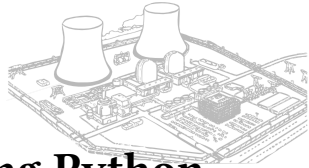
RISK ENGINEERING



Statistics and risk modelling using Python

Eric Marsden

<eric.marsden@risk-engineering.org>

Statistics is the science of learning from experience, particularly experience that arrives a little bit at a time.
— B. Efron, Stanford

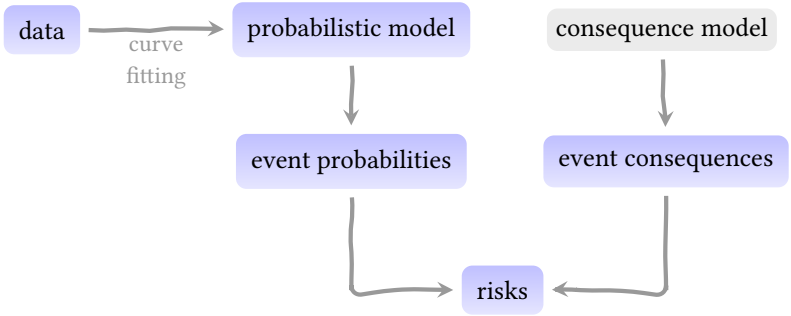
Learning objectives

Using Python/SciPy tools:

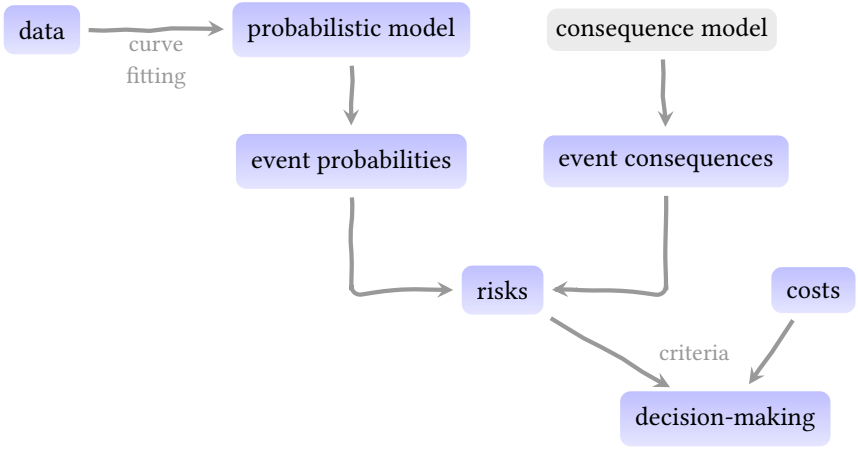
1. Analyze data using descriptive statistics and graphical tools
2. Fit a probability distribution to data (estimate distribution parameters)
3. Express various risk measures as statistical tests
4. Determine quantile measures of various risk metrics
5. Build flexible models to allow estimation of quantities of interest and associated uncertainty measures
6. Select appropriate distributions of random variables/vectors for stochastic phenomena

RISK ENGINEERING

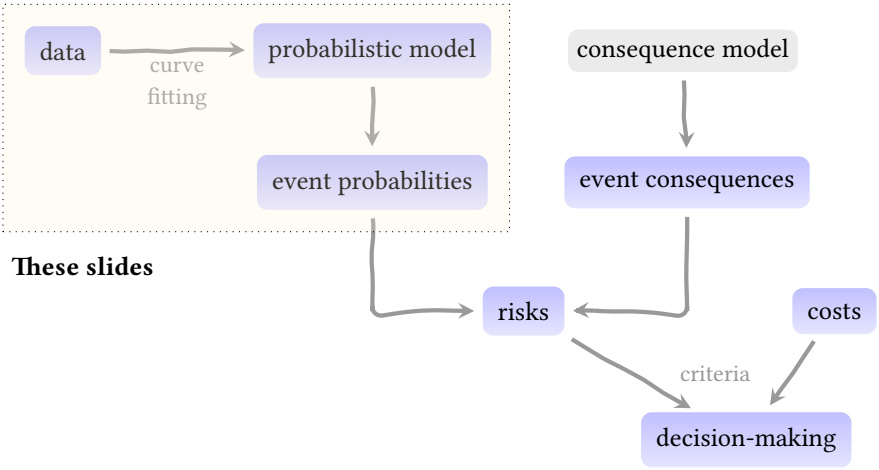
Where does this fit into risk engineering?



Where does this fit into risk engineering?



Where does this fit into risk engineering?



Angle of attack: computational approach to statistics

- Emphasize practical results rather than formulæ and proofs
- Include new statistical tools which have become practical thanks to power of modern computers
  - called “resampling” or “simulation” or “Monte Carlo” methods
- Our target: “Analyze risk-related data using computers”
- If talking to a recruiter, use the term **data science**
  - very sought-after skill in 2016!

ARTWORK: TAHAR COHEN, ANDREW J BUBOLTZ, 2011, SILK SCR
ON A PAGE FROM A HIGH SCHOOL YEARBOOK, 8.5" X 12"

DATA

Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

FROM THE OCTOBER 2012 ISSUE

WHAT TO READ NEXT

Big Data: The Management Revolution

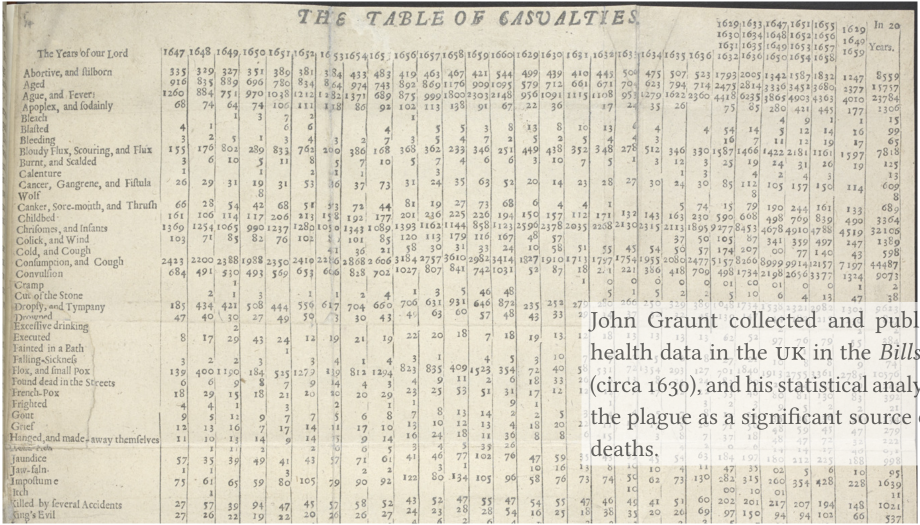
5 Essential Principles for Understanding Analytics

Data Scientists Don't Scale

A sought-after skill

Source: indeed.com/jobtrends

A long history



John Graunt collected and published public health data in the UK in the *Bills of Mortality* (circa 1630), and his statistical analysis identified the plague as a significant source of premature deaths.

Image source: British Library, public domain

Python and SciPy

Environment used in this coursework:

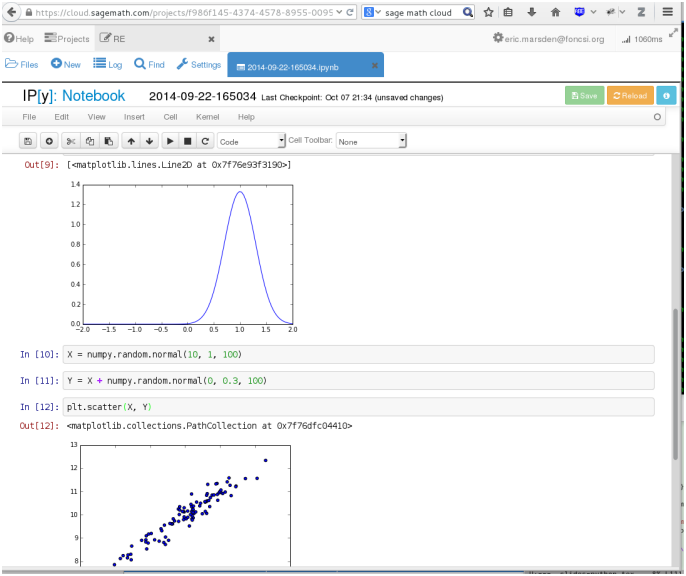
- Python programming language + SciPy + NumPy + matplotlib libraries
- Alternative to Matlab, Scilab, Octave, R
- Free software
- A real programming language with simple syntax
  - much, much more powerful than a spreadsheet!
- Rich scientific computing libraries
  - statistical measures
  - visual presentation of data
  - optimization, interpolation and curve fitting
  - stochastic simulation
  - machine learning, image processing...

Installation options

- **Microsoft Windows**: install one of
  - Anaconda from `continuum.io/downloads`
  - pythonxy from `python-xy.github.io`
- **MacOS**: install one of
  - Anaconda from `continuum.io/downloads`
  - Pyzo, from `pyzo.org`
- **Linux**: install packages `python`, `numpy`, `matplotlib`, `scipy`
  - your distribution's packages are probably fine
- **Tablet/cloud** without local installation: Sage Math Cloud
  - Python and Sage running in the cloud

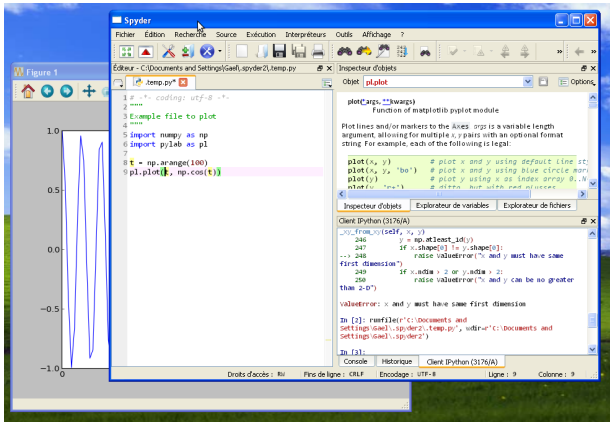
RISK ENGINEERING

Sage Math Cloud



- Runs in cloud, access via browser
- No local installation needed
- Access to Python in a Jupyter notebook, Sage, R
- Create an account for free
- `cloud.sagemath.com`
- Similar: `pythonanywhere.com`

Spyder environment for Python

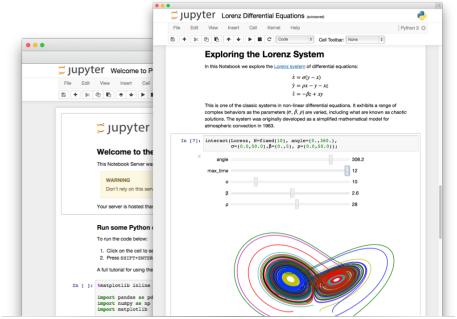


GUI interface to local Python installation

`code.google.com/p/spyderlib/`

Python: interactive work in a web browser

- Jupyter python notebook: interact via your web browser
- Type expressions into a cell then press `Ctrl-Enter`
- Results appear in the web page
- Jupyter notebooks are live computational documents which allow the embedding of rich media (anything that a web browser can display)
  - try one out online at `try.jupyter.org`



More on Jupyter at `jupyter.org`

Python: loading a program from a script

- For more complex problems, write a program into a file
- Load the program using implementation's `Load` facility

Python as a statistical calculator

```
In [1]: import numpy

In [2]: 2 + 2
Out[2]: 4

In [3]: numpy.sqrt(2 + 2)
Out[3]: 2.0

In [4]: numpy.pi
Out[4]: 3.141592653589793

In [5]: numpy.sin(numpy.pi)
Out[5]: 1.2246467991473532e-16

In [6]: numpy.random.uniform(20, 30)
Out[6]: 28.890905809912784

In [7]: numpy.random.uniform(20, 30)
Out[7]: 20.58728078429875
```

Download this content as a Python notebook at risk-engineering.org

RISK ENGINEERING

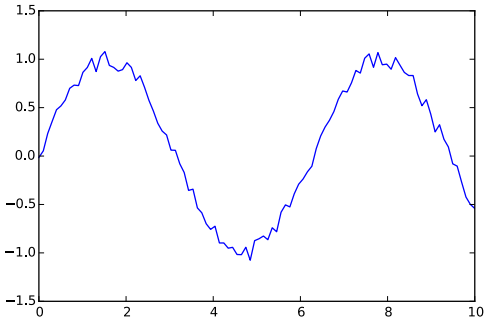
Python as a statistical calculator

```
In [3]: obs = numpy.random.uniform(20, 30, 10)

In [4]: obs
Out[4]:
array([ 25.64917726,  21.35270677,  21.71122725,  27.94435625,
        25.43993038,  22.72479854,  22.35164765,  20.23228629,
        26.05497056,  22.01504739])

In [5]: len(obs)
Out[5]: 10

In [6]: obs + obs
Out[6]:
array([ 51.29835453,  42.70541355,  43.42245451,  55.8887125 ,
        50.87986076,  45.44959708,  44.7032953 ,  40.46457257,
        52.10994112,  44.03009478])

In [7]: obs - 25
Out[7]:
array([ 0.64917726, -3.64729323, -3.28877275,  2.94435625,
0.43993038,
       -2.27520146, -2.64835235, -4.76771371,  1.05497056,
-2.98495261])

In [8]: obs.mean()
Out[8]: 23.547614834213316

In [9]: obs.sum()
Out[9]: 235.47614834213317

In [10]: obs.min()
Out[10]: 20.232286285845483
```

RISK ENGINEERING

Python as a statistical calculator: plotting

```
In [2]: import numpy, matplotlib.pyplot as plt

In [3]: x = numpy.linspace(0, 10, 100)

In [4]: obs = numpy.sin(x) + numpy.random.uniform(-0.1, 0.1, 100)

In [5]: plt.plot(x, obs)
Out[5]: [<matplotlib.lines.Line2D at 0x7f47ecc96da0>]

In [7]: plt.plot(x, obs)
Out[7]: [<matplotlib.lines.Line2D at 0x7f47ed42f0f0>]
```



Coming from Matlab

Generate random numbers

MATLAB/Octave	Python	Description
`rand(1,10)`	`random.random((10,))` `random.uniform((10,))`	Uniform distribution
`2+5*rand(1,10)`	`random.uniform(2,7,(10,))`	Uniform: Numbers between 2 and 7
`rand(6)`	`random.uniform(0,1,(6,6))`	Uniform: 6,6 array
`randn(1,10)`	`random.standard_normal((10,))`	Normal distribution

Vectors

MATLAB/Octave	Python	Description
`a=[2 3 4 5];`	`a=array([2,3,4,5])`	Row vector, $1 \times n$ matrix

Source: `http://mathesaurus.sourceforge.net/matlab-numpy.html`

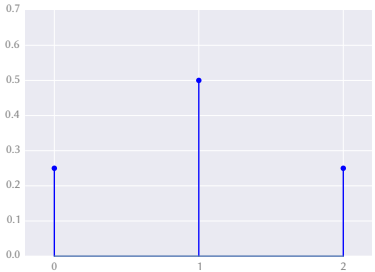
RISK ENGINEERING

Probability Mass Functions

- A random variable is a set of possible values from a stochastic experiment
- For all values x that a discrete random variable X may take, we define the function

$$p_X(x) \triangleq \Pr(X \text{ takes the value } x)$$

- This is called the **probability mass function** (PMF) of X
- Example: X = *"number of heads when tossing a coin twice"*
  - $p_X(0) \triangleq \Pr(X = 0) = 1/4$
  - $p_X(1) \triangleq \Pr(X = 1) = 2/4$
  - $p_X(2) \triangleq \Pr(X = 2) = 1/4$



Probability Mass Functions: two coins

- **Task**: simulate *"expected number of heads when tossing a coin twice"*

- Let's simulate a coin toss by random choice between 0 and 1

```
> numpy.random.random_integers(0, 1)
1
```

minimum

maximum

Download this content as a Python notebook at `risk-engineering.org`

Probability Mass Functions: two coins

- **Task**: simulate *"expected number of heads when tossing a coin twice"*

- Let's simulate a coin toss by random choice between 0 and 1

```
> numpy.random.random_integers(0, 1)
1
```

minimum maximum

- Toss a coin twice:

```
> numpy.random.random_integers(0, 1, 2)
array([0, 1])
```

count

Download this content as a Python notebook at `risk-engineering.org`

Probability Mass Functions: two coins

- **Task**: simulate *"expected number of heads when tossing a coin twice"*

- Let's simulate a coin toss by random choice between 0 and 1

```
> numpy.random.random_integers(0, 1)
1
```

minimum maximum

- Toss a coin twice:

```
> numpy.random.random_integers(0, 1, 2)
array([0, 1])
```

count

Download this content as a Python notebook at `risk-engineering.org`

- Number of heads when tossing a coin twice:

```
> numpy.random.random_integers(0, 1, 2).sum()
1
```

RISK ENGINEERING

Probability Mass Functions: two coins

- **Task**: simulate *"expected number of heads when tossing a coin twice"*
- Do this 1000 times and plot the resulting PMF:

```
import matplotlib.pyplot as plt

N = 1000
heads = numpy.zeros(N, dtype=int)
for i in range(N):
   heads[i] = numpy.random.random_integers(0, 1, 2).sum()
plt.stem(numpy.bincount(heads), marker='o')
plt.margins(0.1)
```

`heads[i]`: *element number i of the array* `heads`

RISK ENGINEERING

More information on Python programming

For more information on Python syntax, check out the book *Think Python*

Purchase, or read online for free at `greenteapress.com/thinkpython`

RISK ENGINEERING

Probability Mass Functions: properties

- A PMF is always non-negative

$$p_X(x) \geq 0$$

- Sum over the support equals 1

$$\sum_x p_X(x) = 1$$

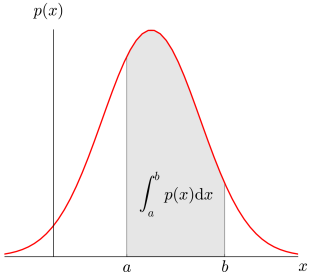
-

$$\Pr(a \leq X \leq b) = \sum_{x \in [a,b]} p_X(x)$$

Probability Density Functions

- For continuous random variables, the **probability density function** $f_X(x)$ is defined by

$$\Pr(a \leq X \leq b) = \int_a^b f_X(x)dx$$



- It is non-negative

$$f_X(x) \geq 0$$

- The integral over $\mathbb{R}$ is 1

$$\int_{-\infty}^{\infty} f_X(x)dx = 1$$

Expectation of a random variable

- The **expectation** (or mean) is defined as a weighted average of all possible realizations of a random variable
- Discrete random variable:

$$\mathbb{E}[X] = \mu_X \stackrel{\text{def}}{=} \sum_{i=1}^{N} x_i \times \Pr(X = x_i)$$

- Continuous random variable:

$$\mathbb{E}[X] = \mu_X \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} u \times f(u) du$$

- Interpretation:
  - the center of gravity of the PMF or PDF
  - the average in a large number of independent realizations of your experiment

Illustration: expected value with coins

- X = *"number of heads when tossing a coin twice"*
- **PMF:**
  - $p_X(0) \triangleq \Pr(X = 0) = 1/4$
  - $p_X(1) \triangleq \Pr(X = 1) = 2/4$
  - $p_X(2) \triangleq \Pr(X = 2) = 1/4$
- $\mathrm{E}[X] \triangleq \sum_k k \times p_X(k) = 0 \times \frac{1}{4} + 1 \times \frac{2}{4} + 2 \times \frac{1}{4} = 1$

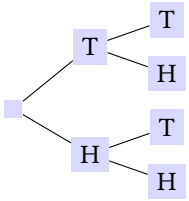
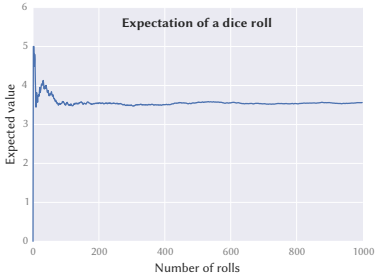


Illustration: expected value of a dice roll

- Expected value of a dice roll is $\sum_{i=1}^{6} i \times \frac{1}{6} = 3.5$
- If we toss a dice a large number of times, the mean value should converge to 3.5
- Let's check that in Python

```
N = 1000
roll = numpy.zeros(N)
expectation = numpy.zeros(N)
for i in range(N):
    roll[i] = numpy.random.random_integers(1, 6)
for i in range(1, N):
    expectation[i] = numpy.mean(roll[0:i])
plt.plot(expectation)
```



Mathematical properties of expectation

If c is a constant and X and Y are random variables, then

- $\mathbb{E}[c] = c$
- $\mathbb{E}[cX] = c\mathbb{E}[X]$
- $\mathbb{E}[c + X] = c + \mathbb{E}[X]$
- $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

Note: in general $\mathbb{E}[g(X)] \neq g(\mathbb{E}[X])$

Aside: existence of expectation

- Not all random variables have an expectation
- Consider a random variable X defined on some (infinite) sample space Ω so that for all positive integers i, X takes the value
  - 2^i with probability 2^{-i-1}
  - -2^i with probability 2^{-i-1}
- Both the positive part and the negative part of X have infinite expectation in this case, so $\mathbb{E}[X]$ would have to be $\infty - \infty$ (meaningless)

Simple descriptive statistics

Task: Choose randomly 1000 integers from a uniform distribution between 100 and 200. Calculate the mean, min, max, variance and standard deviation of this sample.

```
> import numpy
> obs = numpy.random.random_integers(100, 200, 1000)
> obs.mean()
149.49199999999999
> obs.min()
100
> obs.max()
200
> obs.var()
823.99793599999998
> obs.std()
28.705364237368595
```

Variance of a random variable

- The **variance** provides a measure of the dispersion around the mean
  - intuition: how "spread out" the data is
- For a discrete random variable:

$$\mathrm{Var}(X) = \sigma_X^2 \stackrel{\text{def}}{=} \sum_{i=1}^{N} (X_i - \mu_X)^2 \Pr(X = x_i)$$

- For a continuous random variable:

$$\mathrm{Var}(X) = \sigma_X^2 \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} (x - \mu_x)^2 f(u) du$$

- In Python:
  - `obs.var()` if `obs` is a NumPy vector
  - `numpy.var(obs)` for any Python sequence (vector or list)

In Excel, function VAR

RISK ENGINEERING

Variance with coins

- X = *"number of heads when tossing a coin twice"*
- **PMF:**
  - $p_X(0) \triangleq \Pr(X = 0) = 1/4$
  - $p_X(1) \triangleq \Pr(X = 1) = 2/4$
  - $p_X(2) \triangleq \Pr(X = 2) = 1/4$
-

$$
\begin{aligned}
Var(X) &\triangleq \sum_{i=1}^{N} (X_i - \mu_X)^2 \Pr(X = x_i) \\
&= \frac{1}{4} \times (0-1)^2 + \frac{2}{4} \times (1-1)^2 + \frac{1}{4} \times (2-1)^2 = \frac{1}{2}
\end{aligned}
$$

RISK ENGINEERING

Variance of a dice roll

- Analytic calculation of the variance of a dice roll:

$$
\begin{aligned}
Var(X) &\stackrel{\text{def}}{=} \sum_{i=1}^{N} (X_i - \mu_X)^2 \Pr(X = x_i) \\
&= \frac{1}{6} \times \left((1 - 3.5)^2 + (2 - 3.5)^2 + (3 - 3.5)^2 + (4 - 4.5)^2 + (5 - 3.5)^2 + (6 - 3.5)^2 \right) \\
&= 2.916
\end{aligned}
$$

- Let's reproduce that in Python

count

```
> rolls = numpy.random.random_integers(1, 6, 1000)
> len(rolls)
1000
> rolls.var()
2.9463190000000004
```

Properties of variance as a mathematical operator

If c is a constant and X and Y are random variables, then

- $\text{Var}(X) \geq 0$ (variance is always non-negative)
- $\text{Var}(c) = 0$
- $\text{Var}(c + X) = \text{Var}(X)$
- $\text{Var}(cX) = c^2\ \text{Var}(X)$
- $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$, **if X and Y are independent**
- $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y)$ if X and Y are dependent

Beware:

- $\mathbb{E}[X^2] \neq (\mathbb{E}[X])^2$
- $\mathbb{E}[\sqrt{X}] \neq \sqrt{\mathbb{E}[X]}$

Properties of variance as a mathematical operator

If c is a constant and X and Y are random variables, then

- $\text{Var}(X) \geq 0$ (variance is always non-negative)
- $\text{Var}(c) = 0$
- $\text{Var}(c + X) = \text{Var}(X)$
- $\text{Var}(cX) = c^2\ \text{Var}(X)$
- $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$, **if X and Y are independent**
- $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y)$ if X and Y are dependent

Beware:

- $\mathbb{E}[X^2] \neq (\mathbb{E}[X])^2$
- $\mathbb{E}[\sqrt{X}] \neq \sqrt{\mathbb{E}[X]}$

Note:

- $\text{Cov}(X, Y) \stackrel{\text{def}}{=} \mathbb{E}\left[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])\right]$
- $\text{Cov}(X, X) = \text{Var}(X)$

Standard deviation

- Formula for variance: $\mathrm{Var}(X) \triangleq \sum_{i=1}^{N} (X_i - \mu_X)^2 \Pr(X = x_i)$
- If random variable X is expressed in metres, $\mathrm{Var}(X)$ is in m^2
- To obtain a measure of dispersion of a random variable around its expected value which has the same units as the random variable itself, take the square root
- Standard deviation $\sigma \triangleq \sqrt{\mathrm{Var}(X)}$
- In Python:
  - `obs.std()` if `obs` is a NumPy vector
  - `numpy.std(obs)` for any Python sequence (vector or list)

In Excel, function STDEV

Properties of standard deviation

- Suppose $Y = aX + b$, where
  - a and b are scalar
  - X and Y are two random variables
- Then $\sigma(Y) = |a|\,\sigma(X)$
- Let's check that with NumPy:

```
> X = numpy.random.uniform(100, 200, 1000)
> a = -32
> b = 16
> Y = a * X + b
> Y.std()
914.94058476118835
> abs(a) * X.std()
914.94058476118835
```

Properties of expectation & variance: empirical testing with numpy

- $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

```
> X = numpy.random.random_integers(1, 100, 1000)
> Y = numpy.random.random_integers(-300, -200, 1000)
> X.mean()
50.575000000000003
> Y.mean()
-251.613
> (X + Y).mean()
-201.038
```

Properties of expectation & variance: empirical testing with numpy

- $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

```
> X = numpy.random.random_integers(1, 100, 1000)
> Y = numpy.random.random_integers(-300, -200, 1000)
> X.mean()
50.575000000000003
> Y.mean()
-251.613
> (X + Y).mean()
-201.038
```

- $\mathrm{Var}(cX) = c^2\ \mathrm{Var}(X)$

```
> numpy.random.random_integers(1, 100, 1000).var()
836.616716
> numpy.random.random_integers(5, 500, 1000).var()
20514.814318999997
> 5 * 5 * 836
20900
```

Properties of expectation & variance: empirical testing with numpy

- $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

```
> X = numpy.random.random_integers(1, 100, 1000)
> Y = numpy.random.random_integers(-300, -200, 1000)
> X.mean()
50.575000000000003
> Y.mean()
-251.613
> (X + Y).mean()
-201.038
```

- $\mathrm{Var}(cX) = c^2\ \mathrm{Var}(X)$

```
> numpy.random.random_integers(1, 100, 1000).var()
836.616716
> numpy.random.random_integers(5, 500, 1000).var()
20514.814318999997
> 5 * 5 * 836
20900
```

Cumulative Distribution Function

- The cumulative distribution function (CDF) of random variable X, denoted by $F(x)$, indicates the probability that X assumes a value $\leq x$, where x is any real number

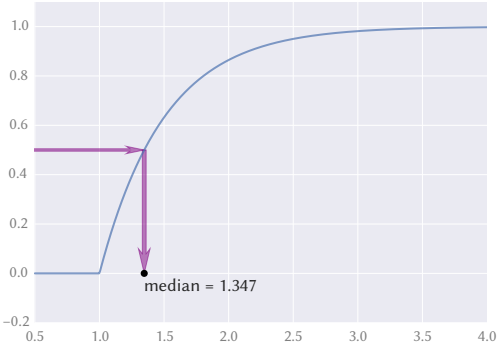
$$F(x) = \Pr(X \leq x) \qquad -\infty \leq x \leq \infty$$

- Properties of a CDF:
  - $F(x)$ is a non-decreasing function of x
  - $0 \leq F(x) \leq 1$
  - $\lim_{x \to \infty} F(x) = 1$
  - $\lim_{x \to -\infty} F(x) = 0$
  - if $x \leq y$ then $F(x) \leq F(y)$
  - $\Pr(a < X \leq b) = F(b) - F(a) \quad \forall b > a$

Interpreting the Cumulative Distribution Function

How do you use the CDF?

The **median** is the point where half the population is below and half is above (it's the 0.5 quantile).



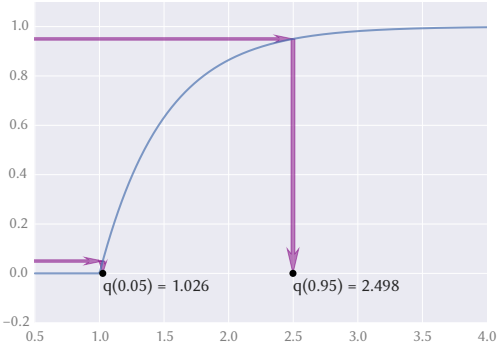
RISK ENGINEERING

Interpreting the Cumulative Distribution Function

How do you use the CDF?

A **90% confidence interval** is the set of points between the 0.05 quantile (5% of my observations are smaller than this value) and the 0.95 quantile (only 5% of my observations are larger than this value).

In the example to the right, the 90% confidence interval is [1.026, 2.498].

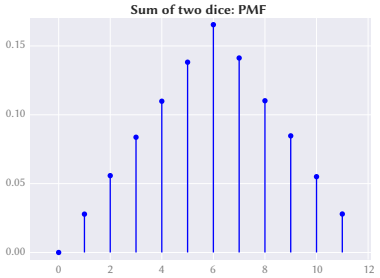


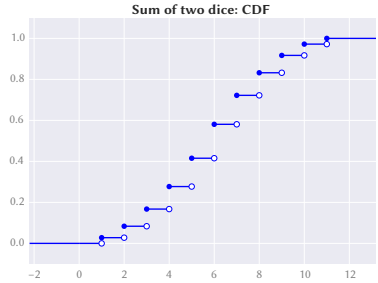
RISK ENGINEERING

CDF of a discrete distribution

$$F(x) = \Pr(X \leq x) \qquad -\infty \leq x \leq \infty$$
$$= \sum_{x_i \leq x} \Pr(X = x_i)$$

Example: sum of two dice





RISK ENGINEERING

Scipy.stats package

- The `scipy.stats` module implements many continuous and discrete random variables and their associated distributions
  - binomial, poisson, exponential, normal, uniform, weibull…
- Usage: instantiate a distribution then call a method
  - `rvs`: random variates
  - `pdf`: Probability Density Function
  - `cdf`: Cumulative Distribution Function
  - `sf`: Survival Function (1-CDF)
  - `ppf`: Percent Point Function (inverse of CDF)
  - `isf`: Inverse Survival Function (inverse of SF)

Simulating dice throws

- Maximum of 1000 throws

```
> dice = scipy.stats.randint(1, 7)
> dice.rvs(1000).max()
6
```

exclusive upper bound

- What is the probability of a die rolling 4?

```
> dice.pmf(4)
0.16666666666666666
```

- What is the probability of rolling 4 or below?

```
> dice.cdf(4)
0.6666666666666663
```

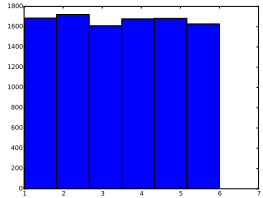
- What is the probability of rolling between 2 and 4 (inclusive)?

```
> dice.cdf(4) - dice.cdf(1)
0.5
```

RISK ENGINEERING

Simulating dice

```
> import numpy
> import matplotlib.pyplot as plt

> tosses = numpy.random.choice(range(1, 7))
> tosses
2
> tosses = numpy.random.choice(range(1, 7), 10000)
> tosses
array([6, 6, 4, ..., 2, 4, 5])
> tosses.mean()
3.4815
> numpy.median(tosses)
3.0
> len(numpy.where(tosses > 3)[0])
4951
> plt.hist(tosses, bins=6)
```



Scipy.stats examples

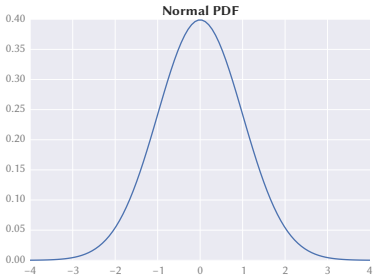
- Generate 5 random variates from a continuous uniform distribution between 90 and 100
- Check that the expected value of the distribution is around 95
- Check that around 20% of variates are less than 92

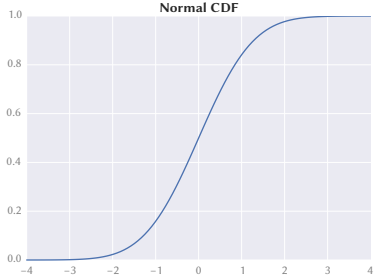
```
> import scipy.stats

> u = scipy.stats.uniform(90, 10)
> u.rvs(5)
array([ 94.0970853 ,  92.41951494,
  90.25127254, 91.69097729,
  96.1811148 ])
> u.rvs(1000).mean()
94.892801456986376
> (u.rvs(1000) < 92).sum() / 1000.0
0.193
```

RISK ENGINEERING

CDF of a continuous distribution

$$F(x) = \Pr(X \leq x)$$
$$= \int_{-\infty}^{x} f(u)du$$





Python: `scipy.stats.norm.pdf(x, loc=mu, scale=sigma)`

Python: `scipy.stats.norm.cdf(x, loc=mu, scale=sigma)`

RISK ENGINEERING

Some important probability distributions

Some important probability distributions

Coin tossing with uneven coin	Bernoulli	`scipy.stats.bernoulli`
Rolling a dice	uniform	`scipy.stats.randint`
Counting errors/successes	Binomial	`scipy.stats.binom`
Trying until success	geometric	`scipy.stats.geom`
Countable, rare events whose occurrence is independent	Poisson	`scipy.stats.poisson`
Random “noise”, sums of many variables	normal	`scipy.stats.norm`

RISK ENGINEERING

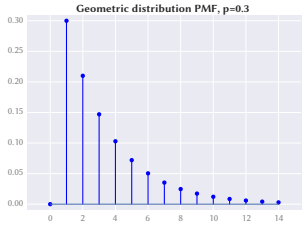
Bernoulli trials

- A *trial* is an experiment which can be repeated many times with the same probabilities, each realization being independent of the others
- Bernoulli trial: an experiment in which N trials are made of an event, with probability p of "success" in any given trial and probability $1 - p$ of "failure"
  - "success" and "failure" are mutually exclusive
  - example: sequence of coin tosses
  - example: arrival of requests in a web server per time slot

Jakob Bernoulli (1654–1705)

The geometric distribution (trying until success)

- We conduct a sequence of Bernoulli trials, each with success probability p
- What's the probability that it takes k trials to get a success?
  - Before we can succeed at trial k, we must have had $k - 1$ failures
  - Each failure occurred with probability $1 - p$, so total probability $(1 - p)^{k-1}$
  - Then a single success with probability p
- $\Pr(X = k) = (1 - p)^{k-1} p$

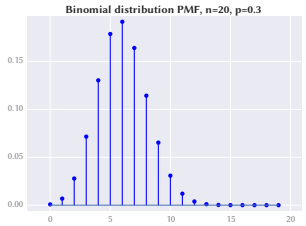


The geometric distribution: intuition

- Suppose I am at a party and I start asking girls to dance. Let X be the number of girls that I need to ask in order to find a partner.
  - If the first girl accepts, then $X = 1$
  - If the first girl declines but the next girl accepts, then $X = 2$
- $X = k$ means that I failed on the first $k - 1$ tries and succeeded on the k^{th} try
  - My probability of failing on the first try is $(1 - p)$
  - My probability of failing on the first two tries is $(1 - p)(1 - p)$
  - My probability of failing on the first $n - 1$ tries is $(1 - p)^{k-1}$
  - Then, my probability of succeeding on the n^{th} try is p
- Properties:
  - $\mathrm{E}[X] = \frac{1}{p}$
  - $Var(X) = \frac{1-p}{p^2}$

The binomial distribution (counting successes)

- Also arises when observing multiple Bernoulli trials
  - exactly two mutually exclusive outcomes, “success” and “failure”
- *Binomial*(p, k, n): probability of observing k successes in n trials, where probability of success on a single trial is p
  - example: toss a coin n times ($p = 0.5$) and see k heads
- We have k successes, which happens with a probability of p^k
- We have $n - k$ failures, which happens with probability $(1 - p)^{n-k}$
- We can generate these k successes in many different ways from n trials, $\binom{n}{k}$ ways
- $\Pr(X = k) = \binom{n}{k}(1 - p)^{n-k}p^k$



Reminder: binomial coefficient $\binom{n}{k}$ is $\frac{n!}{k!(n-k)!}$

Binomial distribution: application

- Consider a medical test with an error rate of 0.1 applied to 100 patients
- What is the probability that we see at most 1 test error?
- What is the probability that we see at most 10 errors?
- What is the smallest k such that $P(X \leq k)$ is at least 0.05?

```
> import scipy.stats

> test = scipy.stats.binom(n=100, p=0.1)
> test.cdf(1)
0.00032168805319411544
> test.cdf(10)
0.58315551226649232
> test.ppf(0.05)
5.0
```

RISK ENGINEERING

Binomial distribution: application

Q: A company drills 9 oil exploration wells, each with a 10% chance of success. Eight of the nine wells fail. What is the probability of that happening?

Binomial distribution: application

Q: A company drills 9 oil exploration wells, each with a 10% chance of success. Eight of the nine wells fail. What is the probability of that happening?

Analytic solution

Each well is a binomial trial with $p = 0.1$. We want the probability of exactly one success.

```
> import scipy.stats
> wells = scipy.stats.binom(n=9, p=0.1)
> wells.pmf(1)
0.38742048899999959
```

Answer by simulation

Run 20 000 trials of the model and count the number that generate 1 positive result

```
> import scipy.stats
> N = 20000
> wells = scipy.stats.binom(n=9, p=0.1)
> trials = wells.rvs(N)
> (trials == 1).sum() / float(N)
0.38679999999999998
```

RISK ENGINEERING

Binomial distribution: application

Q: A company drills 9 oil exploration wells, each with a 10% chance of success. Eight of the nine wells fail. What is the probability of that happening?

Analytic solution

Each well is a binomial trial with $p = 0.1$. We want the probability of exactly one success.

```
> import scipy.stats
> wells = scipy.stats.binom(n=9, p=0.1)
> wells.pmf(1)
0.38742048899999959
```

Answer by simulation

Run 20 000 trials of the model and count the number that generate 1 positive result

```
> import scipy.stats
> N = 20000
> wells = scipy.stats.binom(n=9, p=0.1)
> trials = wells.rvs(N)
> (trials == 1).sum() / float(N)
0.38679999999999998
```

The probability of all 9 wells failing is $0.9^9 = 0.3874$ (and also `wells.pmf(0)`).

The probability of *at least* 8 wells failing is `wells.cdf(1)`. It's also `wells.pmf(0) + wells.pmf(1)` (it's 0.7748).

RISK ENGINEERING

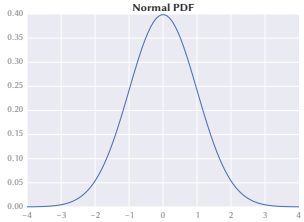
Binomial distribution: properties

Exercise: check empirically (with SciPy) the following properties of the binomial distribution:

- the mean of the distribution (μ_x) is equal to $n \times p$
- the variance (σ_X^2) is $n \times p \times (1 - p)$
- the standard deviation (σ_x) is $\sqrt{n \times p \times (1 - p)}$

Gaussian (normal) distribution

- The famous “bell shaped” curve, fully described by its mean and standard deviation
- Good representation of distribution of measurement errors and many population characteristics
  - size, mechanical strength, duration, speed
- Symmetric around the mean
- Mean = median = mode



RISK ENGINEERING

Scipy.stats examples

- Consider a Gaussian distribution centered in 5, standard deviation of 1
- Check that half the distribution is located to the left of 5
- Find the first percentile (value of x which has 1% of realizations to the left)
- Check that it is equal to the 99% survival quantile

```
> norm = scipy.stats.norm(5, 1)
> norm.cdf(5)
0.5
> norm.ppf(0.5)
5.0
> norm.ppf(0.01)
2.6736521259591592
> norm.isf(0.99)
2.6736521259591592
> norm.cdf(2.67)
0.0099030755591642452
```

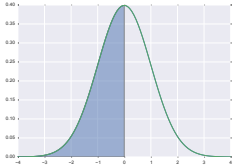
RISK ENGINEERING

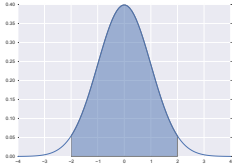
Area under the normal distribution

```
In [1]: import numpy

In [2]: from scipy.stats import norm

In [3]: norm.ppf(0.5)
Out[3]: 0.0

In [4]: norm.cdf(0)
Out[4]: 0.5

In [5]: norm.cdf(2) - norm.cdf(-2)
Out[5]: 0.95449973610364158

In [6]: norm.cdf(3) - norm.cdf(-3)
Out[6]: 0.99730020393673979
```

quantile function

there is a 95% chance that the number drawn falls within 2 standard deviations of the mean





In prehistoric times, statistics textbooks contained large tables of different quantile values of the normal distribution. With cheap computing power, no longer necessary!

The “68–95–99.7 rule”

- The 68–95–99.7 rule (aka the *three-sigma rule*) states that if x is an observation from a normally distributed random variable with mean μ and standard deviation σ, then
  - $\Pr(\mu - \sigma \le x \le \mu + \sigma) \approx 0.6827$
  - $\Pr(\mu - 2\sigma \le x \le \mu + 2\sigma) \approx 0.9545$
  - $\Pr(\mu - 3\sigma \le x \le \mu + 3\sigma) \approx 0.9973$
- The 6σ quality management method pioneered by Motorola aims for 99.99966% of production to meet quality standards
  - 3.4 defective parts per million opportunities (DPMO)
  - actually that’s only 4.5 sigma!
  - `(scipy.stats.norm.cdf(6) - scipy.stats.norm.cdf(-6)) * 100` ⟶ 99.999999802682453

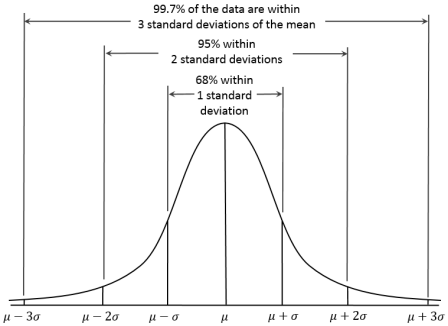
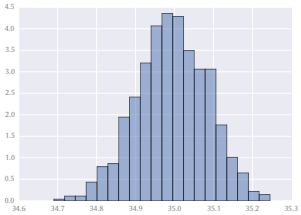


Image source: Wikipedia on 68–95–99.7 rule

Central limit theorem

- Theorem states that the mean (also true of the sum) of a set of random measurements will tend to a normal distribution, no matter the shape of the original measurement distribution
- Part of the reason for the ubiquity of the normal distribution in science
- Python simulation:

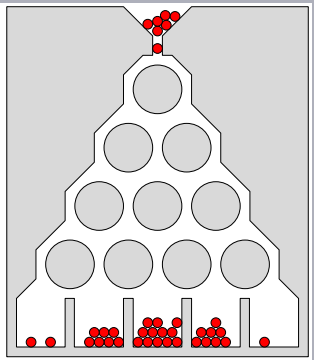
```
N = 10000
sim = numpy.zeros(N)
for i in range(N):
    sim[i] = numpy.random.uniform(30, 40, 100).mean()
plt.hist(sim, bins=20, alpha=0.5, normed=True)
```



- **Exercise**: try this with other probability distributions and check that the simulations tend towards a normal distribution

RISK ENGINEERING

Galton board



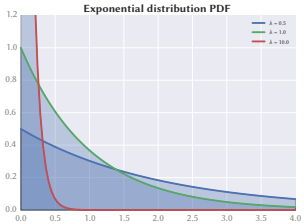
- The Galton board (or "bean machine") has two parts:
  - top: evenly-spaced pegs in staggered rows
  - bottom: evenly-spaced rectangular slots
- Balls introduced at the top bounce of the pegs, equal probability of going right or left at each successive row
  - each vertical step is a Bernoulli trial
- Balls collect in the slots at the bottom with heights following a binomial distribution
  - and for large number of balls, a normal distribution
- Interactive applet emulating a Galton board:

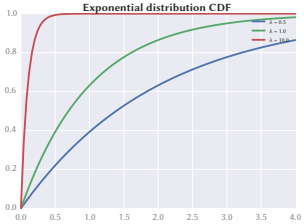
☛ `math.uah.edu/stat/applets/GaltonBoardExperiment.html`

Named after English psychometrician Sir Francis Galton (1822–1911)

Exponential distribution

- PDF: $f_Z(z) = \lambda e^{-\lambda z}, z \geq 0$
- CDF: $\Pr(Z \leq z) = F_Z(z) = 1 - e^{-\lambda z}$
- The hazard function, or *failure rate*, is constant, equal to λ
  - $1/\lambda$ is the "mean time between failures", or MTBF
  - λ can be calculated by $\frac{\text{total number of failures}}{\text{total operating time}}$
- Often used in reliability engineering to represent failure of electronic equipment (no wear)
- Property: expected value of exponential random variable is $\frac{1}{\lambda}$





Exponential distribution

Let's check that the expected value of an exponential random variable is $\frac{1}{\lambda}$

```
> import scipy.stats
> lda = 25
> obs = scipy.stats.expon.rvs(scale=1/float(lda), size=1000)
> obs.mean()
0.041137615318791773
> obs.std()
0.03915081431615041
> 1/float(lda)
0.04
```

RISK ENGINEERING

Exponential distribution: memoryless property

- An exponentially distributed random variable T obeys

$$\Pr(T > s + t | T > s) = \Pr(T > t), \qquad \forall s, t \geq 0$$

- Interpretation: if T represents time of failure
  - The distribution of the remaining lifetime does not depend on how long the component has been operating (item is "as good as new")
  - Distribution of remaining lifetime is the same as the original lifetime
  - An observed failure is the result of some suddenly appearing failure, not due to gradual deterioration

Failure of power transistors (1/2)

- Suppose we are studying the reliability of a power system, which fails if any of 3 power transistors fails
- Let X, Y, Z be random variables modelling failure time of each transistor (in hours)
  - transistors have no physical wear, so model by exponential random variables
- Failures are assumed to be independent
- $X \sim Exp(1/5000)$ (mean failure time of 5000 hours)
- $Y \sim Exp(1/8000)$ (mean failure time of 8000 hours)
- $Z \sim Exp(1/4000)$ (mean failure time of 4000 hours)

RISK ENGINEERING

Failure of power transistors (2/2)

- System fails if any transistor fails, so time to failure T is $\min(X, Y, Z)$

$$
\begin{aligned}
\Pr(T \le t) &= 1 - \Pr(T > t) \\
&= 1 - \Pr(\min(X, Y, Z)) \\
&= 1 - \Pr(X > t, Y > t, Z > t) \\
&= 1 - \Pr(X > t) \times \Pr(Y > t) \times \Pr(Z > t) && \text{(independence)} \\
&= 1 - (1 - \Pr(X \le t))\,(1 - \Pr(Y \le t))\,(1 - \Pr(Z \le t)) \\
&= 1 - (1 - (1 - e^{-t/5000}))\,(1 - (1 - e^{-t/8000}))\,(1 - (1 - e^{-t/4000})) && \text{(exponential CDF)} \\
&= 1 - e^{-t/5000} e^{-t/8000} e^{-t/4000} \\
&= 1 - e^{-t(1/5000 + 1/8000 + 1/4000)} \\
&= 1 - e^{-0.000575t}
\end{aligned}
$$

- System failure time is also exponentially distributed, with parameter 0.000575
- Expected time to system failure is 1/0.000575 = 1739 hours

Poisson process: exponential arrival times

- A Poisson process is any process where independent events occur at a constant average rate
  - example: babies are born at a hospital at a rate of five per hour
- Time between each pair of consecutive events follows an exponential distribution with parameter λ (the *arrival rate*)
- Each of these inter-arrival times is assumed to be independent of other inter-arrival times
- The process is *memoryless*: number of arrivals in any bounded interval of time after time t is independent of the number of arrivals before t
- Good model for many types of phenomena:
  - number of road crashes in a zone
  - number of faulty items in a production batch
  - arrival of customers in a queue
  - occurrence of earthquakes

Simulating a Poisson process (1/2)

- Naïve approach: simulate “they occur at an average rate” by splitting our simulation time into small time intervals
- During each short interval, the probability of an event is about `rate × dt`
- Occurrence of events are independent of each other (memoryless property)
- We simulate the process and draw a vertical line for each non-zero time bin (where an event occurred)

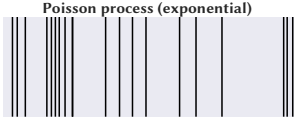
```
rate = 20.0  # average number of events per second
dt = 0.001  # time step
n = int(1.0/dt)  # number of time steps
x = numpy.zeros(n)
x[numpy.random.rand(n) <= rate*dt] = 1
plt.figure(figsize=(6,2))
plt.vlines(numpy.nonzero(x)[0], 0, 1)
plt.xticks([])
plt.yticks([])
```

Poisson process simulated with time bins

Simulating a Poisson process (2/2)

- Alternative approach (more efficient)
- We know that the time interval between two successive events is an exponential random variable
  - simulate inter-arrival times by sampling from an exponential random variable
  - simulate a Poisson process by summing the time intervals
- We draw a vertical line for each event

```
y = numpy.cumsum(numpy.random.exponential(1.0/rate, size=int(rate)))
plt.figure(figsize=(6,2))
plt.vlines(y, 0, 1)
plt.xticks([])
plt.yticks([])
```



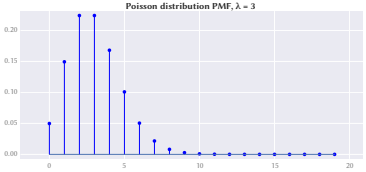
RISK ENGINEERING

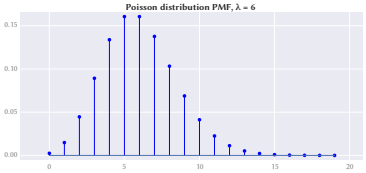
The Poisson distribution

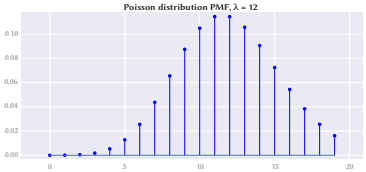
- The probability distribution of the *counting process* associated with a Poisson process
  - the number of events of the Poisson process over a time interval
- Probability mass function:

$$\Pr(Z = k) = \frac{\lambda^k e^{-\lambda}}{k!}, k = 0, 1, 2...$$

- the parameter λ is called the *intensity* of the Poisson distribution
  - increasing λ adds more probability to larger values
- SciPy: `scipy.stats.poisson`





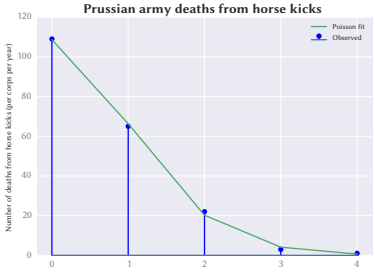


RISK ENGINEERING

Poisson distribution and Prussian horses

- Number of fatalities for the Prussian cavalry resulting from being kicked by a horse was recorded over a period of 20 years
  - for 10 army corps, so total number of observations is 200

Deaths	Occurrences
0	109
1	65
2	22
3	3
4	1
> 4	0



- It follows a Poisson distribution

- **Exercise**: reproduce the plot on the right which shows a fit between a Poisson distribution and the historical data

The Poisson distribution: properties

- Expected value of the Poisson distribution is equal to its parameter μ
- Variance of the Poisson distribution is equal to its parameter μ
- The sum of independent Poisson random variables is also Poisson
- Specifically, if Y_1 and Y_2 are independent with $Y_i \sim Poisson(\mu_i)$ for $i = 1, 2$ then $Y_1 + Y_2 \sim Poisson(\mu_1 + \mu_2)$
- **Exercise**: test these properties empirically with SciPy

Notation: ~ means "follows in distribution"

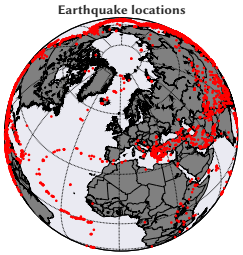
Simulating earthquake occurrences (1/2)

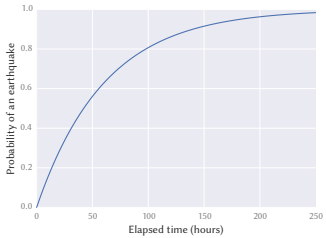
- Suppose we live in an area where there are typically 0.05 earthquakes of intensity 5 or more per year
- Assume earthquake arrival is a Poisson process
  - interval between earthquakes follows an exponential distribution
  - events are independent
- Simulate the random intervals between the next earthquakes of intensity 5 or greater
- What is the 25^{th} percentile of the interval between 5+ earthquakes?

```
> from scipy.stats import expon

> expon.rvs(scale=1/0.03, size=15)
array([ 23.23763551,  28.73209684,  29.7729332 ,  46.66320369,
         4.03328973,  84.03262547,  42.22440297,  14.14994806,
        29.90516283,  87.07194806,  11.25694683,  15.08286603,
        35.72159516,  44.70480237,  44.67294338])

> expon.ppf(0.25, scale=1.0/0.03)
9.5894024150593644
```

Simulating earthquake occurrences (2/2)

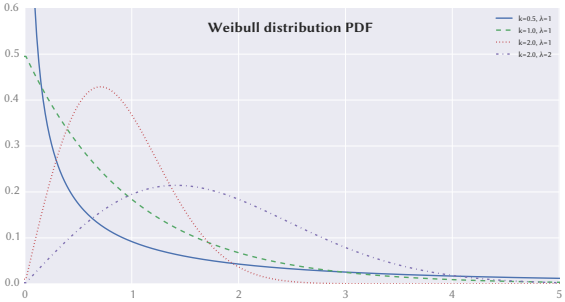
- Worldwide: 144 earthquakes of magnitude 6 or greater in 2013 (one every 60.8 hours on average)
- Rate: $\lambda = \frac{1}{60.8}$ per hour
- What's the probability that an earthquake of magnitude 6 or greater will occur (worldwide) in the next day?
  - right: plot of the CDF of the corresponding exponential distribution
  - `scipy.stats.expon.cdf(24, scale=60.8) = 0.326`





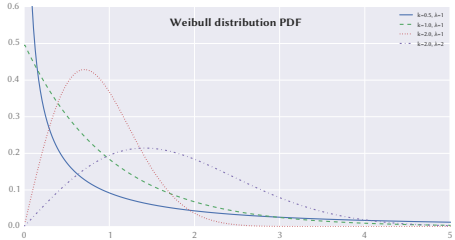
Data source: `http://earthquake.usgs.gov/earthquakes/search/`

Weibull distribution



- Very flexible distribution (able to model left-skewed, right-skewed, symmetric data)
- Widely used for modeling **reliability data**
- Python: `scipy.stats.dweibull(k, μ, λ)`

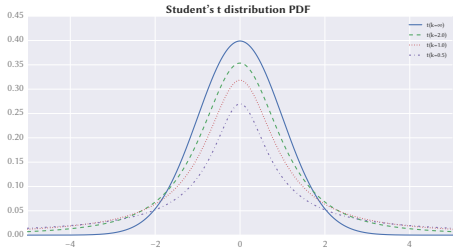
Weibull distribution



- $k < 1$: the failure rate **decreases over time**
  - significant “infant mortality”
  - defective items failing early and being weeded out of the population
- $k = 1$: the failure rate is **constant over time**
  - suggests random external events are causing failure
- $k > 1$: the failure rate **increases with time**
  - “aging” process causes parts to be more likely to fail as time goes on

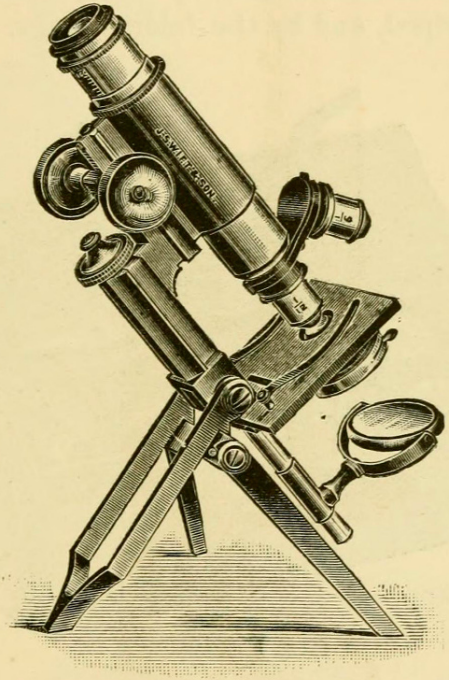
RISK ENGINEERING

Student's t distribution



- Symmetric and bell-shaped like the normal distribution, but with heavier tails
- As the number of degrees of freedom *df* grows, the t-distribution approaches the normal distribution with mean 0 and variance 1
- Python: `scipy.stats.t(df)`
- First used by W. S. Gosset (aka Mr Student, 1876-1937), for quality control at Guinness breweries

RISK ENGINEERING



Analyzing data

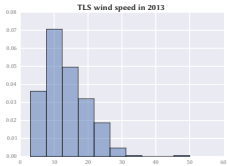
Curve fitting

- Generate 1000 random numbers from a normal distribution centered at 10, stddev of 2
- Find median of the sample
- Check that 50% quantile is equal to the median
- Find 5% quantile of the sample and compare with expected analytic value
- Fit a normal distribution to the sample and check that shape parameters are appropriate

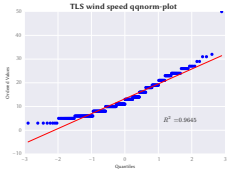
```
In [1]: import numpy, scipy.stats

In [2]: data = numpy.random.normal(loc=10, scale=2, size=1000)

In [3]: numpy.median(data)
Out[3]: 9.8519071069711401

In [4]: numpy.percentile(data, 50)
Out[4]: 9.8519071069711401

In [5]: numpy.percentile(data, 5)
Out[5]: 6.5815103766848306

In [6]: scipy.stats.norm(loc=10, scale=2).ppf(0.05)
Out[6]: 6.7102927460970543

In [7]: mu, sigma = scipy.stats.norm.fit(data)

In [8]: print(mu, sigma)
9.91652864354 1.99215863255
```

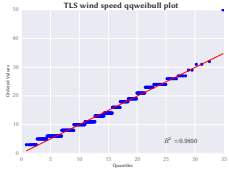
RISK ENGINEERING

Analyzing data: wind speed



- Import wind speed data for Toulouse airport
- Find the mean of the distribution
- Plot a histogram of the data
- Does the data seem to follow a normal distribution?
  - use a Q-Q plot to check
- Check whether a Weibull distribution fits better
- Predict the highest wind speed expected in a 10-year interval





Data downloaded from `wunderground.com/history/airport/LFBO/`

Analyze HadCRUT4 data on global temperature change

- HadCRUT4 is a gridded dataset of global historical surface temperature anomalies relative to a 1961-1990 reference period
- Data available from `metoffice.gov.uk/hadobs/hadcrut4/`
- **Exercise**: import and plot the northern hemisphere ensemble median time series data, including uncertainty intervals

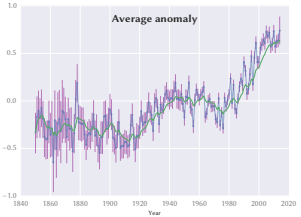


Image credits

- Dice on slide 39, `flic.kr/p/9SJ5g`, CC BY-NC-ND licence
- Galton board on slide 57: Wikimedia Commons, CC BY-SA licence
- Transistor on slide 61: `flic.kr/p/4d4XSj`, CC BY licence)
- Photo of Mr Gosset (aka Mr Student) on slide 72 from Wikimedia Commons, public domain
- Microscope on slide 73 adapted from `flic.kr/p/aeh1J5`, CC BY licence

For more free course materials on risk engineering, visit `risk-engineering.org`

RISK ENGINEERING

For more information

- SciPy lecture notes: `scipy-lectures.github.io`
- Book *Statistics done wrong*, available online at `statisticsdonewrong.com`
- A gallery of interesting Python notebooks: `github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks`

For more free course materials on risk engineering, visit `risk-engineering.org`

RISK ENGINEERING

Feedback welcome!

Was some of the content unclear? Which parts of the lecture were most useful to you? Your comments to `feedback@risk-engineering.org` (email) or `@LearnRiskEng` (Twitter) will help us to improve these course materials. Thanks!

`@LearnRiskEng`

`fb.me/RiskEngineering`

`google.com/+RiskengineeringOrgCourseware`

For more free course materials on risk engineering, visit `risk-engineering.org`