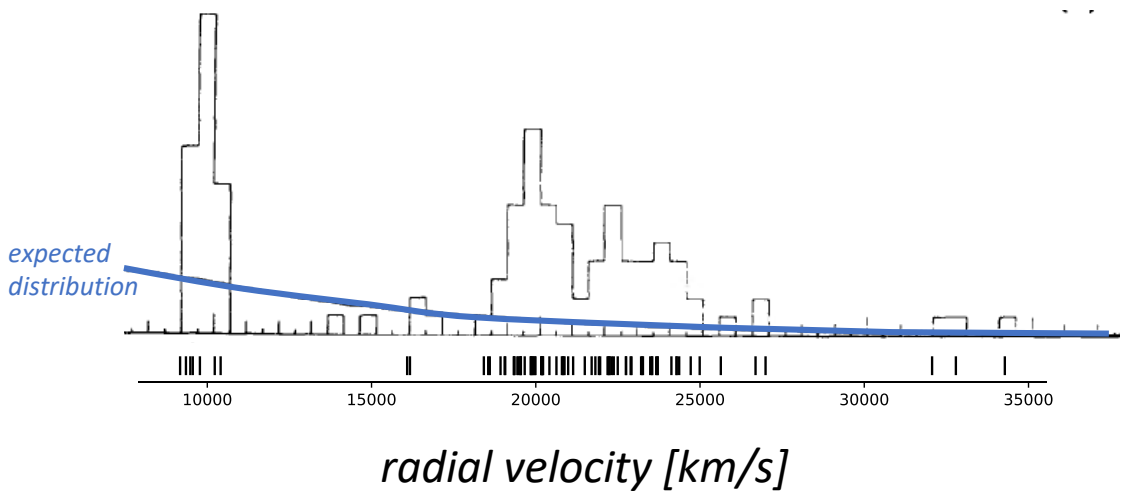
Clustering



This chart shows the distribution of the speeds of 82 galaxies, from a survey of the Corona Borealis region.

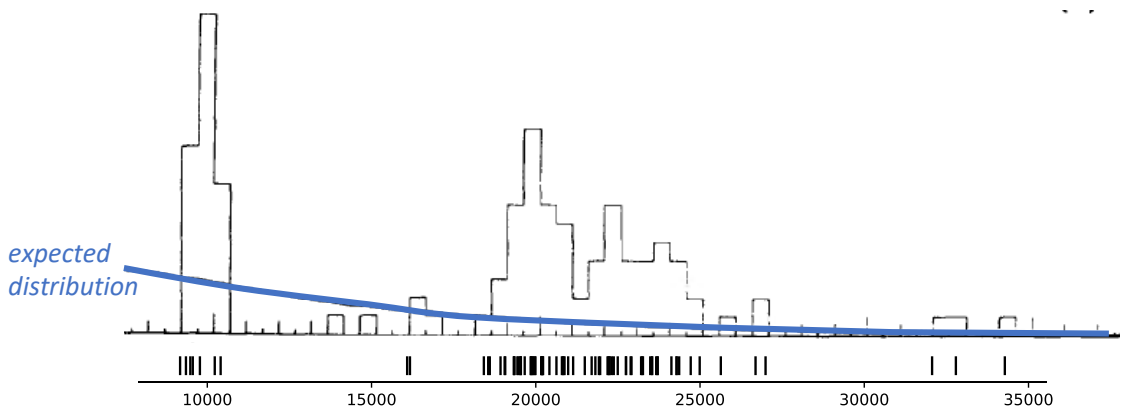
The smooth line shows the expected distribution for a uniform universe. The data however is clumpy, indicating voids and clusters of galaxies.

How can we pick out the clusters?

Since the early 1980s, multi-object spectrographs, CCD detectors, and some dedicated telescopes have allowed the mass production of galaxy redshifts. These large surveys have revealed a very surprising picture of the luminous matter in the Universe. Many astronomers had imagined roughly spherical galaxy clusters floating amongst randomly scattered field galaxies, like meatballs in sauce. Instead, they saw galaxies concentrated into enormous walls and streamers, surrounding huge voids that appear largely empty. The galaxy distribution has been compared to walls of soapy water, surrounding bubbles of air in a basinful of suds; linear filaments appear where the walls of different soap bubbles join, and rich clusters where three or more walls run into each other. A more accurate metaphor is that of a sponge; the voids are interlinked by low-density `holes' in the walls.

We must remember that in dynamical terms, this structure is very young. Galaxies are not highly concentrated into groups and clusters, in the way that luminous stars are largely confined within galaxies. Averaged over a rich cluster like Coma or Perseus, the density is only 10-100 times greater than the cosmic mean. So the gravitational forces binding a galaxy cluster or group together are relatively weak, and the time for an individual galaxy to move across the system is long. We saw in Section 6.5 that since the Big Bang, galaxies in the outskirts of the Coma cluster have not yet had time to travel to its core; in the Local Group, the large galaxies are falling together for the first time.

Clustering



radial velocity [km/s]

This chart shows the distribution of the speeds of 82 galaxies, from a survey of the Corona Borealis region.

The smooth line shows the expected distribution for a uniform universe. The data however is clumpy, indicating voids and clusters of galaxies.

How can we pick out the clusters?

```
def rx(p1,p2,p3, μ1,μ2,μ3, σ1,σ2,σ3):
    u = random.random()
    if u <= p1:
        μ,σ = μ1,σ1
    else if u <= p1+p2:
        μ,σ = μ2,σ2
    else:
        μ,σ = μ3,σ3
    return numpy.random.normal(loc=μ, scale=σ)
```

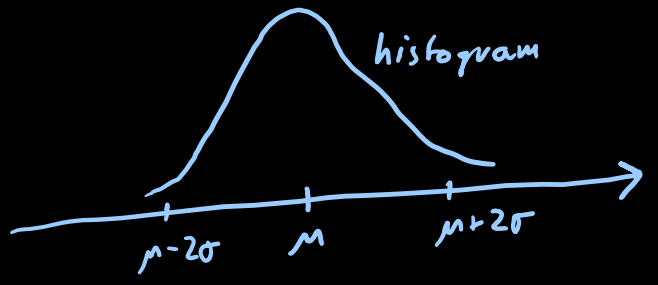
Method

1. Invent a probability model i.e. a simulator, parameterized by what we want to learn
2. Write out the likelihood
3. Maximize it

```
def rx(p1,p2,p3, μ1,μ2,μ3, σ1,σ2,σ3):
    u = random.random()
    if u <= p1:
        μ,σ = μ1,σ1
    else if u <= p1+p2:
        μ,σ = μ2,σ2
    else:
        μ,σ = μ3,σ3
    return numpy.random.normal(loc=μ, scale=σ)
```

Where did this simulator come from?

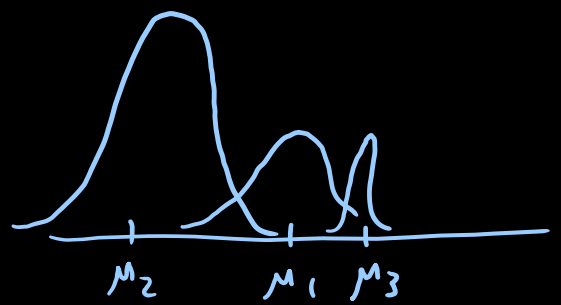
numpy.random.normal(loc=μ, scale=σ) generates values like



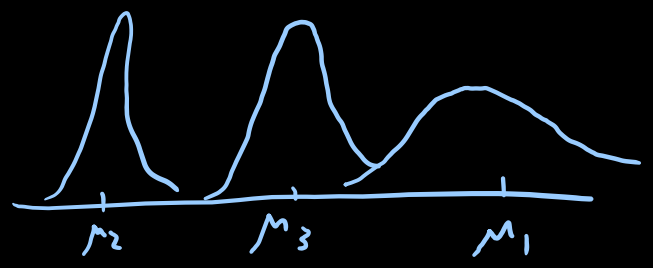
It seems fine for describing a single cluster.

It looks like the data might have three clusters, each with different μ and σ.

So, let's pick the μ, σ randomly from a set of choices.



if p_2 is large



if p_1, p_2, p_3 roughly equal

notes p.7

1.3, 1.4. Random variables in code, and in maths

tl;dr. A *random variable* is a function that can give different answers, e.g. a random number generator.

Two random variables are *independent* if knowing the value of one gives no information about the value of the other.

```
def ry(a,b):
    x = random.random()
    y = a * x + b
    return y
```

$X \sim U[0,1]$
$Y = aX + b$

Generate X from U[0,1]
Sample x from "

upper case – r.v.
lower case – constants

```
def ry():
    x1 = random.random()
    x2 = random.random()
    return x1 * math.log(x2)
```

$X_1, X_2 \sim U[0,1]$
$Y = X_1 \log X_2$

Generate x_1 and x_2 independently from U[0,1]

```
def ry():
    (x1,x2) = my_pair_rng()
    return x1 * math.log(x2)
```

$(X_1, X_2) \sim \ldots$
$Y = X_1 \log X_2$

Generate X_1 and X_2 (perhaps) not independently

```
(x1,x2) = [random.random()
           for _ in range(2)]
```

$X_1, X_2 \sim U[0,1]$

X_1, X_2 independent.

```
x1 = random.random()
x2 = 1 - random.random()
```

$X_1, X_2 \sim U[0,1]$
$X_1 \sim X_2$

X_1, X_2 are distributed as U[0,1]
X_1, X_2 are identically distributed

There are standard numerical random variables that you should know:

notes p.10

DISCRETE RANDOM VARIABLES

Uniform	$X \sim U\{a, \ldots, b\}$	An integer, uniformly distributed in $\{a, \ldots, b\}$.	`numpy.random.randint(a,b+1)`
Geometric	$X \sim \text{Geom}(p)$	Counts the number of failures before a success, where success happens with probability p	`numpy.random.geometric(p) - 1`
Binomial	$X \sim \text{Bin}(n, p)$	Counts the number of heads in n tosses of a biased coin, where p is the probability of heads	`numpy.random.binomial(n,p)`
Poisson	$X \sim \text{Pois}(\lambda)$	Used for modelling counts such as the number of buses that arrive in an interval of time.	`numpy.random.poisson(`λ`)`

CONTINUOUS RANDOM VARIABLES

Uniform	$X \sim U[a, b]$	A floating point value, uniformly distributed in the interval $[a, b]$.	`numpy.random.uniform(a,b)`
Exponential	$X \sim \text{Exp}(\lambda)$	Used for modelling lifetimes, e.g. the time until the next bus arrives.	`numpy.random.exponential(`λ`)`
Normal / Gaussian	$X \sim N(\mu, \sigma^2)$	Used to model magnitudes, e.g. the height of a person.	`numpy.random.normal(`μ`,`σ`)`
Beta	$X \sim \text{Beta}(a, b)$	Arises in Bayesian inference.	`numpy.random.beta(a,b)`

There are standard numerical random variables that you should know:

notes p.10

DISCRETE RANDOM VARIABLES

Uniform	$X \sim U\{a, \ldots, b\}$	An integer, uniformly distributed in $\{a, \ldots, b\}$.	`numpy.random.randint(a,b+1)`
Geometric	$X \sim \text{Geom}(p)$	Counts the number of failures before a success, where success happens with probability p	`numpy.random.geometric(p) - 1`
Binomial		ts the number of heads in n tosses of a d coin, where p is the probability of heads	`numpy.random.binomial(n,p)`
Poisson		for modelling counts such as the number of that arrive in an interval of time.	`numpy.random.poisson(λ)`

the return type is integer (or, more generally, a value from some finite / countable set)

CONTINUOUS RANDOM VARIABLES

Uniform	$X \sim U[a, b]$	A floating point value, uniformly distributed in the interval $[a, b]$.	`numpy.random.uniform(a,b)`
Exponential		for modelling lifetimes, e.g. the time until xt bus arrives.	`numpy.random.exponential(λ)`
Normal / Gaus		to model magnitudes, e.g. the height of a n.	`numpy.random.normal(μ,σ)`
Beta		s in Bayesian inference.	`numpy.random.beta(a,b)`

the return type is floating point (and there's also a smoothness condition…)

There are standard numerical random variables that you should know:

notes p.10

DISCRETE RANDOM VARIABLES

Uniform		distributed in $\{a, \ldots, b\}$.	`numpy.random.randint(a,b+1)`
Geometric		ailures before a success, with probability p	`numpy.random.geometric(p) - 1`
Binomial		eads in n tosses of a the probability of heads	`numpy.random.binomial(n,p)`
Poisson		unts such as the number of interval of time.	`numpy.random.poisson(λ)`

The Normal distribution is used very widely. It's easy to work with, and it's good as an approximation in very many settings.

Useful fact: if

$$X \sim N(\mu, \sigma^2)$$

then

$$aX + b \sim N(a\mu + b, a^2\sigma^2)$$

CONTINUOUS

Uniform		uniformly distributed in the interval $[a, b]$.	`numpy.random.uniform(a,b)`
Exponential	$\text{Exp}(\lambda)$	Used for modelling lifetimes, e.g. the time until the next bus arrives.	`numpy.random.exponential(λ)`
Normal / Gaussian	$X \sim N(\mu, \sigma^2)$	Used to model magnitudes, e.g. the height of a person.	`numpy.random.normal(μ,σ)`
Beta	$X \sim \text{Beta}(a, b)$	Arises in Bayesian inference.	`numpy.random.beta(a,b)`

Specifying numerical random variables

Any random variable X can be specified by its *probability distribution*,

$$\mathbb{P}(X \in A) \text{ for every set } A.$$

Numerical random variables can equivalently be specified by their *cumulative distribution function*,

$$F(x) = \mathbb{P}(X \leq x) \text{ for every } x.$$

If X is a discrete random variable (integer-valued), $F(x)$ is a step function,

$$F(x) = \mathbb{P}(X \leq x) = \sum_{y=-\infty}^{\text{floor}(x)} \mathbb{P}(X = y).$$

X is a continuous random variable if $F(x)$ is differentiable. If so, the *density function* for X is defined to be $f(x) = F'(x)$, and

$$F(x) = \mathbb{P}(X \leq x) = \mathbb{P}(X < x) = \int_{y=-\infty}^{x} f(y)\, dy.$$

For a continuous random variable X, $\mathbb{P}(X = x) = 0$ for all x.

So $\mathbb{P}(X \leq x)$
$= \mathbb{P}(X < x \text{ or } X = x)$
$= \mathbb{P}(X < x) + \mathbb{P}(X = x)$
$= \mathbb{P}(X < x)$.

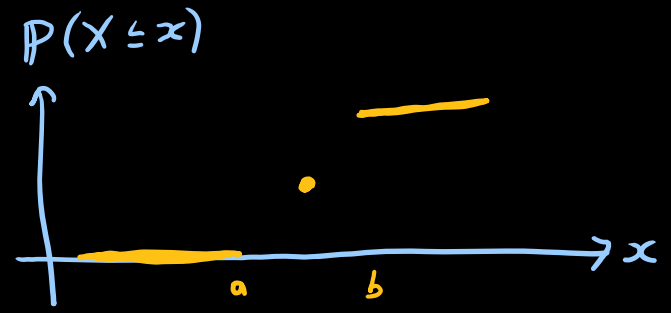
Exercise (The uniform distribution).

For X generated by `random.uniform(a,b)`, which generates a floating point number uniformly distributed in $[a, b]$, sketch $\mathbb{P}(X \leq x)$ and derive the density.

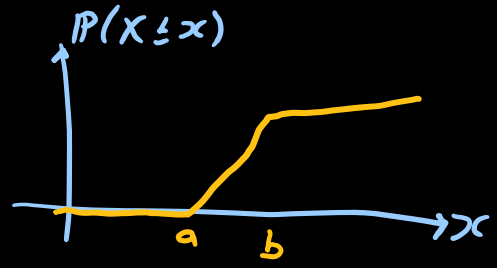
There's no general method for this. You just have to use common sense, and think through all the cases carefully.

Let's try some cases:

- If $x < a$ then $\mathbb{P}(X \leq x) = 0$ since values $< a$ are impossible
- If $x > b$ then $\mathbb{P}(X \leq x) = 1$ since all return values are $\leq b$
- If $x = \frac{a+b}{2}$ then $\mathbb{P}(X \leq x) = \frac{1}{2}$ since all values in $[a,b]$ are equally likely



Generalizing, we can spot that $\mathbb{P}(X \leq x) = \begin{cases} \frac{x-a}{b-a} & \text{for } x \in [a,b] \\ 0 & \text{for } x < a \\ 1 & \text{for } x > b \end{cases}$



The density is $f(x) = \frac{d}{dx}\mathbb{P}(X \leq x) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a,b] \\ 0 & \text{else} \end{cases}$

Exercise (The uniform distribution).

For X generated by `random.uniform(a,b)`, which generates a floating point number uniformly distributed in $[a, b]$, sketch $\mathbb{P}(X \leq x)$ and derive the density.

We derived the density

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b] \\ 0 & \text{otherwise.} \end{cases}$$

This is commonly written with an <u>indicator function</u>

$$f(x) = \frac{1}{b-a} 1_{x \in [a,b]}$$

The indicator function maps boolean to integer: $1_{true} = 1$, $1_{false} = 0$.

We'll use a unified density notation:

$$\Pr_X(x) = \begin{cases} \mathbb{P}(X = x) & \text{if } X \text{ is a discrete r.v.} \\ f(x) & \text{if } X \text{ is continuous, density } f \end{cases}$$

With this notation, it's cleaner to write the various rules...

In lecture 1, we said "maximize the probability of the observed data". But for continuous random variables, $\mathbb{P}(\text{observed data} \mid \theta) = 0$ for any θ, so we need to use Pr not $\mathbb{P}$.

Maximum likelihood estimation.

The maximum likelihood estimator for an unknown parameter θ is that value which maximizes

$$\Pr(\text{observed data} \mid \theta)$$

Joint density and independence

notes p.12

Two random variables X and Y have a *joint density*, written $\Pr_{X,Y}(x, y)$.

For discrete random variables, it is defined as

$$\Pr_{X,Y}(x, y) = \mathbb{P}(X = x \text{ and } Y = y)$$

For continuous random variables, … (it's trickier — left to a later lecture)

Independence. Two random variables X and Y are independent if

$$\Pr_{X,Y}(x, y) = \Pr_X(x) \Pr_Y(y) \quad \text{for all } x \text{ and } y$$

or equivalently if

$$\mathbb{P}(X \in A \text{ and } Y \in B) = \mathbb{P}(X \in A)\, \mathbb{P}(Y \in B) \quad \text{for all sets } A \text{ and } B$$

Handling numerical random variables

notes p.41

- To fit a model (i.e. to learn its parameters from data), we need the likelihood, which needs $\Pr_X(x)$
- If the random variable is continuous, then $\Pr_X(x)$ is defined to be the derivative of $\mathbb{P}(X \leq x)$, so we should work out $\mathbb{P}(X \leq x)$ first.

Exercise 3.3 (Transforming random variables).
Let $X \sim U[0,1]$ *and let* $Y = X^2$*. Find the density of* Y*.*

We want a formula for $F(y) = \mathbb{P}(Y \leq y)$.

If $y < 0$: $\mathbb{P}(Y \leq y) = 0$.
If $y > 1$: $\mathbb{P}(Y \leq y) = 1$
} by common sense

If $y \in [0,1]$: $\mathbb{P}(Y \leq y) = \mathbb{P}(X^2 \leq y) = \mathbb{P}(X \leq \sqrt{y})$ ← good strategy: rewrite the probability you want to work out in terms of simpler random variables, e.g. the standard random variables listed earlier

$= \frac{\sqrt{y} - 0}{1 - 0}$ ← we've already worked out the c.d.f. for a Uniform $[a, b]$ random variable

$= \sqrt{y}$

So the density is $\Pr_Y(y) = \frac{d}{dy}\mathbb{P}(Y \leq y) = \frac{1}{2} y^{-\frac{1}{2}} 1_{y \in [0,1]}$

notes p.13

1.5. Learning generative models

tl;dr. Given a dataset $x_1, \ldots, x_n$ can we design a random number generator (i.e. a probability model) that might have generated it?

1. Choose a distribution with tuneable parameters, call it X. We want $x_1, \ldots, x_n$ to look like independent samples from X.
2. Write out the likelihood $\text{lik}(\theta|x_1, \ldots, x_n) = \Pr_X(x_1|\theta) \times \cdots \times \Pr_X(x_n|\theta)$
3. Fit the model—i.e. find the maximum likelihood estimator for θ.

The likelihood is the Pr of the observed data:

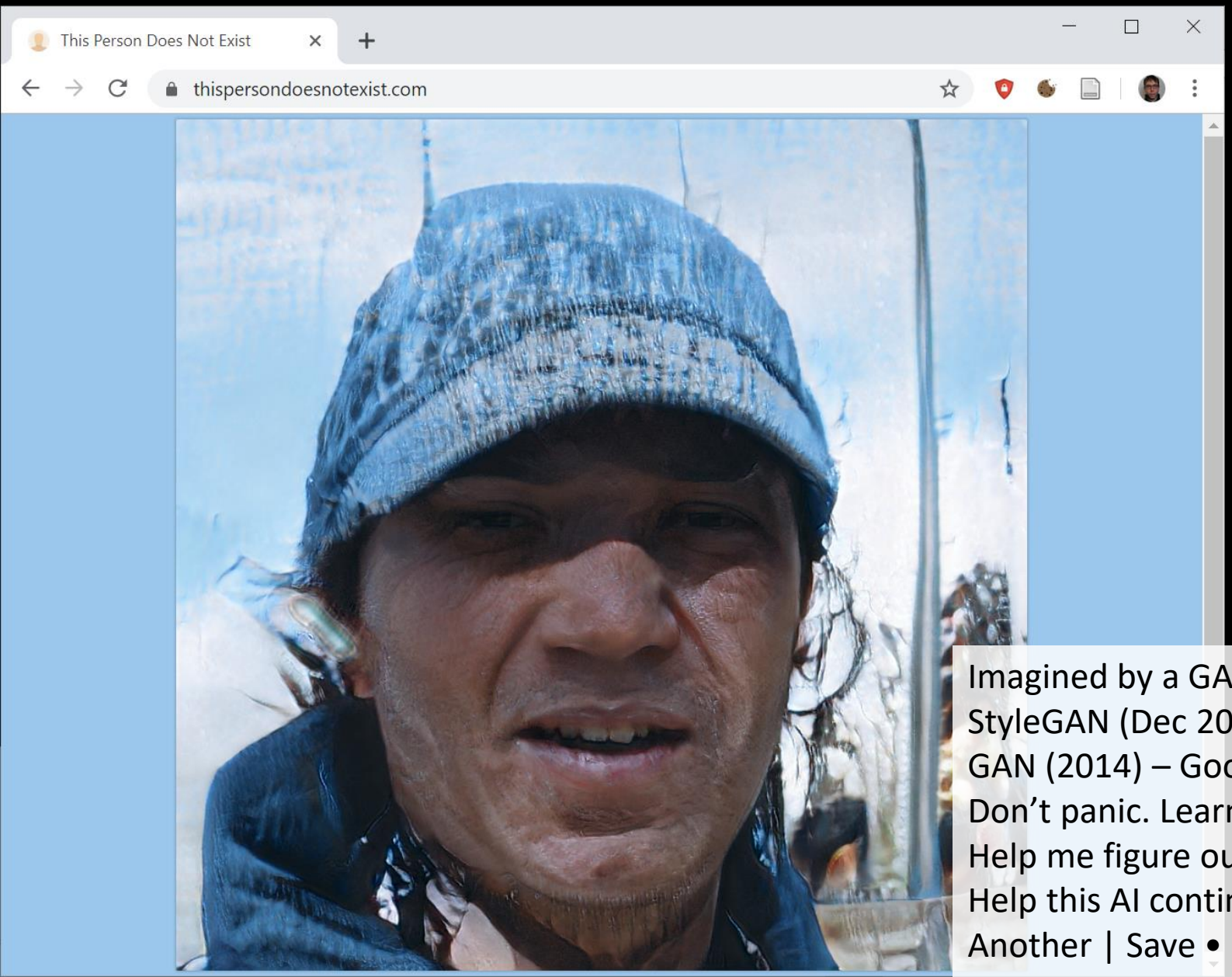
$\text{lik}(\theta|x_1 \cdots x_n)$

$= \Pr(x_1, \cdots, x_n|\theta)$

$= \Pr(x_1|\theta) \times \cdots \times \Pr(x_n|\theta)$

by independence.

https://thispersondoesnotexist.com/



Imagined by a GAN (generative adversarial network)
StyleGAN (Dec 2018) – Karras et al. and Nvidia Original GAN (2014) – Goodfellow et al.
Don’t panic. Learn about how it works.
Help me figure out what was learned here.
Help this AI continue to dream.
Another | Save • Cats | Articles | TV Friends - Office

Exercise 1.7 (coin tosses).

We take a biased coin and toss it $n = 10$ times, and observe the outcomes

$$(x_1, \ldots, x_{10}) = (H, H, t, t, H, H, t, t, H, H).$$

Fit the probability model

$$X = \begin{cases} H & \text{with probability } p \\ t & \text{with probability } 1 - p \end{cases}$$

For a single observation: $Pr_x(x|p) = \begin{cases} p & \text{if } x = H \\ 1-p & \text{if } x = t \end{cases}$

So $lik(p|x_1, \ldots, x_n) = Pr(x_1, \ldots, x_n|p) = Pr_x(x_1|p) \times \cdots \times Pr_x(x_n|p) = \prod_{i=1}^{n} Pr_x(x_i|p) = \prod_{i=1}^{n} \begin{cases} p & \text{if } x_i = H \\ 1-p & \text{if } x_i = t \end{cases}$

$= p^y (1-p)^{n-y}$ where $y = \#\text{heads}$.

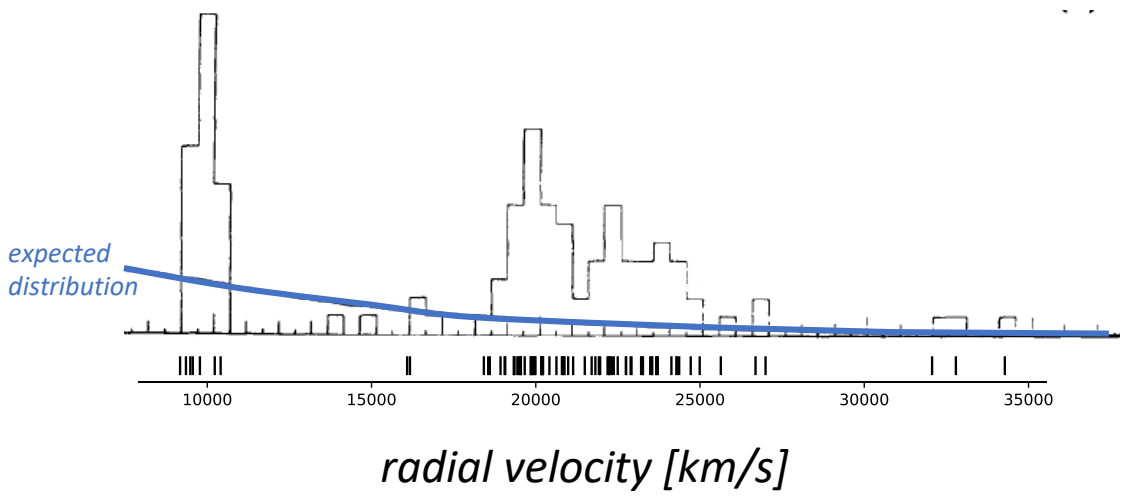
$\Rightarrow \quad \log lik(p|x_1, \ldots, x_n) = y \log p + (n-y) \log(1-p).$

To fit the model, ie find the maximum likelihood estimator $\hat{p}$,

$$\frac{d}{dp} \log lik = \frac{y}{p} - \frac{n-y}{1-p} = 0 \quad \Rightarrow \quad \hat{p} = \frac{y}{n} = 0.6 \quad \text{for this dataset.}$$

Clustering

notes p.14



This chart shows the distribution of the speeds of 82 galaxies, from a survey of the Corona Borealis region.

The smooth line shows the expected distribution for a uniform universe. The data however is clumpy, indicating voids and clusters of galaxies.

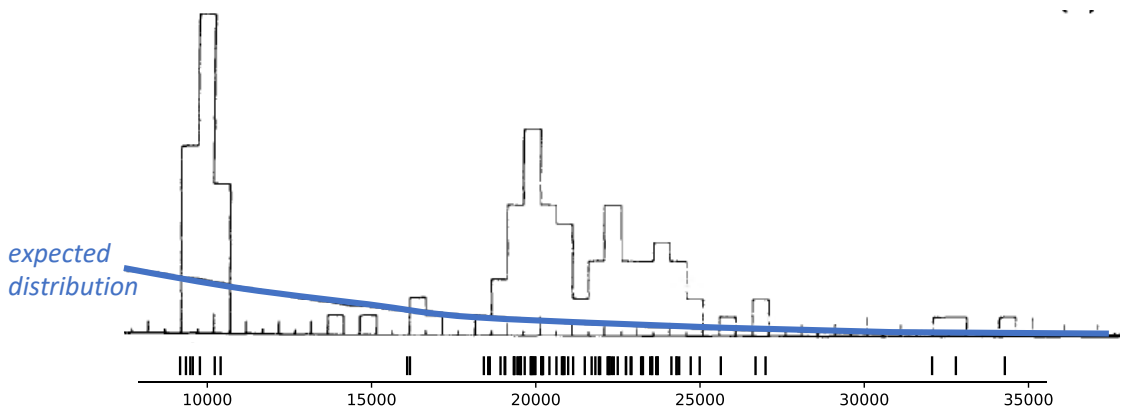
How can we pick out the clusters?

Method

1. Invent a probability model i.e. a simulator, parameterized by what we want to learn
2. Write out the likelihood
3. Maximize it

Clustering

notes p.14



radial velocity [km/s]

This chart shows the distribution of the speeds of 82 galaxies, from a survey of the Corona Borealis region.

The smooth line shows the expected distribution for a uniform universe. The data however is clumpy, indicating voids and clusters of galaxies.

How can we pick out the clusters?

```
def rx(p₁,p₂,p₃, μ₁,μ₂,μ₃, σ₁,σ₂,σ₃):
    u = random.random()
    if u <= p₁:
        μ,σ = μ₁,σ₁
    else if u <= p₁+p₂:
        μ,σ = μ₂,σ₂
    else:
        μ,σ = μ₃,σ₃
    return numpy.random.normal(loc=μ, scale=σ)
```

In maths notation,

$$K = \begin{cases} 1 & \text{with probability } p_1 \\ 2 & \text{with probability } p_2 \\ 3 & \text{with probability } p_3 \end{cases}$$

$$X \sim N(\mu_K, \sigma_K^2)$$

1. Invent a probability model, i.e. a simulator, parameterized by what we want to learn

$$K = \begin{cases} 1 & \text{with probability } p_1 \\ 2 & \text{with probability } p_2 \\ 3 & \text{with probability } p_3 \end{cases}$$

$$X \sim N(\mu_K, \sigma_K^2)$$

2. Write out the likelihood

the observed data

$$= \Pr(x_1, \ldots, x_n \mid \text{parameters})$$

$$= \prod_{i=1}^{n} \Pr(x_i \mid \text{params}) \quad \text{by independence}$$

Figure out $\Pr$ by calculating $\frac{d}{dx} \mathbb{P}(X \le x)$ (see examplesheet 1)

3. Maximize it

Use numerical optimization (scipy.optimize.fmin) to maximize over the nine unknown parameters

$\mu_1, \mu_2, \mu_3 \in \mathbb{R}$

$\sigma_1, \sigma_2, \sigma_3 > 0$ so use the $\sigma = e^\tau$ trick and optimize over $\tau \in \mathbb{R}$

$p_1, p_2, p_3 \in [0,1]$, $p_1 + p_2 + p_3 = 1$, so use the softmax trick from Lecture 1 / exercise 1.5

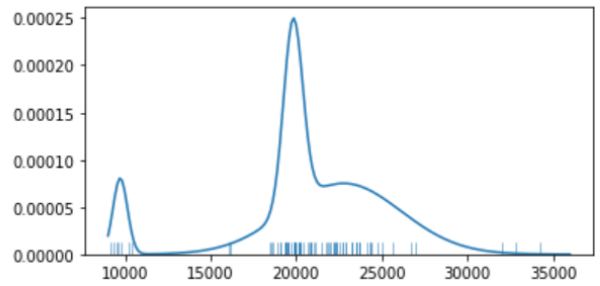
```
import scipy.stats
Pr = scipy.stats.norm.pdf

def loglik(θ, x):
    ξ1,ξ2,μ1,μ2,μ3,τ1,τ2,τ3 = θ
    p = numpy.exp([ξ1,ξ2,0])
    [p1,p2,p3] = p / sum(p)
    σ1,σ2,σ3 = numpy.exp([τ1,τ2,τ3])
    lik = p1*Pr(x,loc=μ1, scale=σ1) + p2*Pr(x,loc=μ2,scale=σ2) + p3*Pr(x,loc=μ3,scale=σ3)
    return numpy.log(lik)
```

Watch out for sensitivity on the initial guess! Later on in the course, we'll learn strategies for reasoning about confidence in the answer.

```
initial_guess = [-1,-1, 10000,20000,24000, math.log(1000),math.log(5000),math.log(8000)]
θhat = scipy.optimize.fmin(lambda θ: -numpy.sum(loglik(θ,galaxies)), initial_guess, maxiter=5000)
```

```
Optimization terminated successfully.
         Current function value: 776.169426
         Iterations: 1360
         Function evaluations: 2008
```

```
fig,ax = plt.subplots(figsize=(6,3))
x = numpy.linspace(9000,36000,200)
f = numpy.exp(loglik(θhat, x))
ax.plot(x, f)
for x in galaxies:
    ax.plot([x,x],[0,.05*max(f)], color='#1f77b4', lw=0.5)
ax.set_ylim(bottom=0)
plt.show()
```



Code snippet on Azure Notebooks.

Implementation note:
my loglik(θ|x) function returns a list $[\text{loglik}(\theta|x_1), \ldots, \text{loglik}(\theta|x_n)]$

and the sum is done later.

That's handy when it comes to plotting the $\Pr(x|\theta)$ function — my loglik/logPr function is already vectorized.