EE511

10 Jan 2017

Tools:

git: learn it.

you can "version" your code, make changes, recall past versions and settings (called: "history").

File-by-file and line-by-line changes to track "diffs".

An (almost) perfect audit of your work.

- Not required to learn but 3% grade (overall) extra credit if all your projects (with history) available in git. Note: cannot do it all at the end, must have evidence throughout course.

- I will post a link to tutorial but basic idea is:
  "Add" file, commit, edit, add, commit.

- Distributions for Windows and Linux (and Mac).

latex: not required but fast and efficient layout (typesetting) tool. You can automatically generate document w/ latest images, etc. You can also break your reports into bitesize pieces.

Benefit: All text based so you can version (and search) your project reports right along with your code.

PhD: you will eventually need.

Masters: good skill to have.

"The not so short introduction to latex" (Ch: 1,2,3)

10 Jan 2017

EE511

Languages:

Matlab: free for students (software.usc.edu).

Most class examples will use Matlab because it allows rapid prototyping.

Both TAs are strong with MatLab.

Java, C/C++: some of my favorites but I don't recommend for this class. Very useful for the same problems during research due to execution speed.

Python: Powerful general purpose computing language. A good skill to develop. and relatively easy to learn.

I recomend python v3.6 unless you have specific compatibility needs (its complicated).

Recomended packages: scipy, numpy, matplotlib

"scientific" computing   array processing   plotting library.

R: powerful statistics tool. Very practical resume builder.

The "lingua franca" in many big data and information processing fields.

For beginners, I suggest: RStudio. It provides a nice command window interface and simplifies dealing with graphics and packages.

Others: Excel, Perl, ... whatever you like.

EE511 10 Jan 2017

Coding Style:

#1 Rule: make it readable. It should be clear enough even to someone not familiar with the syntax for a particular language.

Hints:

1. Comment code, document reasoning, describe assumptions, numeric considerations (valid ranges, etc.).

   ** required for EE511 **

2. Make code readable
   - comment should be helpful, not superflouous.
   - nice indenting
   - descriptive variable names

---

see syllabus: not necessarily marked down if not but if the grader cannot follow/parse your code they will deduct.

TAs can help with general methods; but they might not be able to diagnose specific failures, compilation issues, etc.
↓
Post to Piazza.

<u>Moral</u>: "Know your gear" before you start and

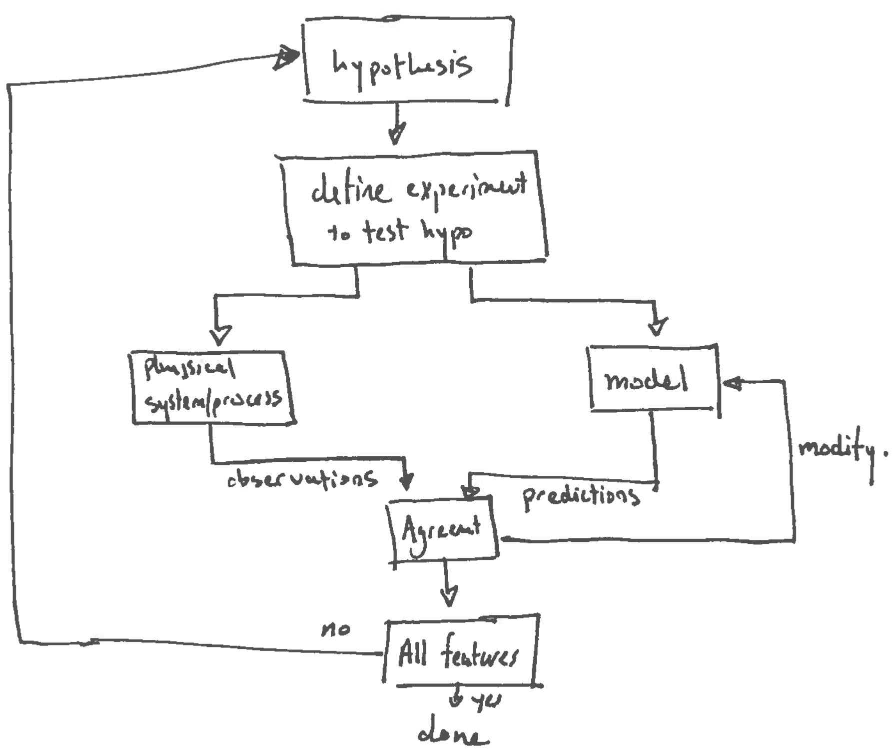
"Know what you expect" before every experiment.

To encourage new efforts:
receive +2% to overall grade for submitting a different language than your previous projects (max: 4%).

So submitting project 1 with Matlab, Project 2 with C/C++, and project 3 with python gives +4% class grade.

<u>Some theory</u>:

Mathematical models: explain observed behavior using simple and understandable rules with measurable properties.



EE511

Each of these steps require modeller input and each decision requires an expert opinion.

For EE511: you are the expert.

Iterative process continues until model is sufficient (or run out of funding)
(good enough)

Types of models:

1. Deterministic model.
   exact inputs produce same output.

Consider electrical circuit: $V = i \cdot R$.. In this model knowing i and R. gives the "exact" (theoretical) voltage.

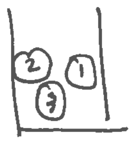
Note: repeating the experiment yields the exact same model prediction.

2. Probability models
   exact inputs produce different outputs (usually).

Defn: A random experiment is an experiment in which the outcome varies in an unpredictable fashion when the experiment is repeated under the same condition.

* You cannot generally use a deterministic model for random experiments.

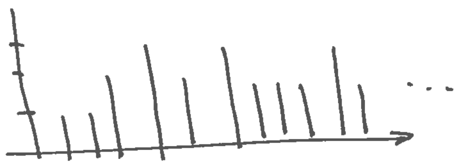
Ex: 3-ball urn experiment.



1. randomize and select.
2. record number and return

experiment outcome: number from set $S = \{1,2,3\}$.

Repeat the experiment many times: 1, 1, 2, 3, 2, 3, 2 ...

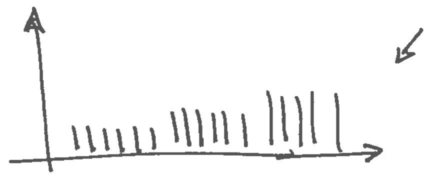


time

summary method: plot the raw samples (just the data).

Q: Does order really matter?

probably, consider plotting sorted:



In this class we take advantage of Statistical Regularity

regularity is a means to quantify predictability.

Many models rely on the fact that averages of long sequences of experiments (each called a "trial") will yield approximately the same value.

Ex: Let $N_1(n)$, $N_2(n)$, and $N_3(n)$ be the number of times the experimental outcome is 1, 2, and 3 respectively.

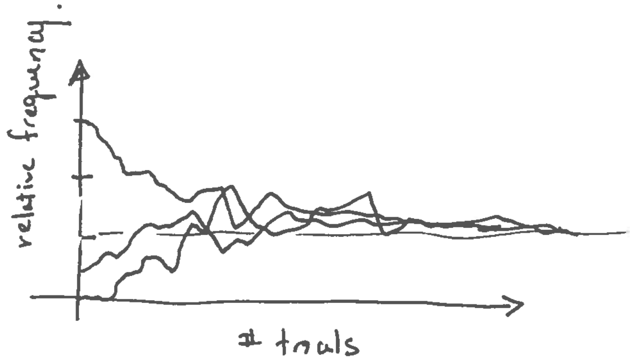
Define: the relative frequency $f_k(n) = \frac{N_k(n)}{n}$

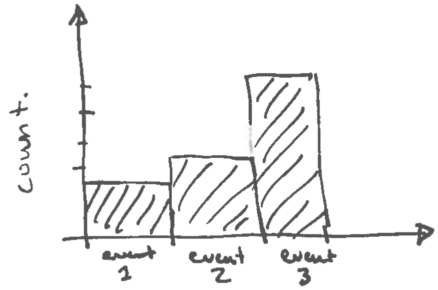
Statistical regularity means that $f_k(n)$ varies less-and-less as n gets large

Specifically: $\lim_{n \to \infty} f_k(n) = p_k$

constant p_k is called the probability of outcome k.

EE511 10 Jan 2017





summary method: plot relative frequency (or count) vs # trials.

summary method: plot counts for each outcome (or range of outcomes). **HISTOGRAM**

Statistical regularity means we could re-run the trials and each may start with very different relative frequencies but they will eventually all converge to the same point.

By simulation result, $P_K \rightarrow 1/3$ for each outcome (agrees with intuition).

Consider: modify experiment and add additional #1 to urn.

then $P_1 = 1/2$, $P_2 = 1/4$, $P_3 = 1/4$.

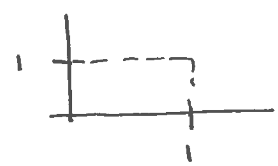
EE511 considerations: Project #1.

Just flipping a coin ... as "easy" as it gets (0 or 1).

Bernoulli trial
⇓
random experiment with two possible outcomes 0,1 | H,T. | etc.

Hint: Use the rand( ) function in your preferred langauge.

Assuming $X \sim U[0,1]$.



then threshold:

rand( ) ≥ 0.5 ← Head

rand( ) < 0.5 ← Tail

Q: Does it matter if you switch? (no).

10 Jan 2017

EE511

Coding hints:

1. Put your experiment in a function (or lang. equivalent)
   - you can easily invoke with different parameters without having to disrupt your algorithm code.

Naming:

Bad:	experiment1 ( )	// return 0 or 1.
Better:	flip-coin ( )	
Better:	flip-coin (0.5)	// "H" probability.
Best:	flip-coin-n (0.5, 50)	// multiple flips. (may be faster in some languages).

2. Start building a "library" of useful functions for re-use.

e.g. flip-fair-coin ( )
   return flip-coin (0.5).

→ within reason, note the "factorial problem" in the extreme.