X-Informatics
Physics Use Case Part IV
Looking for Higgs Particle

Random Variables, Distributions, Central Limit Theorem

July 7 2013

Geoffrey Fox
gcf@indiana.edu

http://www.infomall.org/X-InformaticsSpring2013/index.html

Associate Dean for Research, School of Informatics and Computing
Indiana University Bloomington

2013

Big Data Ecosystem in One Sentence

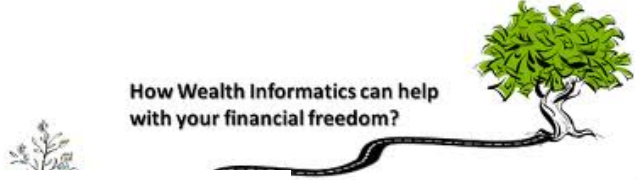
Use Clouds running Data Analytics Collaboratively processing Big Data to solve problems in X-Informatics (or e-X)

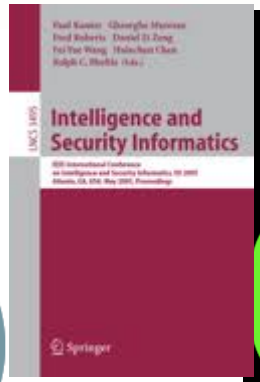
X = Astronomy, Biology, Biomedicine, Business, Chemistry, Climate, Crisis, Earth Science, Energy, Environment, Finance, Health, Intelligence, Lifestyle, Marketing, Medicine, Pathology, Policy, Radar, Security, Sensor, Social, Sustainability, Wealth and Wellness with more fields (physics) defined implicitly

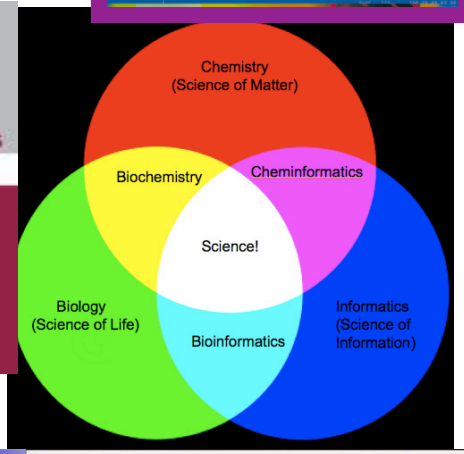
Spans Industry and Science (research)

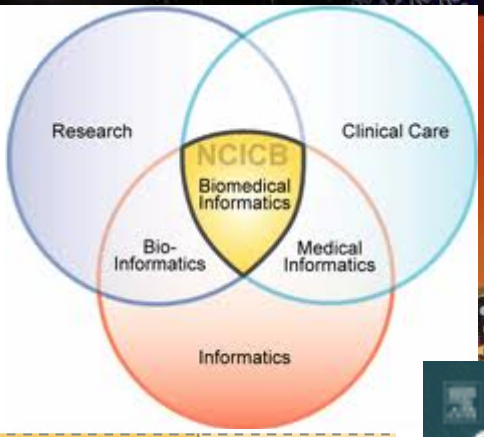
Education: Data Science see recent New York Times articles

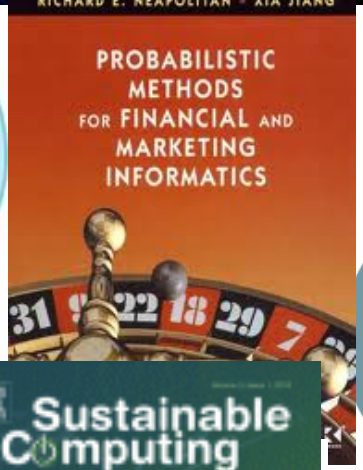
http://datascience101.wordpress.com/2013/04/13/new-york-times-data-science-articles/

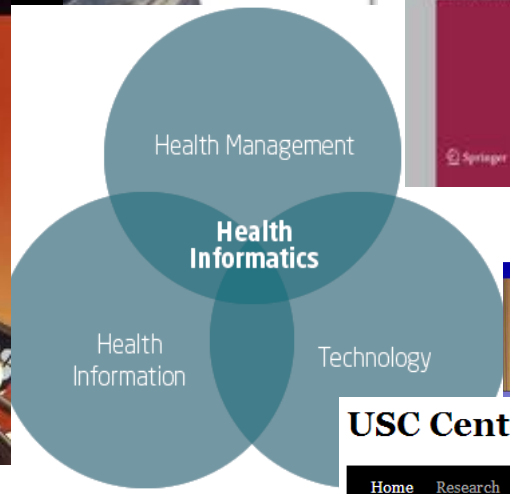


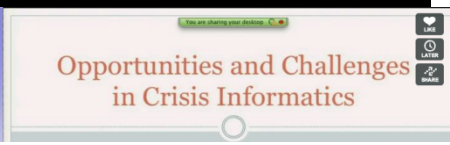


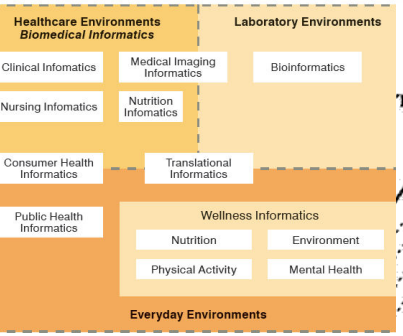


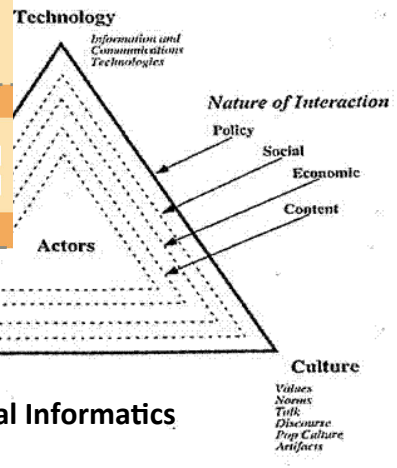








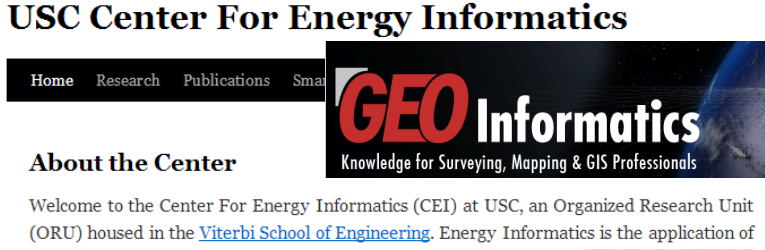


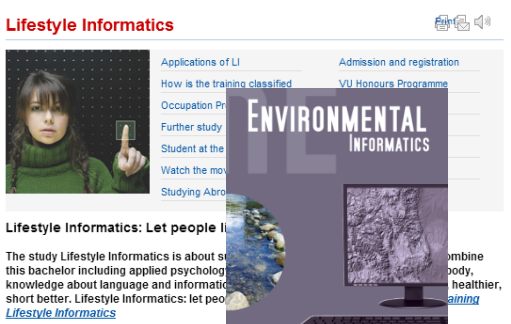


Noelia Penelope Greer (Ed.)

Business Informatics

Information technology, Management,





Random Numbers

Generators and Seeds

Random Numbers I

- Would be really random if we observed nuclei decaying but in fact nearly all random numbers are “pseudo random numbers” using “principle” that complicated messy thing end up with random results
  - In particular low order digits of integer arithmetic are essentially random
- http://en.wikipedia.org/wiki/Pseudorandom_number_generator
- http://en.wikipedia.org/wiki/Mersenne_twister
- http://en.wikipedia.org/wiki/Mersenne_prime
- Python uses Mersenne Twister method built around Mersenne primes
- Pseudorandom numbers are deterministic; they are generated in order and for a given starting point – seed – the numbers are completely determined
- Some computers always use same seed and so always return same sequence by default
- Python takes seed from time of day (pretty random) and generates a different sequence each call

Random Number Generators

- Simplest pseudo random number generator

$$X_{n+1} \equiv (aX_n + c) \qquad (\mathrm{mod}\ m)$$

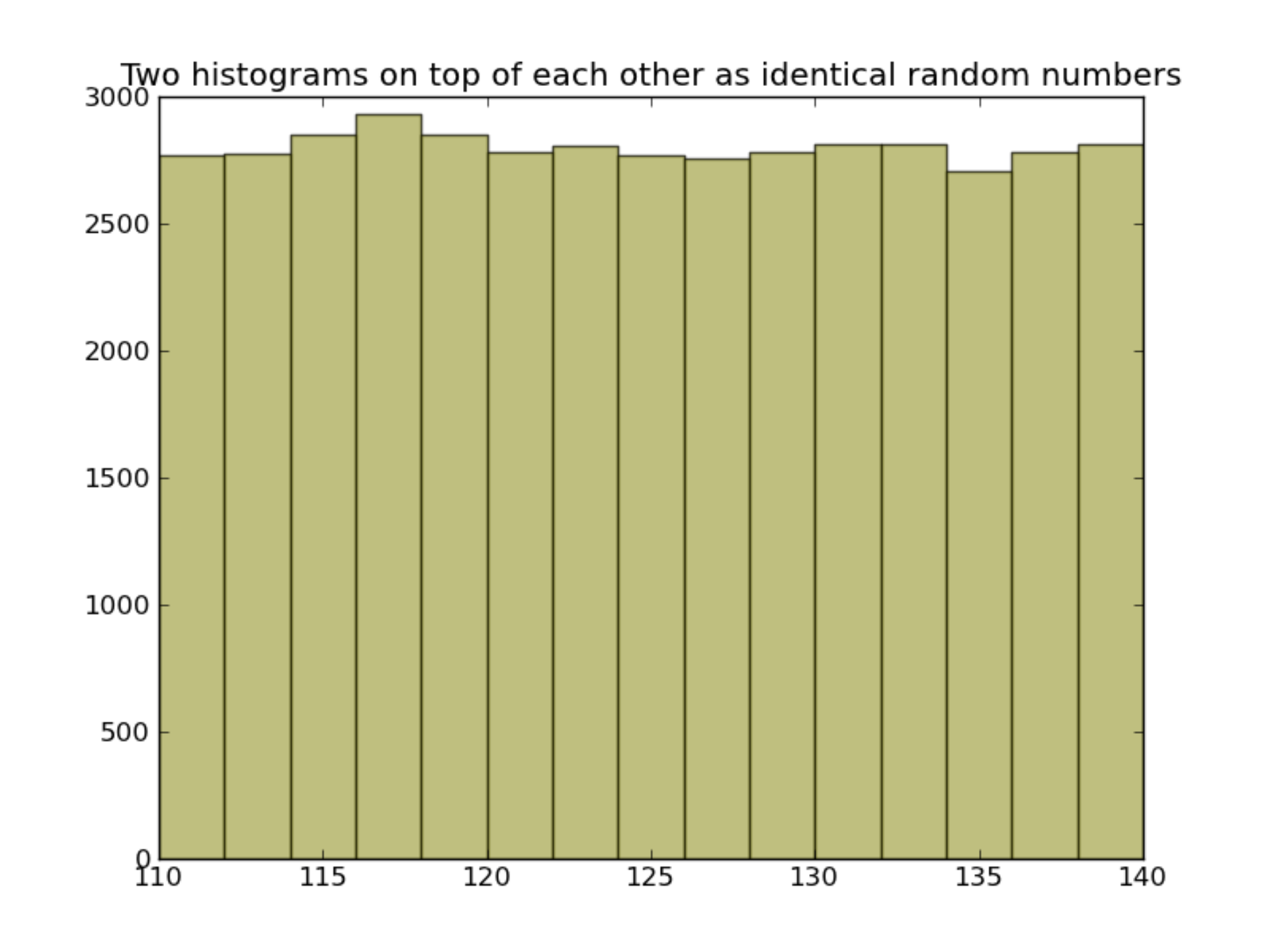
- m = 2^{32} is nice as then finding modulus is just masking the low order 32 bits of X_{n+1} in its binary form
- Numerical Recipes gives
- a = 1664525
- C = 1013904223

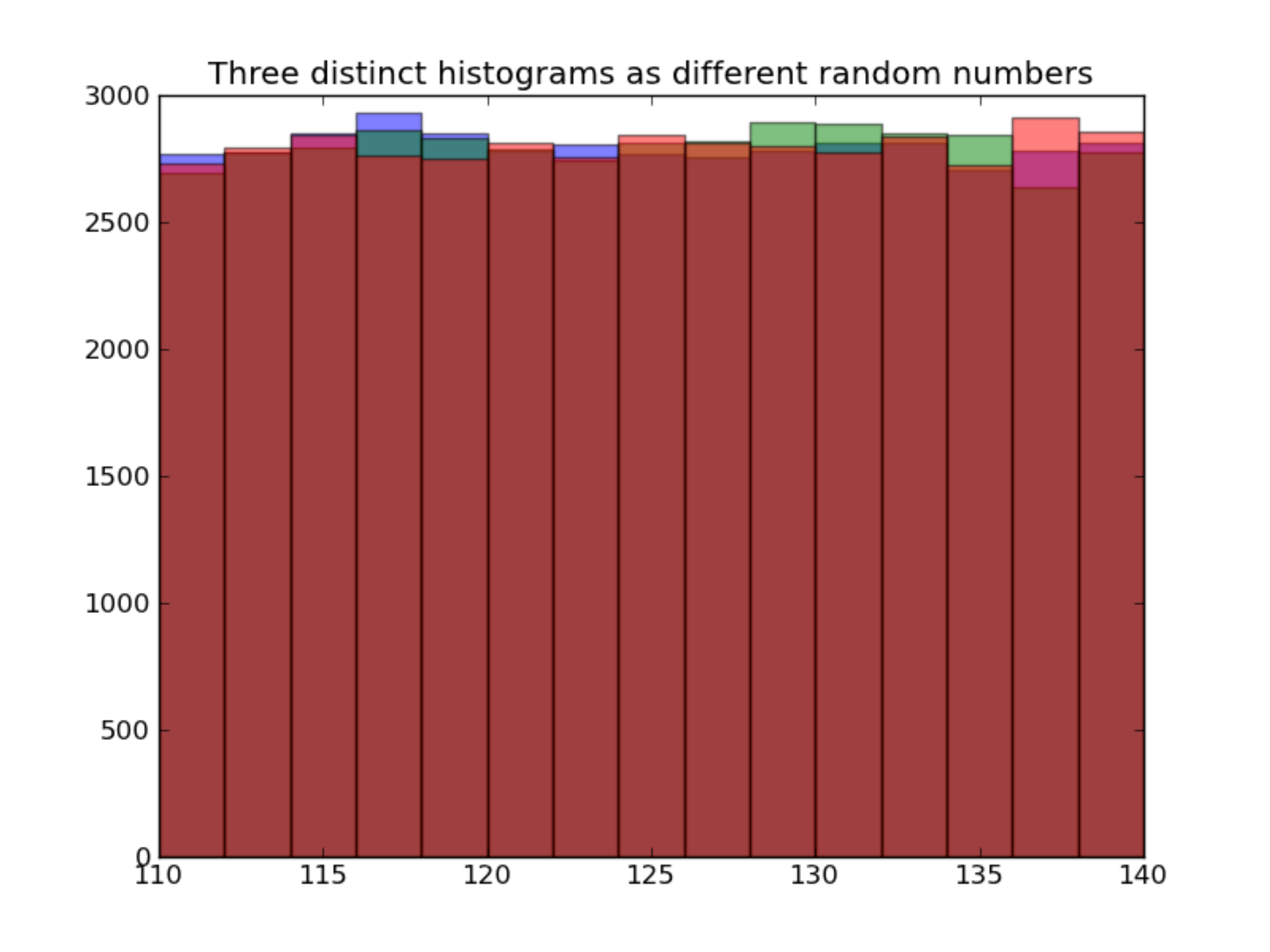
Random Numbers II

- Sometimes its good to get random numbers but always have the same e.g. if you are debugging something
- Use random.seed(seed=some integer) to guarantee same start
- random.seed(seed=1234567)
- figure("On Top of Each Other")
- Base = 110 + 30* np.random.rand(42000)
- plt.hist(Base, bins=15, range =(110,140), alpha = 0.5, color="blue")
- #
- random.seed(seed=1234567)
- Base2 = 110 + 30* np.random.rand(42000)
- plt.hist(Base2, bins=15, range =(110,140), alpha = 0.5, color="yellow")
- plt.title("Two histograms on top of each other as identical random numbers")

More on Seeds

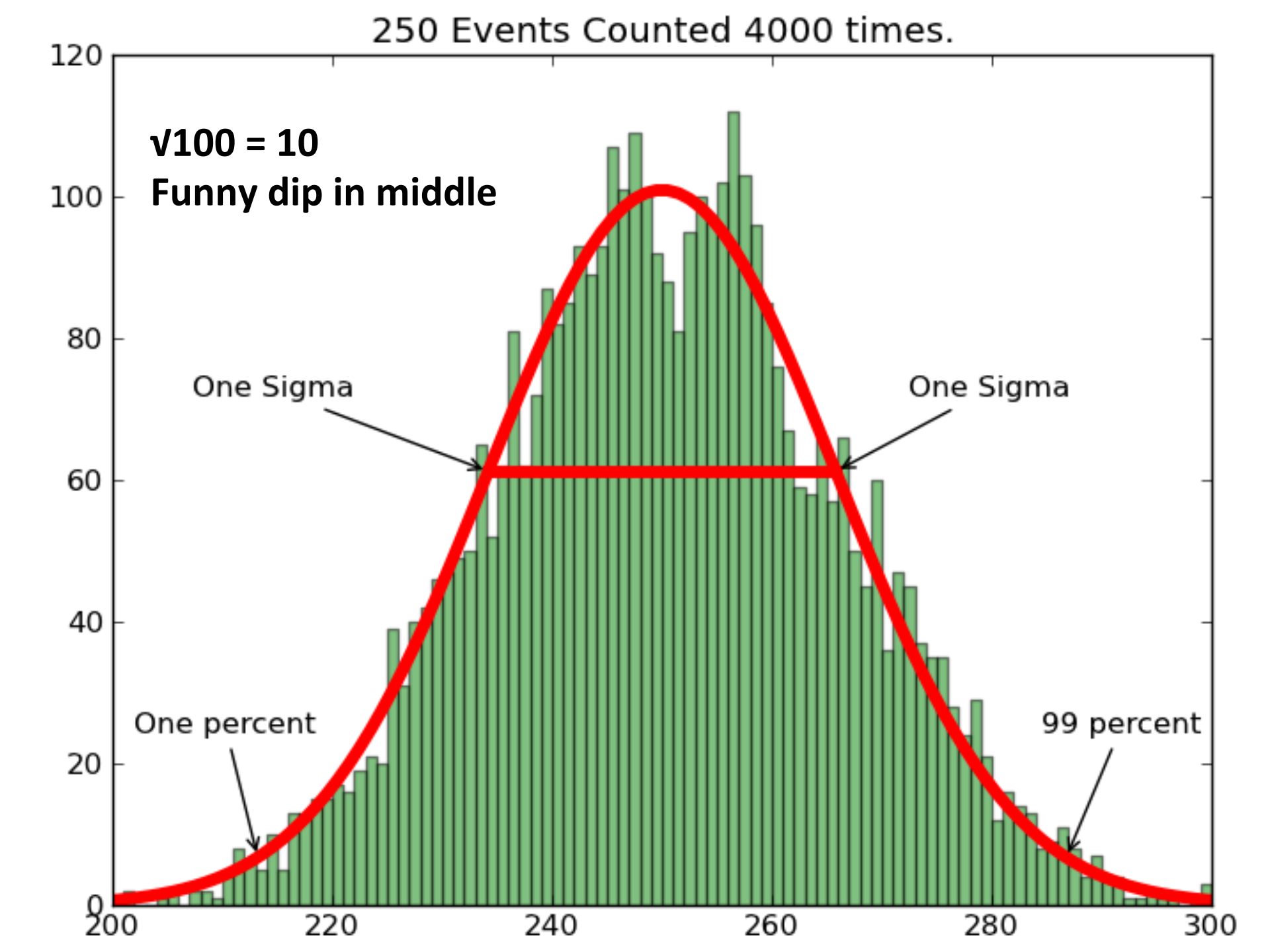
- figure("Different")
- #
- random.seed(seed=1234567)
- Base4 = 110 + 30* np.random.rand(42000)
- plt.hist(Base4, bins=15, range =(110,140), alpha = 0.5, color="blue")
- Base1 = 110 + 30* np.random.rand(42000)
- # Note Base1 starts where Base4 ends so is a different set of random numbers
- plt.hist(Base1, bins=15, range =(110,140), alpha = 0.5, color="green")
- #
- random.seed(seed=7654321)
- Base3 = 110 + 30* np.random.rand(42000)
- plt.hist(Base3, bins=15, range =(110,140), alpha = 0.5, color="red")

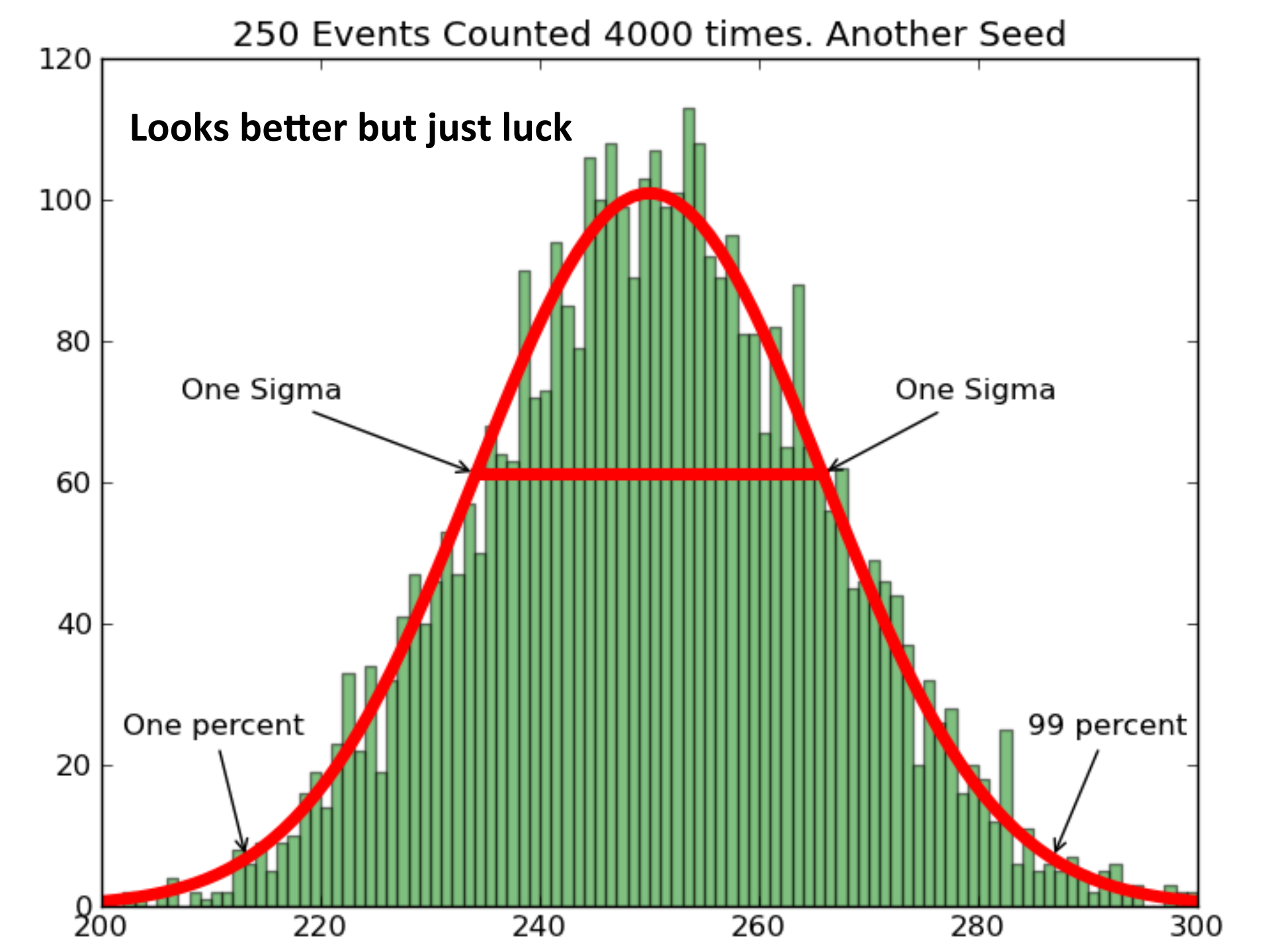

Two histograms on top of each other as identical random numbers
3000
2500
2000
1500
1000
500
0
110
115
120
125
130
135
140


Three distinct histograms as different random numbers
3000
2500
2000
1500
1000
500
0
110
115
120
125
130
135
140

Look at Numbers

- random.seed(seed=1234567)
- Base = 110 + 30* np.random.rand(42000)
- array([117.11087505, 110.22945122, 110.59490925, ..., 117.34085492,
- 138.72156377, 117.25522386])
- random.seed(seed=1234567)
- Base2 = 110 + 30* np.random.rand(42000)
- array([117.11087505, 110.22945122, 110.59490925, ..., 117.34085492,
- 138.72156377, 117.25522386])
- Base1 = 110 + 30* np.random.rand(42000)
- array([133.51704792, 132.87435972, 128.95204036, ..., 115.25271937,
- 130.21388378, 112.55649254])
- random.seed(seed=7654321)
- Base3 = 110 + 30* np.random.rand(42000)
- array([111.05068005, 112.63158568, 124.30148179, ..., 120.35946227,
- 112.44901602, 133.75223691])


250 Events Counted 4000 times.
√100 = 10
Funny dip in middle
One Sigma
One Sigma
One percent
99 percent
120
100
80
60
40
20
0
200
220
240
260
280
300


250 Events Counted 4000 times. Another Seed
Looks better but just luck
One Sigma
One Sigma
One percent
99 percent
120
100
80
60
40
20
0
200
220
240
260
280
300

Random Numbers

Binomial Distribution

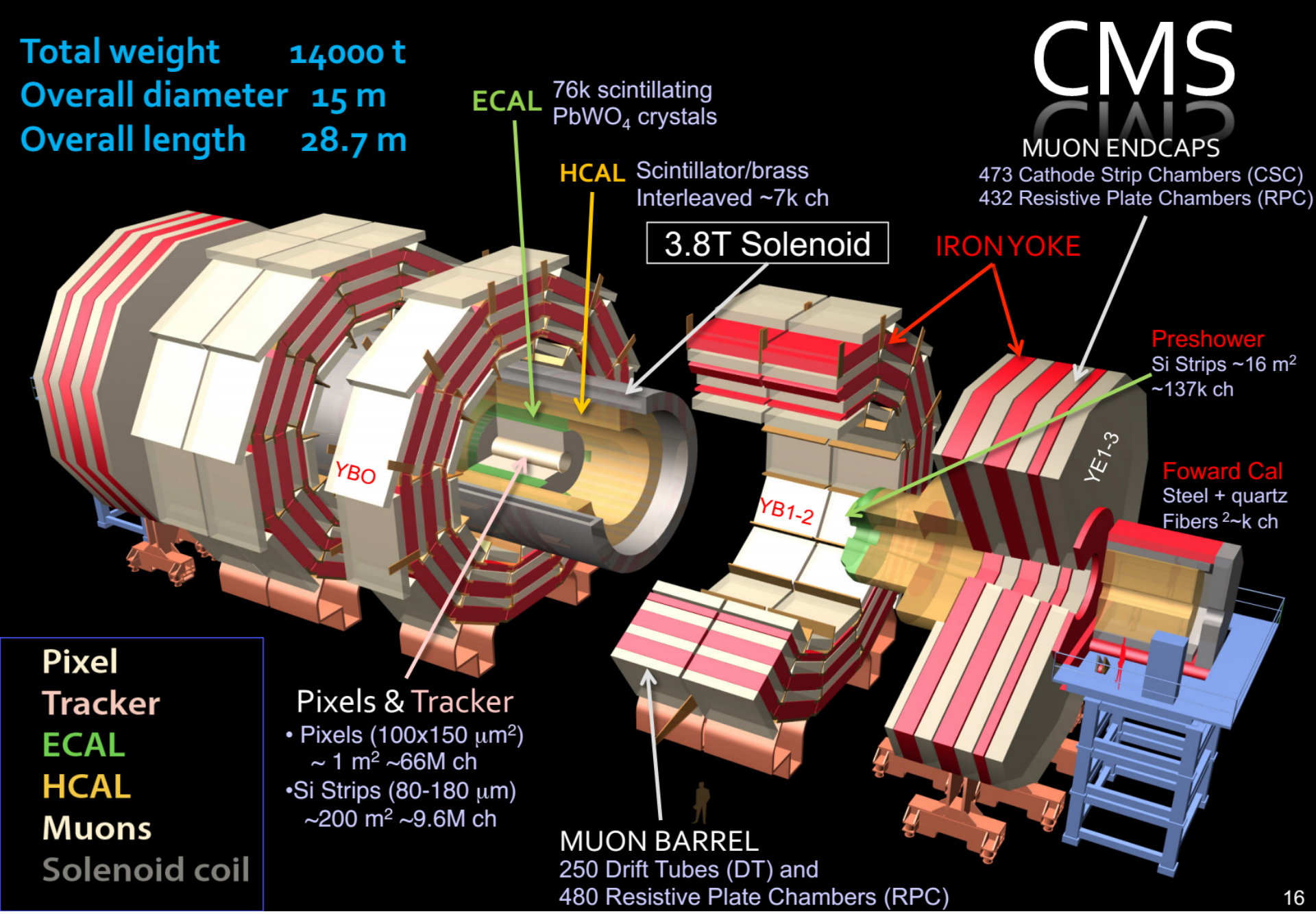
Counting and Binomial Distribution I

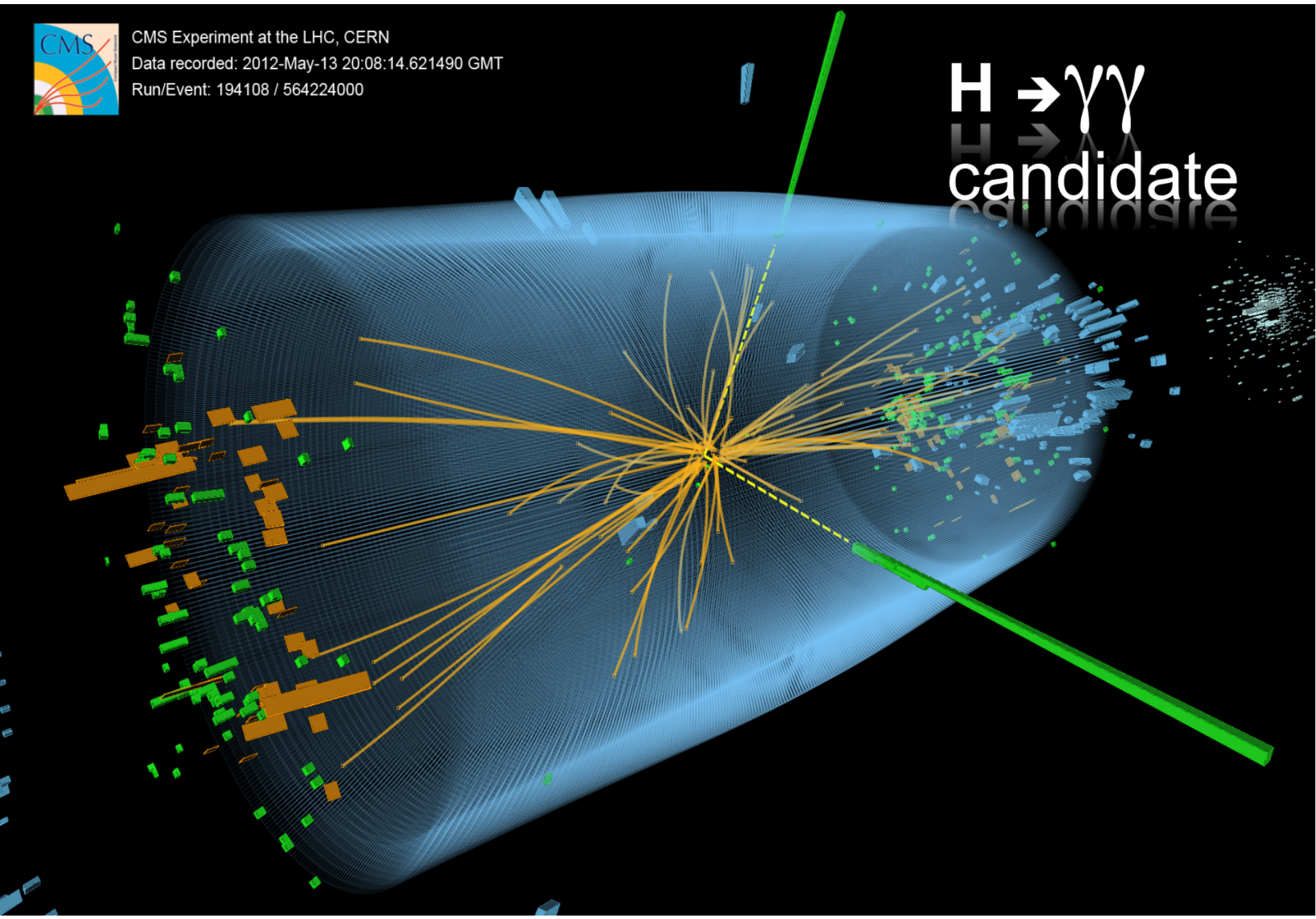
- Suppose random variable X = 1 (event in a particular bin of histogram) and X=0 (event not in bin)
- Let μ be probability that a particular event lies in a bin
  - This is coin tossing problem with μ probability of heads
- If a random variable takes just two values with probability μ (for 1) and 1-μ (for 0), then
- **Average of X** = $\{0.\ (1-\mu) + 1.\ \mu\}/\{(1-\mu) + \mu\} = \mu$
- **Average of $(X-\mu)^2$** = $\{(-\mu)^2.\ (1-\mu) + (1-\mu)^2\ .\ \mu\}$

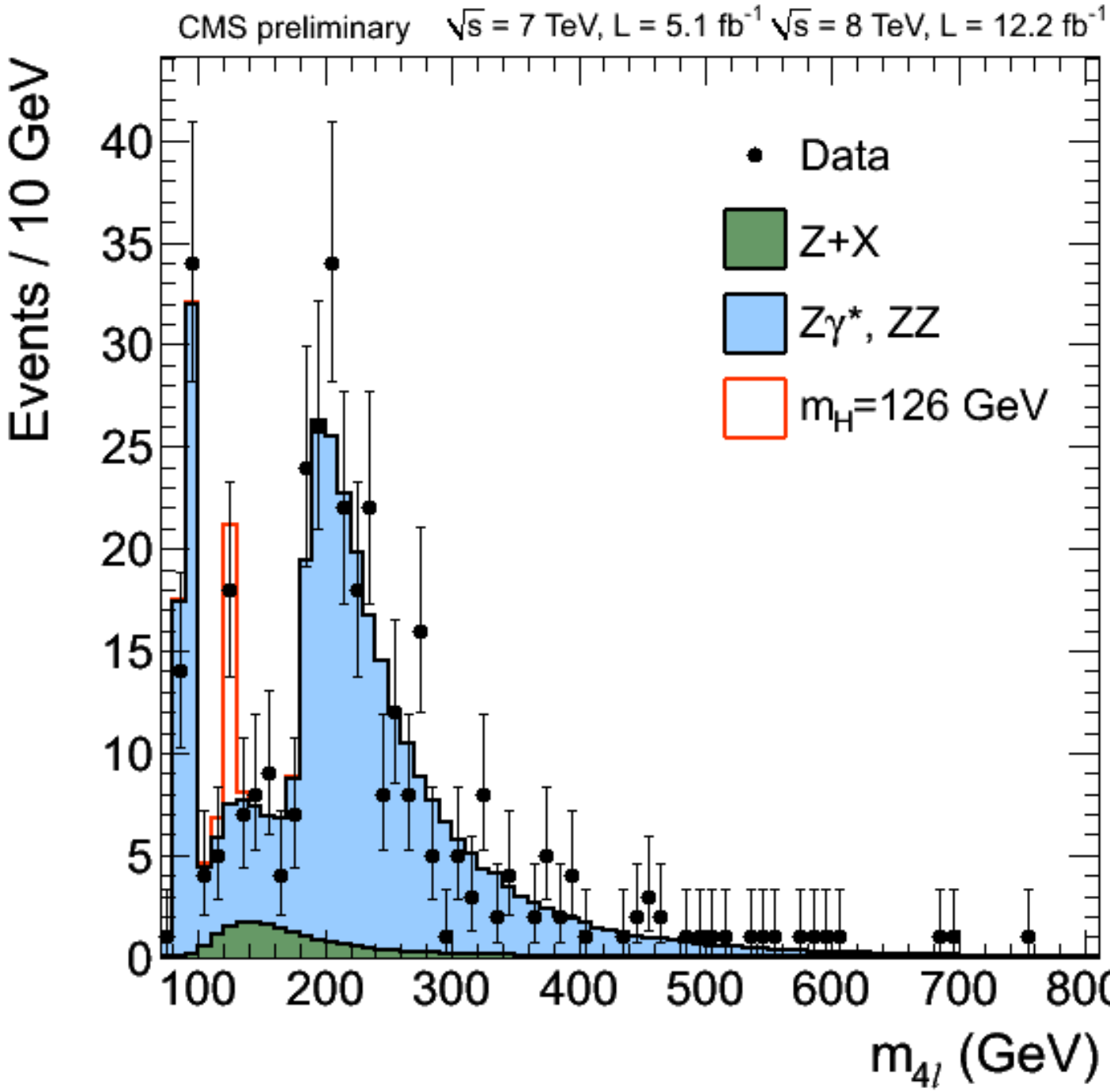
$= \mathbf{(1-\mu)\ \mu}$

Counting and Binomial Distribution II

- Now in our application μ is tiny
- So Average of X = Average of $(X-\mu)^2 = \mu$
- i.e. standard deviation of X is square root of mean
- If we have a sum of random variables with a binomial distribution
- $O = \sum_{i=1}^{N} X_i$
- Then O has **mean Nμ** and **standard deviation = $\sqrt{N}$ $\sqrt{\mu}$ = $\sqrt{\text{mean}}$**


Total weight 14000 t
Overall diameter 15 m
Overall length 28.7 m
CMS
ECAL 76k scintillating PbWO4 crystals
HCAL Scintillator/brass Interleaved ~7k ch
3.8T Solenoid
IRON YOKE
MUON ENDCAPS
473 Cathode Strip Chambers (CSC)
432 Resistive Plate Chambers (RPC)
Preshower
Si Strips ~16 m2
~137k ch
Foward Cal
Steel + quartz
Fibers 2~k ch
YBO
YB1-2
YE1-3
Pixel
Tracker
ECAL
HCAL
Muons
Solenoid coil
Pixels & Tracker
• Pixels (100x150 μm2)
~ 1 m2 ~66M ch
•Si Strips (80-180 μm)
~200 m2 ~9.6M ch
MUON BARREL
250 Drift Tubes (DT) and
480 Resistive Plate Chambers (RPC)


CMS Experiment at the LHC, CERN
Data recorded: 2012-May-13 20:08:14.621490 GMT
Run/Event: 194108 / 564224000
H → γγ
candidate

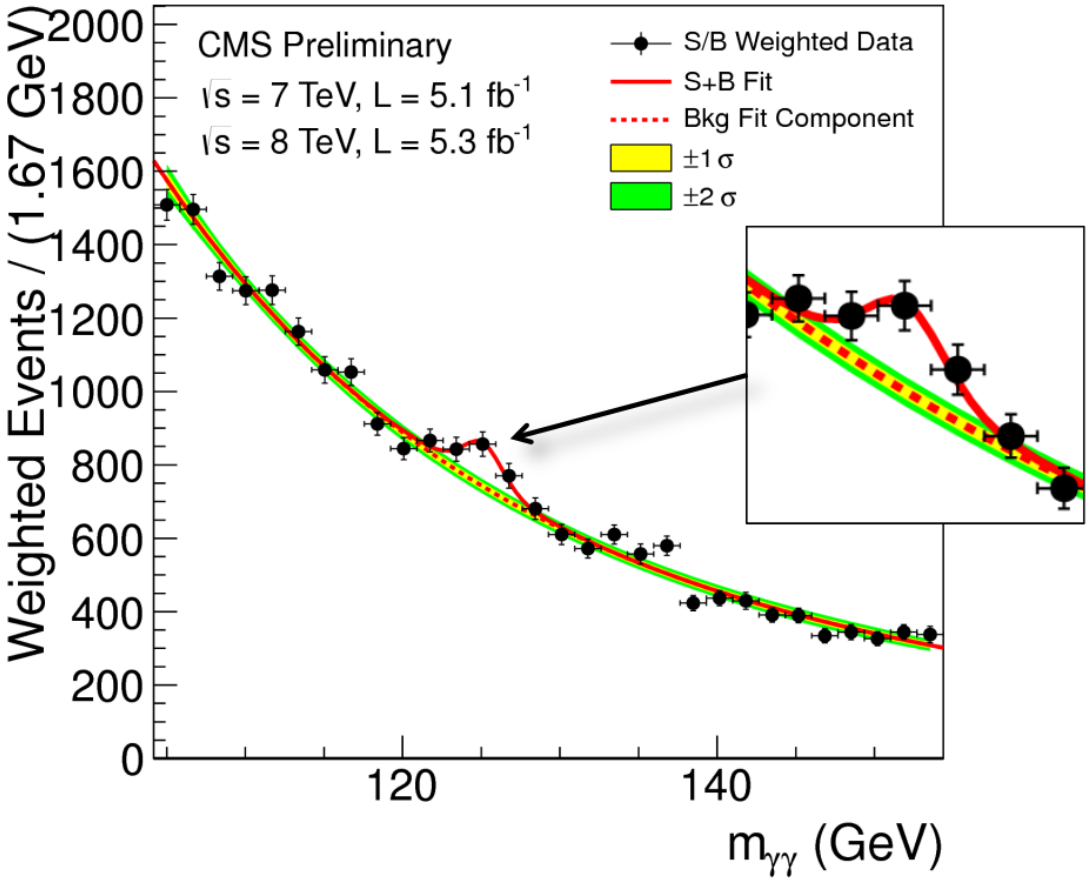


Here is a histogram with 140 bins. We have a random variable for each bin.

There is an excess of events in bin from 125 to 130 GeV

S/B Weighted Mass Distribution

- Sum of mass distributions for each event class, weighted by S/B
  - B is integral of background model over a constant signal fraction interval



Comments

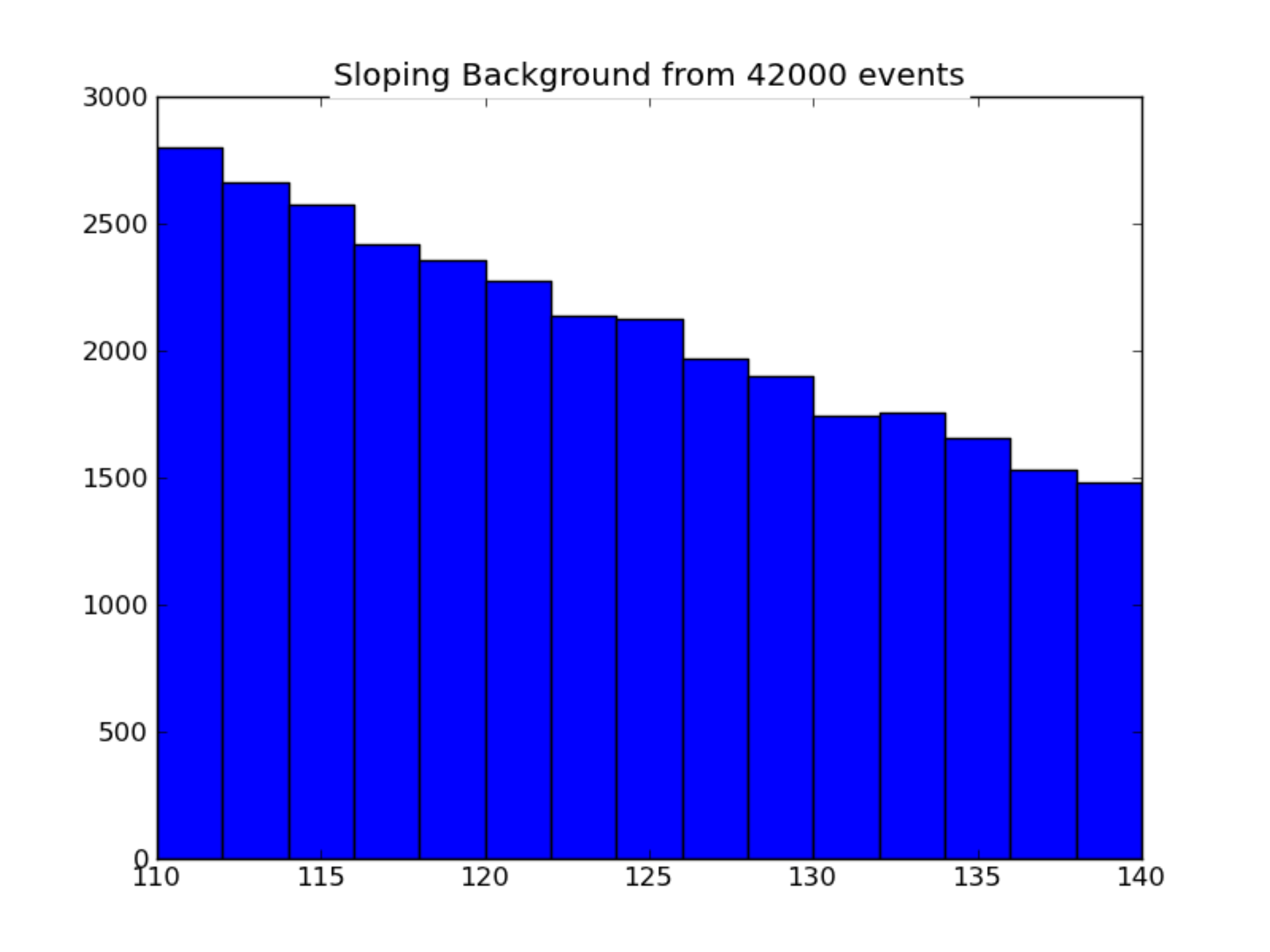
- Note some measurements have a small signal but a small background
- Others have a larger signal but also a larger background
- If you have signal N_S and background N_B, then statistical error is $\sqrt{(N_S + N_B)}$ and one needs
- $\sqrt{(N_S + N_B)}$ much smaller than N_S which is harder than $\sqrt{N_S}$ much smaller than N_S
- Typically one quotes “systematic errors” as well. These reflect model-based uncertainties in analysis and will not decrease like sqrt of total event sample

Random Numbers

Accept-Reject

Generating Sloping distribution

- In Homework, we used a flat background for Higgs distribution from 110 to 140 GeV.
- testrand = np.random.rand(42000)
- Base = 110 + 30* np.random.rand(42000)
- index = (1.0 - 0.5* (Base-110)/30) > testrand
- Sloping = Base[index]
- Index is true with a probability that starts at 1 at mass of 110 and becomes 0.5 at 140.
- This makes Sloping a set of random events with a linear decrease from 110 to 140
- This advanced concept is called accept-reject method and is not important for general principles


Sloping Background from 42000 events
3000
2500
2000
1500
1000
500
0
110
115
120
125
130
135
140

Accept-Reject Method

- If you want to generate random numbers in range x_{min} to x_{max} with probability f(x).
- Then generate them uniformly between x_{min} and x_{max}
- Then accept them with probability $f(x)/f_{max}$
- Where f_{max} is maximum value of f(x) in range x_{min} to x_{max}

Random Numbers

Monte Carlo Method

“Monte-Carlo Events”

- The events are very complicated with ~100 particles produced and the apparatus is also complex with multiple detection devices measuring energy, momentum and particle type.
- To understand how an event “should look” and estimate detection efficiencies, Monte Carlo events are produced
- These are “random” events produced by models that build in expected physics
  - One both generates the “fundamental collision” and tracks particles through apparatus
- Although they are motivated by physics, they have many adjustable parameters that are fitted to other data to describe aspects of “theory” that are not predicted “from first principles”
- One analyses Monte Carlo data with EXACTLY the same process used by real data
- Often amount of Monte Carlo data > that of real data

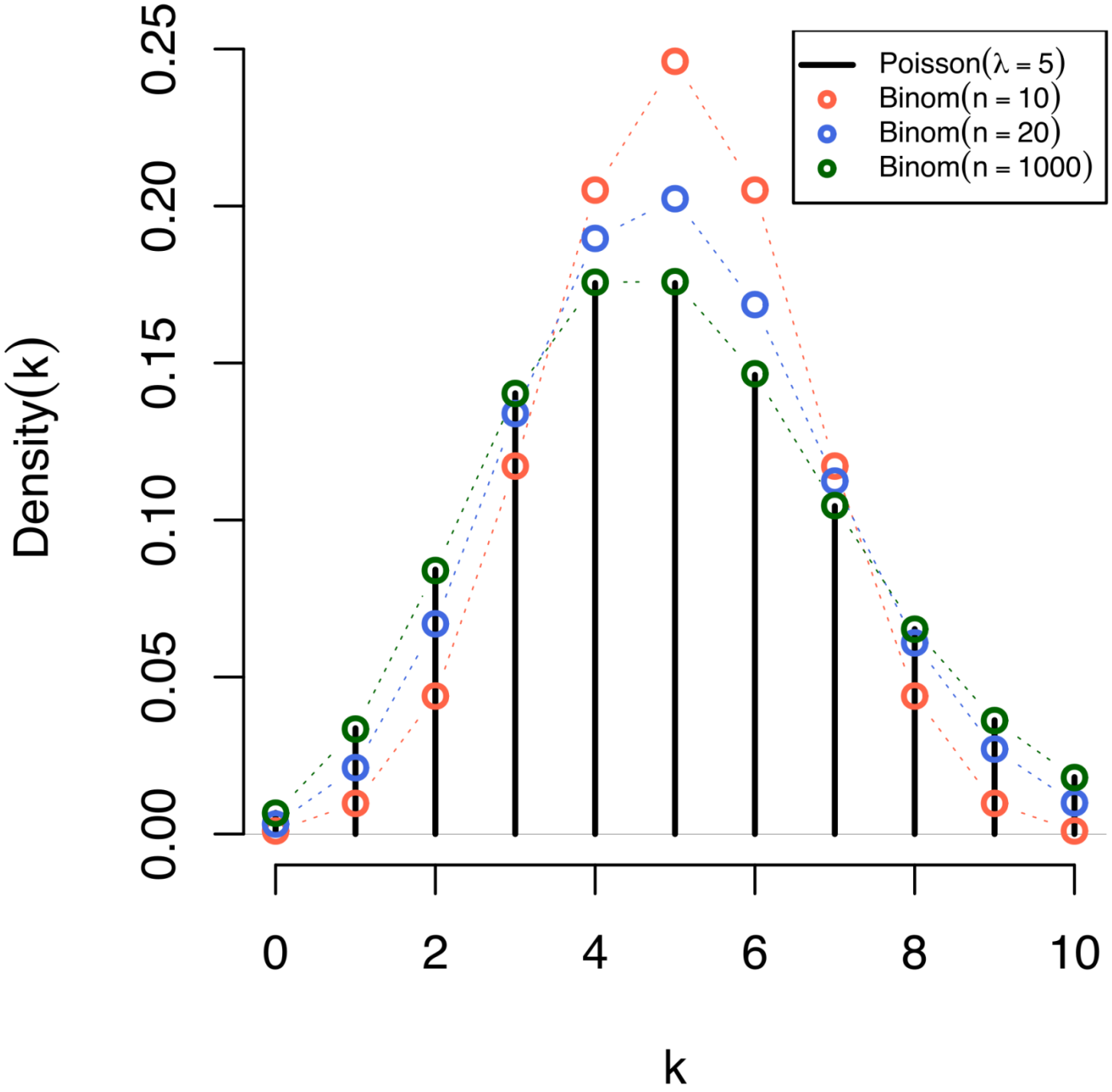
Random Numbers

Poisson Distribution

http://en.wikipedia.org/wiki/Poisson_distribution

Poisson Distribution

- The Poisson distribution is what you get when you take binomial distribution and let μ get small but keep product μN fixed.
  - This is actually what I used in discussing histograms
- The Poisson distribution describes exactly nuclear decays and can be defined as distribution of events at time T where in small time interval δt, the probability of an event arriving is $\lambda \delta t$
- Probability (k events) = $(\lambda T)^k \exp(-\lambda T)/ k!$
- (Written in Wikipedia for case T=1 and varies λ not fixing λ and varying T)
- For Poisson: Mean = $\sigma^2 = \lambda T$



Comparison of the Poisson distribution (black lines) and thebinomial distribution with n=10 (red circles), n=20 (blue circles), n=1000 (green circles). All distributions have a mean of 5. The horizontal axis shows the number of events k. Notice that as n gets larger, the Poisson distribution becomes an increasingly better approximation for the binomial distribution with the same mean. (Wikipedia)

Poisson Examples I (Wikipedia)

- Applications of the Poisson distribution can be found in many fields related to counting:
- Electrical system example: telephone calls arriving in a system.
- Astronomy example: photons arriving at a telescope.
- Biology example: the number of mutations on a strand of DNA per unit length.
- Management example: customers arriving at a counter or call centre.
- Civil engineering example: cars arriving at a traffic light.
- Finance and insurance example: Number of Losses/ Claims occurring in a given period of Time.
- Earthquake seismology example: An asymptotic Poisson model of seismic risk for large earthquakes.

Poisson Examples II (Wikipedia)

- The Poisson distribution arises in connection with Poisson processes. It applies to various phenomena of discrete properties (that is, those that may happen 0, 1, 2, 3, ... times during a given period of time or in a given area) whenever the probability of the phenomenon happening is constant in time or space. Examples of events that may be modelled as a Poisson distribution include:
- The number of soldiers killed by horse-kicks each year in each corps in the Prussian cavalry. This example was made famous by a book of Ladislaus Josephovich Bortkiewicz (1868–1931).
- The number of yeast cells used when brewing Guinness beer. This example was made famous by William Sealy Gosset (1876–1937).
- The number of phone calls arriving at a call centre per minute.
- The number of goals in sports involving two competing teams.
- The number of deaths per year in a given age group.
- The number of jumps in a stock price in a given time interval.
- Under an assumption of homogeneity, the number of times a web server is accessed per minute.
- The number of mutations in a given stretch of DNA after a certain amount of radiation.
- The proportion of cells that will be infected at a given multiplicity of infection.
- The targeting of V-1 rockets on London during World War II.
- These are often called “birth processes”

Random Numbers

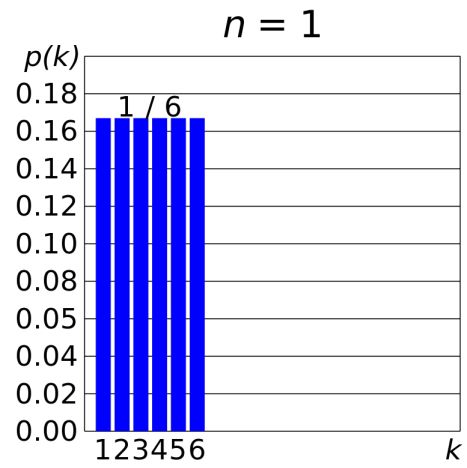
Central Limit Theorem

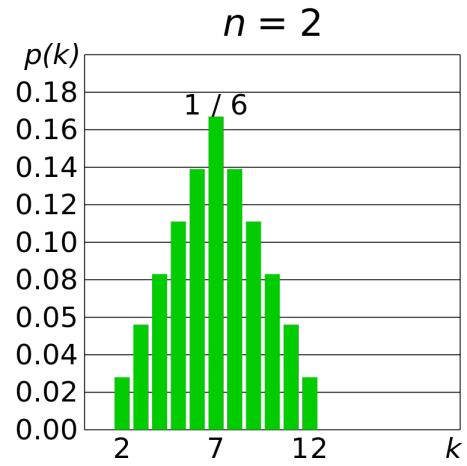
See

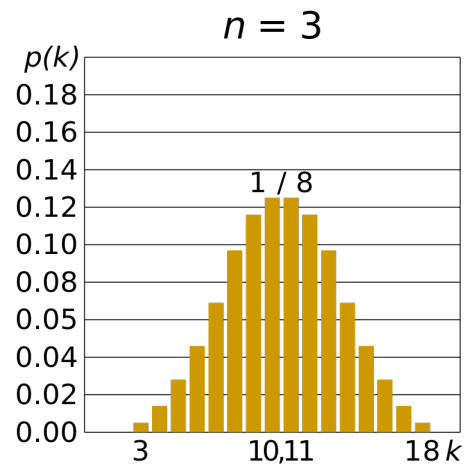
http://en.wikipedia.org/wiki/Central_limit_theorem

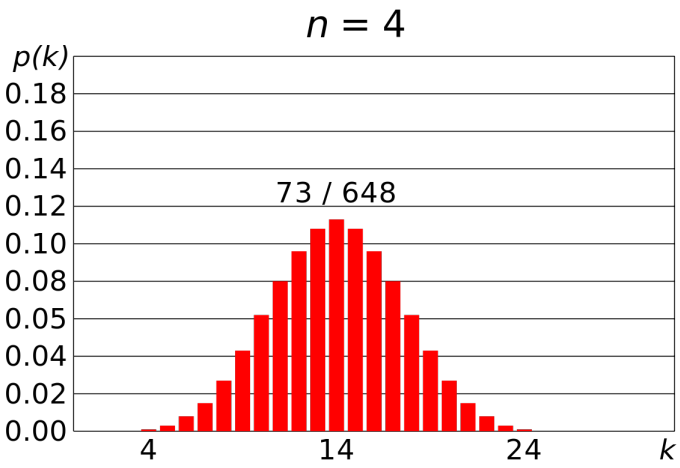
Central Limit Theorem

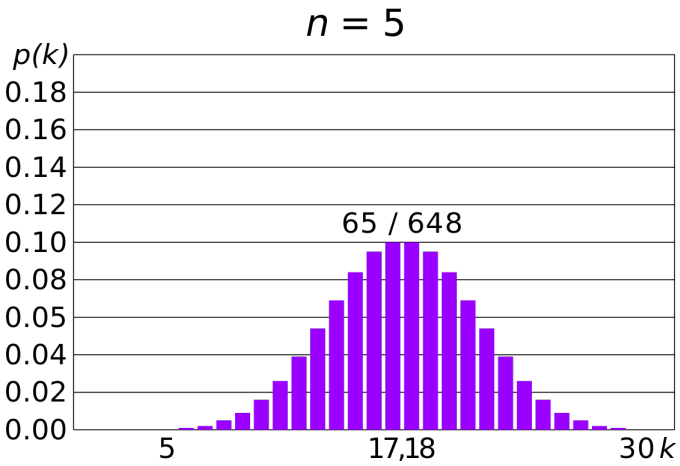
- The large of large numbers (which is usually most important operationally) tells you about the mean and standard deviation (error) of the sum of many IID random variables
- The Central Limit Theorem extends this to tell you about shape of probability distribution and says that for well behaved cases, the shape is that of Gaussian or normal distribution WHATEVER original distribution of X was
- i.e. probability of values far from mean fall off exponentially
- $Pr(O = k) \propto \exp\{ - (k - \langle O \rangle)^2 / 2\langle O \rangle\}$
- Where O (sum off IID's) has mean $\langle O \rangle$

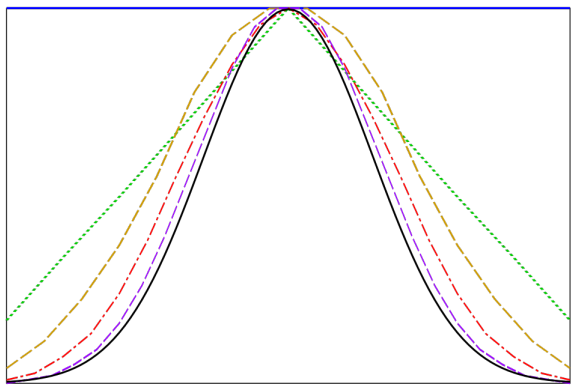




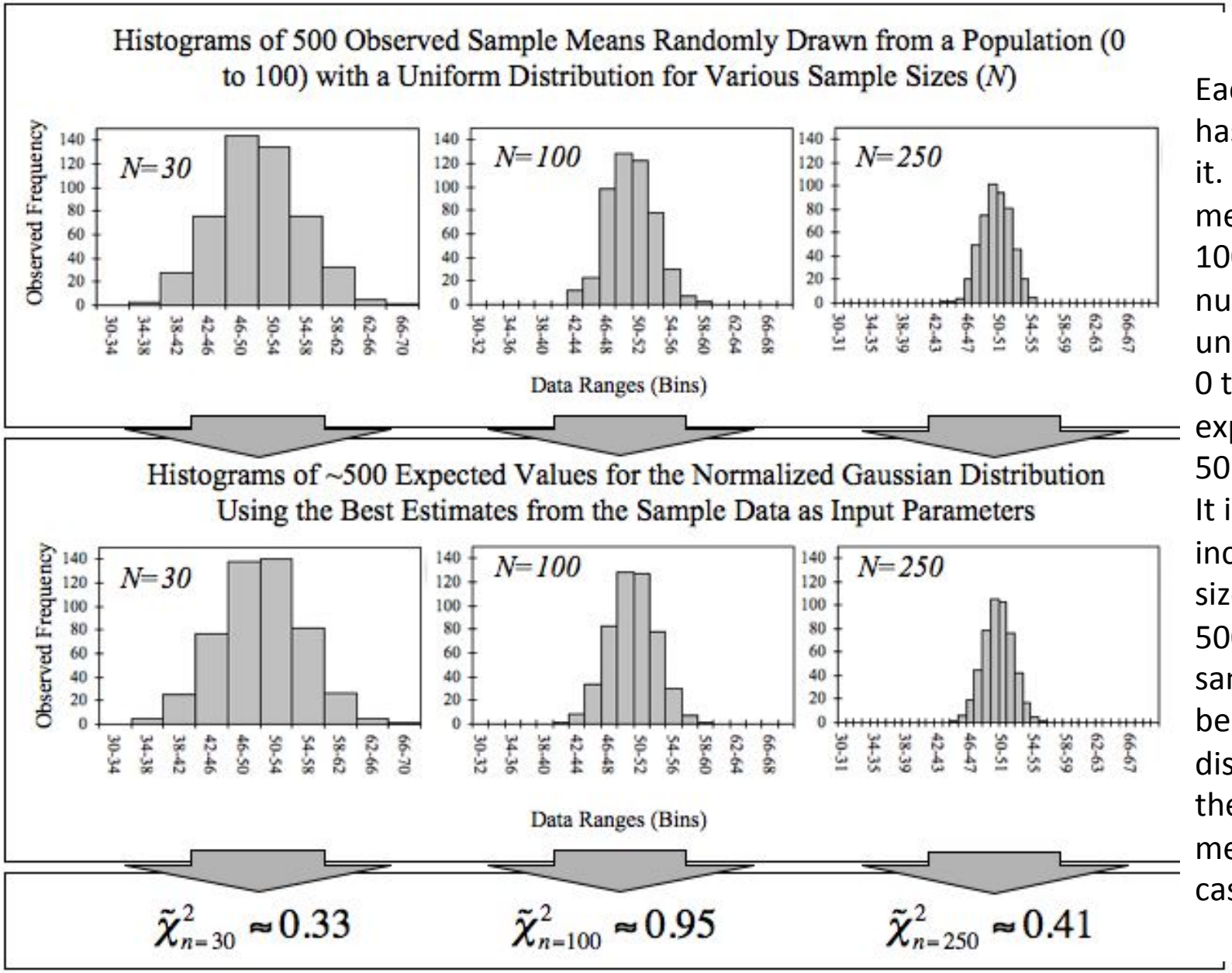








Comparison of probability density functions, p(k) for the sum of n fair 6-sided dice to show their convergence to a normal distribution with increasing n, in accordance to the central limit theorem. In the bottom-right graph, smoothed profiles of the previous graphs are rescaled, superimposed and compared with a normal distribution (black curve).



Each Histogram has 500 points in it. Each point is mean of N (=30, 100, 250) numbers sampled uniformly in range 0 to 100. The expected mean is 50.

It illustrates that increasing sample sizes result in the 500 measured sample means being more closely distributed about the population mean (50 in this case).

It also compares the observed distributions with the distributions that would be expected for a Gaussian distribution, and shows the chi-squared values that quantify the goodness of the fit