0.1 to 0.2.
  - `RandomReal[{0.1, 0.2},15]` gives a list of 15 such numbers.

The first input gives the range; a second input tells how many to make.

The numbers produced by a random number generator are never truly random because they are produced by an algorithm on a deterministic machine.

Simulating flipping a coin

Example. Get a computer to simulate flipping a fair coin 20 times.

Let's use the convention: 1=‘Head’ and 0=‘Tail’.

Simulating flipping a coin

Example. Get a computer to simulate flipping a fair coin 20 times.

Let's use the convention: 1='Head' and 0='Tail'. Then evaluating `RandomInteger[1,20]` will generate a list of 20 coin tosses.

Simulating flipping a coin

Example. Get a computer to simulate flipping a fair coin 20 times.

Let's use the convention: 1=‘Head’ and 0=‘Tail’. Then evaluating `RandomInteger[1,20]` will generate a list of 20 coin tosses.

In[1]: `CoinFlips = RandomInteger[1,20]`
Out[1]: $\{1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1\}$

Simulating flipping a coin

Example. Get a computer to simulate flipping a fair coin 20 times.

Let's use the convention: 1='Head' and 0='Tail'. Then evaluating `RandomInteger[1,20]` will generate a list of 20 coin tosses.

In[1]: `CoinFlips = RandomInteger[1,20]`
Out[1]: $\{1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1\}$

The sum of this list is the total number of heads tossed.

In[2]: `Total[CoinFlips]`
Out[2]: 13

Simulating flipping a coin

Example. Get a computer to simulate flipping a fair coin 20 times.

Let's use the convention: 1='Head' and 0='Tail'. Then evaluating `RandomInteger[1,20]` will generate a list of 20 coin tosses.

In[1]: `CoinFlips = RandomInteger[1,20]`
Out[1]: {1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1}

The sum of this list is the total number of heads tossed.

In[2]: `Total[CoinFlips]`
Out[2]: 13

Running the commands again will simulate another trial of 20 flips.