Computational Statistics with Python

Topic 5: Basic MCMC methods

Expected lecture time: 4 hours

Giancarlo Manzi

Department of Economics, Management and Quantitative Methods

University of Milan, Milan, Italy

Using mixture models

- A finite mixture of distributions is defined by a random variable X having the following density:

$$X \sim f_j, \text{with probability } p_j,$$

$(j : 1, 2, \ldots, k)$, with overall density given by:

$$X \sim p_1 f_1(x) + p_2 f_2(x) + \ldots p_k f_k(x).$$

- For an iid sample $(X_1, \cdots, X_n)$, for which $X_i \sim p_1 f_1(x) + p_2 f_2(x) + \ldots p_k f_k(x), \forall i$, the joint sample density is:

$$\prod_{i=1}^{n} \{p_1 f_1(x) + p_2 f_2(x) + \ldots p_k f_k(x)\},$$

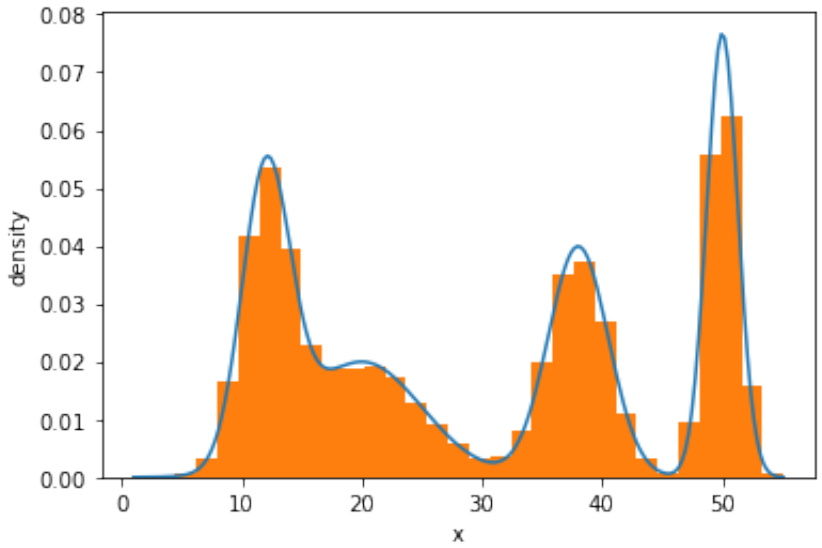
which could be computationally unfeasible, as we have here k^n elementary terms.

- Therefore we need simulation here.

In [2]:

```
import numpy as np
import numpy.random
import scipy.stats as ss
import matplotlib.pyplot as plt

def mixture1():
    np.random.seed(196)
    #import pdb; pdb.set_trace()
    # Parameters of the 4 normals forming the mixture [mean, sd
    normal_parameters = np.array([[12, 2],[20, 5], [38, 2.5], [
    #shape gives the number of elements in an array. In this ca
    n_components = normal_parameters.shape[0]
    #Simulated sample size
    Nsim = 50000
    # Weight of each normal, in this case all of them are 1/4
    # np.ones creates arrays of 1s and therefore in this case w
    #[0.25, 0.25, 0.25, 0.25]
    weights = np.ones(n_components, dtype = np.float64) / 4
    # A stream of indices from which to choose the component
    # With numpy.random.choice
    # The size of mixture_idx is 50000. It contains numbers bet
    # of numpy.random.choice is 4, meaning indexes from 0 to 3.
    #index in this case always equal to 0.25
    mixture_idx = numpy.random.choice(len(weights), size=Nsim,
    # y is the mixture sample
    # ss is the alias for scipy.stats and ss.norm.rvs generates
    # R
    y = numpy.fromiter((ss.norm.rvs(*(normal_parameters[i])) fo
                        dtype=np.float64)

    # Theoretical PDF plotting -- generate the x and y plotting
    xs = np.linspace(y.min(), y.max(), 200)
    ys = np.zeros_like(xs)

    for (l, s), w in zip(normal_parameters, weights):
        ys += ss.norm.pdf(xs, loc=l, scale=s) * w

    plt.plot(xs, ys)
    plt.hist(y, density=True, bins="fd")
    plt.xlabel("x")
    plt.ylabel("density")
    plt.show()

```

In [3]:
```
1 mixture1()
```



Another representation for mixtures exploiting conditional distributions and marginality

- The key idea here is to use the following representation for a pdf $f(x)$ of a continuous r.v.:

$$f(x) = \int_y h(x, y)dy = \int_y g(x|y)p(y)dy,$$

where g and p are distributions from which values can be drawn easily.

- Then, to obtain a value from the r.v. X, first generate a variable Y (if it is easy) from $p(y)$ and then generate X from $g(x|y)$ (if it is easy).

Example (from Robert & Casella, 2010, p. 50 - cont'd)

- **The Negative Binomial distribution**.
- The Negative Binomial random variable is used for count data (number of calls to a mobile phone in a given period of time, number of car rents per hour, number of bike rents in a day in a bike sharing system, etc.).
- It is a mixture representation of a Poisson (which is used for count data too) and a Gamma. Let's see how.
- The probability mass function (pmf) of a Negative binomial r.v. X with parameters n and p is:

$$P(X = x) = \binom{x + n - 1}{x}(1 - p)^n p^x, x = 0, 1$$

- The *Poisson distribution*.
- The pmf is:

$$P(X = x) = \frac{y^x e^{-y}}{x!}, x = 0, 1, 2, \ldots,$$

with expected value $E[X] = y$.

- The *Gamma distribution*.

- The pdf of a Gamma r.v. Y with parameters $\alpha = n$ and $\beta = \frac{1-p}{p}$ is:

$$f(y) = \frac{[\frac{1-p}{p}]^n}{\Gamma[n]} y^{n-1} e^{-y\frac{1-p}{p}}, y > 0,$$

where Γ is the *gamma function*.

Example (from Robert & Casella, 2010, p. 50 - cont'd)

- Let's now consider the following integral:

$$\int_0^\infty \underbrace{\frac{y^x e^{-y}}{x!}}_{X|y\sim\text{Poiss}(y)} \underbrace{\frac{y^{n-1}\, e^{-y\frac{1-p}{p}}}{\Gamma[n](\frac{p}{1-p})^{-n}}}_{Y\sim\text{Gamma}(n,\frac{1-p}{p})} dy.$$

- This integral is therefore equal to:

$$\frac{(1-p)^n p^{-n}}{x!\Gamma[n]} \int_0^\infty y^{n+x-1} e^{-\frac{y}{p}}\, dy. \quad (2)$$

Example (from Robert & Casella, 2010, p. 50 - cont'd)

- With the substitution $z = \frac{y}{p}$ and the Jacobian $J : y \to z = p$, integral (2) becomes:

$$\begin{aligned}
&\frac{(1-p)^n p^{-n}}{x!\Gamma[n]} \int_0^\infty (pz)^{n+x-1} e^{-z} p dz \\
&= \frac{(1-p)^n p^{-n}}{x!\Gamma[n]} p^{n+x} \int_0^\infty z^{n+x-1} e^{-z} dz \\
&= \frac{(1-p)^n p^{-n}}{x!\Gamma[n]} p^{n+x} \Gamma[n+x] \\
&= \frac{\Gamma[n+x]}{x!\Gamma[n]} p^x (1-p)^n \\
&= \frac{(n+x-1)!}{x!(n-1)!} p^x (1-p)^n \\
&= \binom{x+n-1}{x} p^x (1-p)^n,
\end{aligned}$$

which is the density of a negative binomial.

- Therefore, integral (1), being the expression of the density of a negative binomial, gives the mixture representation.

Example (from Robert & Casella, 2010, p. 50 - cont'd)

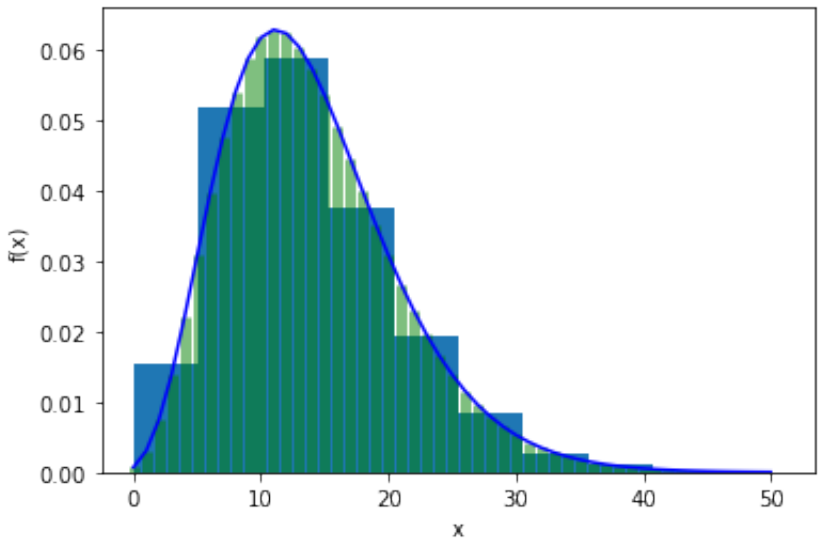
- Therefore, being integral (1) the density of the Negative binomial r.v. X , and given that X can be generated from a Poisson with parameter equal to y , the generated value from the Gamma, the algorithm proposed in Robert & Casella, 2010, p. 50 is justified:

1. Draw a value y from a $\text{Gamma}(n, \frac{1-p}{p})$.
2. Draw a value x from a $\text{Poisson}(y)$.

- A Python code for a Negative binomial with parameters $n = 6$ and $p = 0.3$ is the following, corresponding (more or less, please adjust it!) to the following R code presented in Robert & Casella (2010), p. 50:

In [4]:
```
import numpy as np
import scipy.stats as ss
import matplotlib.pyplot as plt
def sim_nbinom():
    NSim = 10**4
    n = 6
    p = .3
    y = np.random.gamma(n, scale=(1-p)/p, size=NSim)
    x1 = np.random.poisson(y, size=NSim)
    x = np.linspace(0, 50)
    plt.hist(x1, density = True)
    plt.xlabel("x")
    plt.ylabel("f(x)")
    x = [i for i in range(51)]
    plt.plot(x, ss.nbinom.pmf(x, n, p), 'blue', ms=8, label='nb
    plt.vlines(x, 0, ss.nbinom.pmf(x, n, p), colors='g', lw=5,
    plt.show()
```

In [5]:
```
sim_nbinom()
```



Another example: sampling from a Student's t (from Robert & Casella, 2010, p. 50)

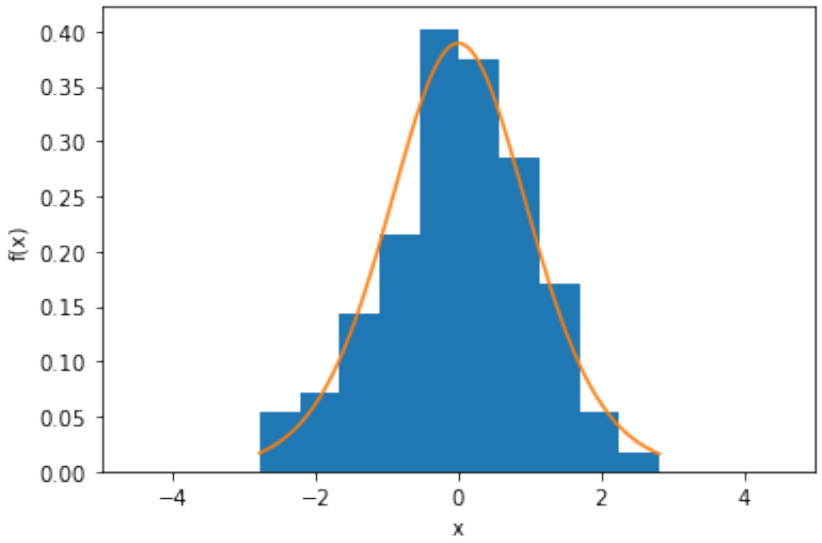
- In mathematical statistics, relationships between normal, chi squared, gamma and Student's t distributions are important results.

- For example, one of these results states that if $X|\theta \sim N(0, \theta^{-1})$ and $\theta \sim Gamma(\alpha, \beta)$, then the unconditional distribution of X has a Student's t distribution with 2α degrees of freedom.

- Another important result (related to the previous one) links the Student's t with the Normal and the chi-squared distributions: if $X|y \sim N(0, \nu/y)$ and $Y \sim \chi^2_\nu$, then $X \sim t_\nu$.

In [8]:
```
# Python code:
import numpy as np
from scipy.stats import t
def sim_t():
    NSim = 200
    nu = 10
    y = np.random.chisquare(nu, size=NSim)
    x1 = np.random.normal (0, (nu/y)**(0.5), size = NSim)
    rv = t(df=nu, loc=0, scale=1)
    xs = np.linspace(x1.min(), x1.max(), 200)
    #xs = np.linspace(rv.ppf(0.0001), rv.ppf(0.9999), NSim)
    y = rv.pdf(xs)

    plt.hist(x1, density = True)
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.xlim(-5,5)
    plt.plot(xs,y)
    plt.show


```

In [9]:
```
sim_t()
```



Use of Known Statistical Theories: normal and chi-square distribution

1. $N(0,1) \longrightarrow \chi^2_1$

   - generate $z \sim N(0,1)$
   - $x = z^2$ will be a χ^2_1 r.v.

2. $\chi^2_1 \longrightarrow N(0,1)$

   - generate $x \sim \chi^2_1$
   - generate $u \sim U(0,1)$
   - set $z = [\text{sign}(u - \frac{1}{2})]\sqrt{x}$, then $z \sim N(0,1)$.

Generate Normal from Uniforms

- Generate $u_1, \cdots, u_n \sim U(0,1)$.
  By CLT,

$$z = \frac{\bar{u} - 0.5}{\sqrt{1/(12n)}} \approx N(0,1) \text{ for sufficiently large } n.$$

- It is not bad for small n since $U(0,1)$ is symmetric. E.g., $n = 12$,
  $z = \sum_{i=1}^{12} u_i - 6$.

A few words turned famous after two centuries...

- PROBLEM.
- *Given the number of times in which an unknown event has happened and failed: Required the chance that the probability of its happening in a single trial lies somewhere*

between any two degrees of probability that can be named.

- In modern language the problem might be translated in the following way: suppose we have a binomial r.v. depending on a 'failure' probability θ, and we observe r failures out of n trials, what is the chance that θ lies between, say θ_1 and θ_2 ?

Bayes' rule for two events

- Given two events A and B we know that: $P(A|B) = \frac{P(A \cap B)}{P(B)}$, provided that $P(B) \neq 0$.

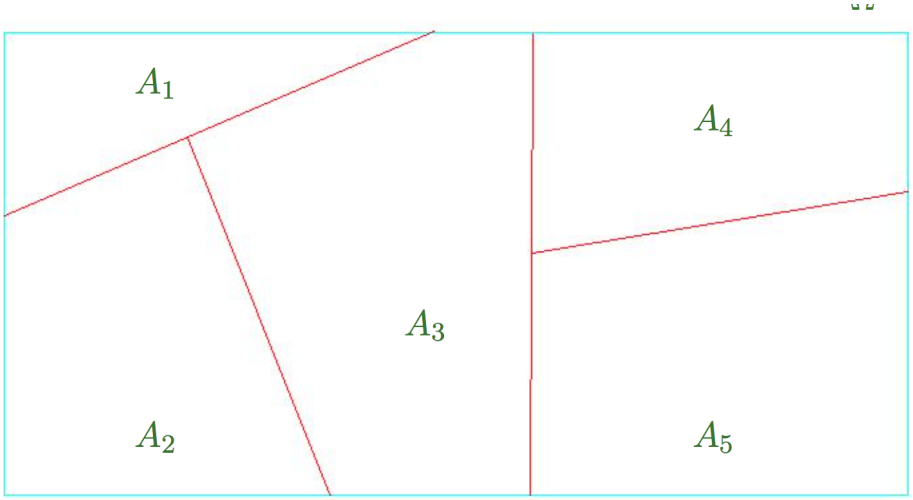
- From this, we easily get:
$$P(B)P(A|B) = P(A \cap B).$$

- Sometimes it is useful to obtain the probability $P(A|B)$ given $P(A)$, $P(B)$ e $P(B|A)$. We can write $P(A|B)$ as follows:
$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B \cap A)}{P(B)}$$
$$= \frac{P(A)}{P(A)} \frac{P(B \cap A)}{P(B)} = \frac{P(A)}{P(B)}$$
$$= \frac{P(A)P(B|A)}{P(B)}.$$
This is the simplest expression of the *Bayes' rule* for two events.

Towards the definition of the Bayes'rule for more than two events: Partition of a set

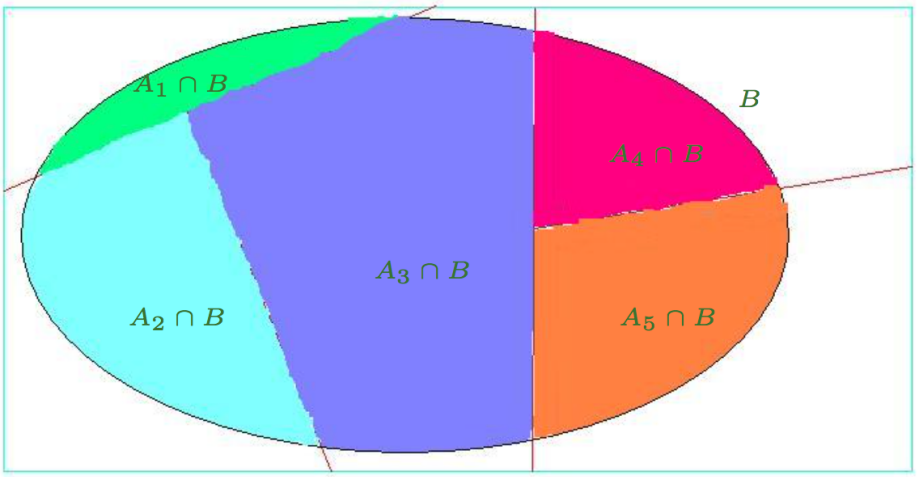
- Subsets $A_1, A_2, \ldots$ form a **partition** of a set Ω if:
  - $A_i \cap A_j = \oslash, \forall i, j, i \neq j$;
  - $A_1 \cup A_2 \cup \ldots = \Omega$.

- The following figure shows the case of a 5-subset partition of Ω.



Towards the definition of the Bayes'rule for more than two events: Partition of a set (cont'd)

- Any set B over a partition can be expressed as a union of disjoint intersections formed with the sets forming the partition.The coloured areas are these intersections:



Extension of Bayes' rule to n disjoint events

- Let's now have $A_1, A_2, \ldots, A_n$. Let these events be such that:
  - $\bigcup_{i=1}^{n} A_i = \Omega$;
  - $\forall A_i, A_j \quad (i \neq j)$: $A_i \cap A_j = \oslash$;
  - Therefore, events $A_1, A_2, \ldots, A_n$ form a partition of Ω.

- Then, for an event B and any event A_j forming the partition, we have the general form of Bayes' rule:

$$
\begin{aligned}
P(A_j|B) &= \frac{P(B \cap A_j)}{P(B)} \\
&= \frac{P(B|A_j)P(A_j)}{P\{(B \cap A_1) \cup (B \cap A_2) \cup \ldots \cup (B \cap A_n)\}} \\
&= \frac{P(B|A_j)P(A_j)}{P(B \cap A_1) + P(B \cap A_2) + \ldots + P(B \cap A_n)} \\
&= \frac{P(B|A_j)P(A_j)}{P(B|A_1)P(A_1) + P(B|A_2)P(A_2) + \ldots + P(B|A_n)} \\
&= \frac{P(B|A_j)P(A_j)}{\sum_{j=1}^{n} P(B|A_j)P(A_j)}.
\end{aligned}
$$

- $P(A_j|B)$ is the **posterior probability** of A_j, $P(A_j)$ is the **prior probability** of A_j.

Bayes' rule: example 1

- Let A_1 and A_2 be two urns containing red (R) and blue (B) balls:

URN	NO. OF RED BALLS	NO. OF BLUE BALLS
A_1	3	2
A_2	4	7

- The prior probabilities of A_1 and A_2 are $P(A_1) = P(A_2) = 1/2$.

- The probability of drawing a blue ball is $P(B) = P(A_1 \cap B) + P(A_2 \cap B) = P(A_1)P(B|A_1) +$

- What is the probability of urn A_2 given B?

- From the table we can say that $P(B|A_1) = 2/5$ and $P(B|A_2) = 7/11$.

- Then, with the Bayes' rule we get:

$$P(A_2|B) = \frac{P(A_2)}{P(B)} P(B|A_2) = \frac{\frac{1}{2}}{(\frac{1}{2} \times \frac{2}{5}) + (\frac{1}{2} \times \frac{7}{11})} \frac{7}{11} = \frac{35}{57}$$

First comments on Bayes' rule

- In the previous example, $P(A_2)$ has increased from 1/2 to 35/57 as soon as we have been informed that a blue ball has been drawn.
- We knew that the probability of B in urn A_2 is greater than the probability of B in urn A_1.
- Therefore, the fact that a blue ball has been drawn increases the *chances* of urn A_2 compared with urn A_1.
- $P(A_2)$ is the prior probability of A_2 (also called the marginal probability of A_2), $P(A_2|B)$ the posterior probability of A_2.

First comments on Bayes' rule (cont'd)

- The prior probability comes from a subjective judgement or previous experience and expresses uncertainty about the event A_2.
- The posterior probability is the same probability modified with empirical evidence given in this case by $P(B|A_2)$.
- We will see later that $P(B|A_2)$ corresponds to the likelihood function when Bayes' rule is applied in a statistical modelling context.
- $P(B)$ is the marginal probability of B and acts as a normalizing constant to ensure the value of $P(A_2|B)$ is a valid probability, i.e. a number between 0 and 1. It may be written as $P(A_1)P(B|A_1) + P(A_2)P(B|A_2)$, a process sometimes known as 'extending the conversation'.

Bayes' rule: example

2

- Assume that the number of men and the number of women is the same, but that 10% of men and 20% of women are *natural left-handed*.
- We want to calculate the probability that, having been extracted a left-handed person, it is a man, using Bayes' rule.
- The probability of a man is $P(M)=0.5, the probability of a woman is P(W)=0.5.
- The probability of being left-handed given that the person is a man is $P(LH|M) = 0.10$.
- The probability of being left-handed given that the person is a woman is $P(LH|W) = 0.20$.
- With the Bayes' rule we get:
$$P(M|LH) = \frac{P(LH|M) \times P(M)}{P(LH)}.$$
- To calculate this we need to compute $P(LH)$. This could be achieved with the following:
$$P(LH) = P(M) \times P(LH|M) + P(W) \times P(LH|W)$$
- Therefore the above probability is:
$$P(M|LH) = \frac{P(LH|M) \times P(M)}{P(LH)} = \frac{0.10 \times 0.5}{0.15} = 0$$

Bayesian inference for parameters

- A controversial aspect of Bayesian analysis is the specification of prior distribution for parameters to be estimated.
- In frequentist statistics only sample data are assumed to be "random variables" with associated probability distributions.
- Parameters are fixed but unknown quantities, and their associated p-values and confidence intervals are based on long-run frequency properties under repeated sampling of the data.
- From a Bayesian point of view, *both data and parameters can have probability distributions*, and so the Bayes' rule can be used to learn about probabilities of unobservable parameters as well as observable events (samples).
- We can set up the Bayes' rule for inference about a parameter θ as follows:

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)},$$

where $p()$ now denotes a probability density rather than a simple probability of an event.

Bayesian inference

- An interpretation of equation $p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$ can be the following:
  - $p(\theta)$ is the prior distribution for θ and expresses our uncertainty about its values before taking account of the observed data.
  - $p(\theta|y)$ is the posterior distribution for θ and represents the uncertainty about θ after conditioning on the data y.
  - $p(y|\theta)$ describes how the data depend on the parameter values (the likelihood of the sample).
  - The normalizing constant, $p(y)$, simply ensures that $p(\theta|y)$ is a valid probability distribution that integrates to 1.

- It turns out that it is usually not necessary to calculate $p(y)$ to evaluate properties of the posterior (from which inference is performed), and so the Bayes' rule can be simply expressed as:

$$p(\theta|y) \propto p(y|\theta)p(\theta),$$

where the proportionality is considered with relation to θ.

Some remarks about $p(y|\theta)$

- $p(y|\theta)$ arises from an assumed sampling distribution for the data.
- In Bayesian analysis we are mainly interested in $p(y|\theta)$ *as a function of* θ *for fixed* y. This means that here we are talking about the *likelihood function* $\mathcal{L}(\theta; y)$.
- Therefore we can restate the Bayes' rule as:
$$p(\theta|y) \propto \mathcal{L}(\theta; y)p(\theta).$$
- Essentially Bayesian analysis is based on the following statement:
$$\text{posterior} \propto \text{likelihood} \times \text{prior},$$
and all we need is to calculate the likelihood and to define our prior.

Toy example on a Bayesian setting: binomial data

- Suppose we have binomial data, i.e. we observe y successful responses out of n Bernoulli trials with probabilities θ.
- Here our parameter of interest is θ, the probability that we have a response in each trial.
- In this case, the likelihood will have the form:

$$\mathcal{L}(\theta|y,n) = \binom{n}{y}\theta^y(1-\theta)^{n-y}.$$

- Dropping the proportionality constant $\binom{n}{y}$ we obtain the likelihood:

$$\mathcal{L}(\theta|y,n) \propto \theta^y(1-\theta)^{n-y}.$$

- Now we have to decide what is our prior on θ.
- If we want that *the prior does not influence the likelihood*, we might choose a *flat prior*, for example a uniform distribution on θ, i.e. $\theta \sim \text{Unif}(0, 1)$.

Toy example on a Bayesian setting: binomial data (cont'd)

- In general, the probability density function of a Unif(a, b) is

$$f(x) = \begin{cases} \frac{1}{b-a}, & \text{for } a \leq x \leq b, \\ 0, & \text{for } x < a \text{ or } x > b \end{cases}$$

- When we have a Unif$(0, 1)$, we obtain:

$$p(\theta|y, \theta) \propto \theta^y (1-\theta)^{n-y} \times 1,$$

so that the posterior is still binomial.

Conjugacy

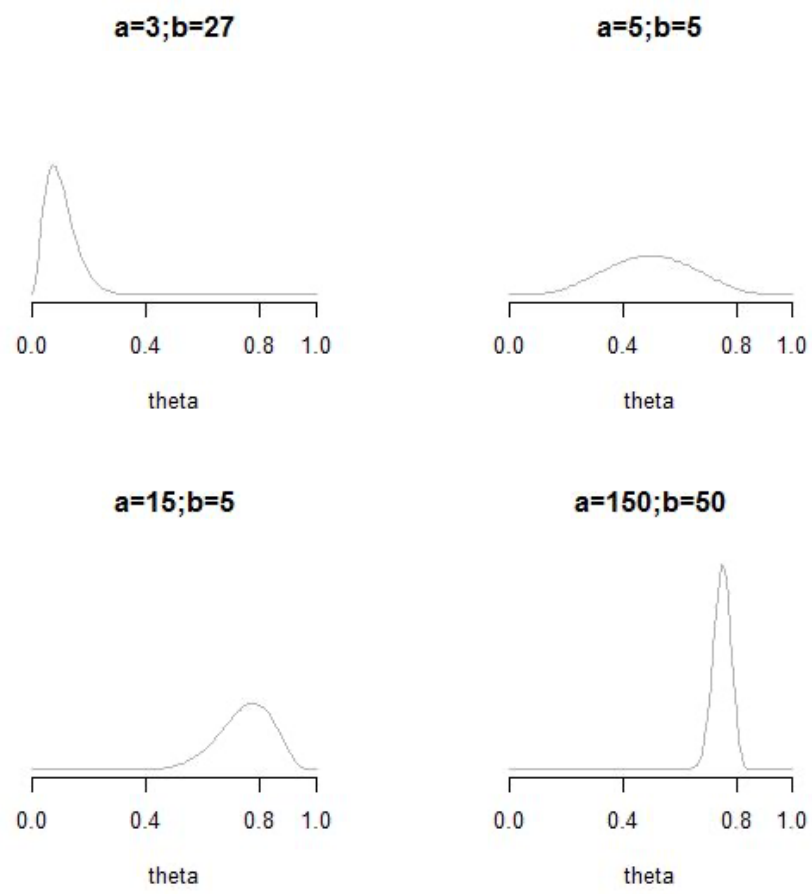
- The case above can be traced back to the case of the *conjugacy* of the Beta distribution with the Binomial likelihood.
- *Conjugacy of the Beta prior with Binomial likelihood*.
- Let $\mathcal{L}(\theta|y,n) = \text{Bin}(n,\theta)$ be the likelihood and $\text{Beta}(a,b)$ the prior density. Then the posterior density is $\text{Beta}(a+y, b+n-y)$.
- Sketch of proof.
- Ignoring the terms that do not depend on θ, we can write the posterior as:

$$\begin{aligned} p(\theta|y,n) &\propto \underbrace{\theta^y(1-\theta)^{n-y}}_{\text{likelihood}}\underbrace{\theta^{a-1}(1-\theta)^{b-1}}_{\text{prior}} \\ &= \theta^{y+a-1}(1-\theta)^{n-y+b-1} \\ &\propto \text{Beta}(y+a, n-y+b) \end{aligned}$$

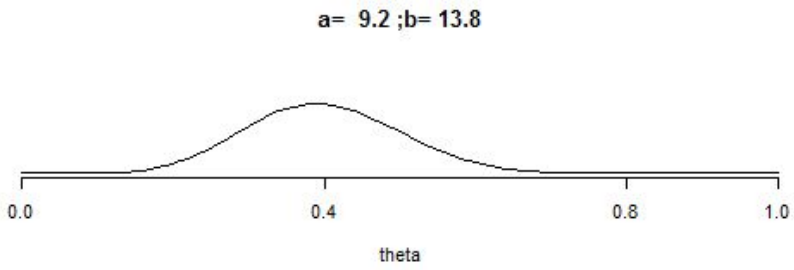
- Noting that a $\text{Beta}(1,1)$ is a $\text{Unif}(0,1)$, using the latter as a prior in a binomial likelihood setting leads to a $\text{Beta}(y+1, n-y+1)$ posterior.

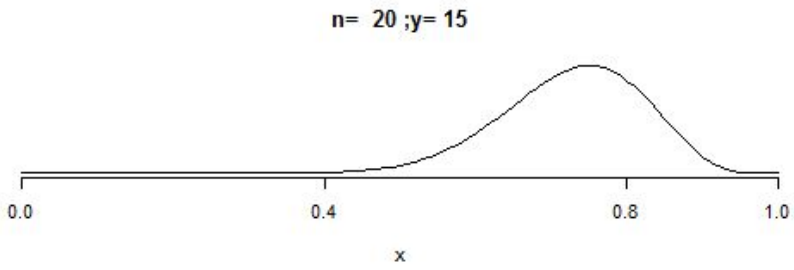
Toy example on a Bayesian setting: binomial data (cont'd)

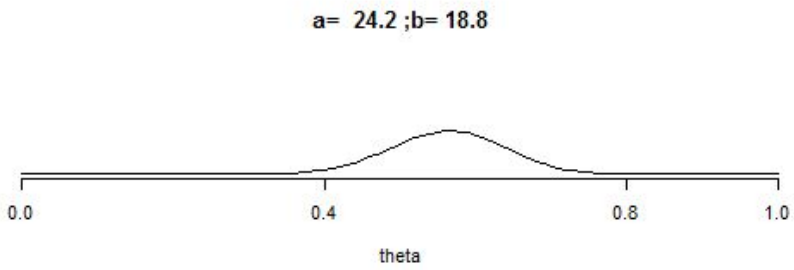
- The following are different Beta distributions with different parameters (a, b):



Toy example on a Bayesian setting: binomial data (cont'd)







Markov chains

- A Markov Chain is a collection of random variables X_t $(t = 1, 2, \ldots)$ having the property that if the chain is in a given *state* s_i, then it moves to state s_j at the next step with probability p_{ij} (transition probability).
- This probability does not depend on past states.
- The collection of all the possible states is called the *state space* (often denoted by $\mathcal{X}$).
- The collection of all transition probabilities from state i to state j forms the *transition matrix* which summarizes all the possibilities we have, given the collection of states.
- A random walk is an example of Markov Chain.

The origin of Markov Chain methods (cont'd)

This is where all started...

Science in Context **19**(4), 591–600 (2006). Copyright © Cambridge University Press
doi:10.1017/S0269889706001074 Printed in the United Kingdom

Classical Text in Translation

An Example of Statistical Investigation of the Text *Eugene Onegin* Concerning the Connection of Samples in Chains

A. A. Markov

(Lecture at the physical-mathematical faculty, Royal Academy of Sciences, St. Petersburg, 23 January 1913)[1]

This study investigates a text excerpt containing 20,000 Russian letters of the alphabet, excluding **Ь** and **Ъ**,[2] from Pushkin's novel *Eugene Onegin* – the entire first chapter and sixteen stanzas of the second.

This sequence provides us with 20,000 connected trials, which are either a vowel or a consonant.

Accordingly, we assume the existence of an unknown constant probability p that the observed letter is a vowel. We determine the approximate value of p by observation, by counting all the vowels and consonants. Apart from p, we shall find – also through observation – the approximate values of two numbers p_1 and p_0, and four numbers $p_{1,1}$, $p_{1,0}$, $p_{0,1}$, and $p_{0,0}$. They represent the following probabilities: p_1 – a vowel follows another vowel; p_0 – a vowel follows a consonant; $p_{1,1}$ – a vowel follows two vowels; $p_{1,0}$ –

The origin of Markov Chain methods

And this is the original paper... (available from myself for Russian-speaking people).

Известія Императорской Академіи Наукъ. — 1913.
(Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg).

Примѣръ статистическаго изслѣдованія надъ текстомъ „Евгенія Онѣгина“ иллюстрирующій связь испытаній въ цѣпь.

А. А. Марковъ.

(Доложено въ засѣданіи Физико-Математическаго Отдѣленія 23 января 1913 г.).

Наше изслѣдованіе относится къ послѣдовательности 20 000 русскихъ буквъ, не считая ъ и ь, въ романѣ А. С. Пушкина «Евгеній Онѣгинъ», которая заполняетъ всю первую главу и шестьнадцать строфъ второй.

Эта послѣдовательность доставляетъ намъ 20 000 связанныхъ испытаній, каждое изъ которыхъ даетъ гласную или согласную букву.

Соотвѣтственно этому мы допускаемъ существованіе неизвѣстной постоянной вѣроятности p буквѣ быть гласной и приближенную величину числа p ищемъ изъ наблюденій, считая число появившихся гласныхъ и согласныхъ буквъ. Кромѣ числа p мы найдемъ, также изъ наблюденій, приближенныя величины двухъ чиселъ p_1 и p_0 и четырехъ чиселъ $p_{1,1}$, $p_{1,0}$, $p_{0,1}$, $p_{0,0}$, представляющихъ такія вѣроятности: p_1 — гласной слѣдовать за гласной, p_0 — гласной слѣдовать за согласной, $p_{1,1}$ — гласной слѣдовать за двумя глас-

Example of a

Example of a transition matrix [From Kemeny et al. (1974)]

- In Baum's "The Wonderful Wizard of Oz" it is stated that the Land of Oz is blessed by many things, but not by good weather!
- Kemeny, Snell, and Thompson (1974) put out an example to explain intuitively a Markov Chain with the following terms (the discussion on Markov chains in the book is in this chapter: https://math.dartmouth.edu/~doyle/docs/finite/fm3/scan (https://math.dartmouth.edu/~doyle/docs/finite/fm3/sca
- In the Land of Oz, they never have two nice days in a row.
- If they have a nice day, they are just as likely to have snow as rain the next day.
- If they have snow or rain, they have an even chance (i.e. 50%) of having the same the next day.
- If there is change from snow or rain, only half of the time is this a change to a nice day.
- With this information try to form the transition

matrix of a Markov chain.

Example of a transition matrix [From Kemeny et al. (1974)] (cont'd)

- We take as states the kinds of weather R, N, and S.
- From the above information we determine the transition probabilities, and, using a square array, we get:

$$\mathbf{P} = \begin{array}{c|ccc} & R & N & S \\ R & 1/2 & 1/4 & 1/4 \\ N & 1/2 & 0 & 1/2 \\ S & 1/4 & 1/4 & 1/2 \end{array}$$

- The first row, for example, represents the probabilities for the various kinds of weather following a rainy day.

Example of a transition matrix [From Kemeny et al. (1974)] (cont'd)

- An important task of a transition matrix is to determine the probability of passing from state i to state j *after some time* (or some steps).
- Now, suppose you want to determine the probability $p_{ij}^{(2)}$ that, given the chain is in state i today, it will be in state j *two days from now*.
- In this example we see that if it is rainy today then the event that ij is snowy two days from now is the disjoint union of the following three events:
    1. it is rainy tomorrow and snowy two days from now;
    2. it is nice tomorrow and snowy two days from now;
    3. it is snowy tomorrow and snowy two days from now.

Example of a transition matrix [From Kemeny et al. (1974)] (cont'd)

- We get these probabilities:

$$p_{11}p_{13} = 1/2 \times 1/4 = 0.125,$$

and

$$p_{12}p_{23} = 1/4 \times 1/2 = 0.125,$$

and

$$p_{13}p_{33} = 1/4 \times 1/2 = 0.125.$$

- The probability is then: $0.125 \times 3 = 0.375$.

- In this way we have found the $(1, 3)$th element of the *power matrix* of order 2. The other elements are found in a similar way:

$$\mathbf{P}^2 = \begin{array}{c} \\ R \\ N \\ S \end{array} \begin{array}{ccc} R & N & S \\ p_{11}^{(2)} = .438 & p_{12}^{(2)} = .188 & p_{13}^{(2)} = .375 \\ p_{21}^{(2)} = .375 & p_{22}^{(2)} = .250 & p_{23}^{(2)} = .375 \\ p_{31}^{(2)} = .375 & p_{32}^{(2)} = .188 & p_{33}^{(2)} = .438 \end{array}$$

.

Markov Chain Monte Carlo algorithms: The basic setup

- A Monte Carlo Markov Chain (MCMC) algorithm is a method to generate random variables by using simulation (first 'MC' in the acronym) and Markov Chains (second 'MC' in the acronym).

- MOTIVATION 1.

    1. We have some probability distribution $\pi(\cdot)$ on some state space $\mathcal{X}$ (finite countable set).
    2. What we want is this: we want to have an i.i.d. sample from $\pi(\cdot)$:

    $$X_1, X_2, \dots, X_n \sim \pi(\cdot).$$

    3. After obtaining this sample we could estimate the mean of $\pi(\cdot)$ by $\frac{1}{n}\sum_{i=1}^{n} X_i$ or we could estimate the expectation of any function g of the sample elements with respect to π (i.e. $E_\pi[g]$) by $\frac{1}{n}\sum_{i=1}^{n} g(X_i)$., where $g : \mathcal{X} \to \mathbb{R}$.
    4. For example, g could represent moments.

Markov Chain Monte Carlo algorithms: The basic setup

- MOTIVATION 2.

    1. Another motivation is that we could be interested in the Bayesian context.
    2. Suppose $\pi(\cdot)$ is a Bayesian posterior distribution.
    3. Most of the times it is a very complicated distribution.
    4. Also, in these contexts $\mathcal{X}$ is high-dimensional.
    5. When multi-dimensional integrals have to be computed, numerical methods might not be useful anymore.
    6. So we use a Markov chain $X_0, X_1, \ldots, X_n$ which has $\pi(\cdots)$ as *stationary distribution* (s.d.), i.e. for large n, $X_n \sim \pi(\cdots)$.
- *Statonary distribution*: A stationary distribution of a MC is a probability distribution that remains unchanged in the Markov chain as time progresses.

Markov Chain Monte Carlo algorithms: The challenge

- Let $x \in \mathcal{X}$, $A \subseteq \mathcal{X}$ and $P(x, A)$ a transition matrix on A , i.e. a matrix containing all the transition probabilities from state x to any state in subset A :

$$P(x, A) \equiv P[X_{n+1} \in A | X_n = X].$$

- The challenge is to have transition probabilities $P(x, A)$ which are easily simulated on a computer, and such that $\pi(\cdot)$ is a stationary distribution (s.d.).

- A s.d. is such that if $X_n \sim \pi(\cdot)$, then $X_{n+1} \sim \pi(\cdot)$, or if $X_n \sim \pi(\cdot)$, then $P[X_{n+1} \in A] = \pi(A)$, $\forall A \subseteq \mathcal{X}$, or $P[X_{n+1} \in A] = P(X, A)$, or, finally:

$$\int \pi(x) P(x, A) dx = \pi(A), \forall A \subseteq \mathcal{X}$$

for the continuous case, and

$$\sum_{x \in \mathcal{X}} \pi(x) P(x, A) = \pi(A), \forall A \subseteq \mathcal{X}, (3)$$

for the discrete case.

A simple example

- Suppose we have three states and therefore the state space is $\mathcal{X} = \{1, 2, 3\}$.
- Suppose we have this initial distribution:

$$\pi(1) = \frac{1}{6}$$
$$\pi(2) = \frac{1}{3}$$
$$\pi(3) = \frac{1}{2}.$$

- Suppose we have the following three possibilities for $A \subseteq \mathcal{X}$: $A = \{1\}$, $A = \{2\}$ or $A = \{3\}$.
- Suppose finally that we have a Markov chain with the following transition matrix:

$$P(1, \{2\}) = 1 \quad P(2, \{1\}) = 1/2 \quad P(3, \{2\}) = 1/$$
$$P(1, \{1\}) = 0 \quad P(2, \{3\}) = 1/2 \quad P(3, \{3\}) = 2/$$
$$P(1, \{3\}) = 0 \quad P(2, \{2\}) = 0 \quad P(3, \{1\}) = 0$$

A simple example (cont'd)

- Using formula (3) we have that:
  - for $A = \{1\}$:

$$\begin{aligned}
&\sum_{x\in\mathcal{X}} \pi(x)P(x,\{1\}) \\
&= \pi(1)P(1,\{1\}) + \pi(2)P(2,\{1\}) + \pi(3 \\
&= \tfrac{1}{6}\times 0 + \tfrac{1}{3}\times\tfrac{1}{2} + \tfrac{1}{2}\times 0 \\
&= \tfrac{1}{6} = \pi(1).
\end{aligned}$$

  - for $A = \{2\}$:

$$\begin{aligned}
&\sum_{x\in\mathcal{X}} \pi(x)P(x,\{2\}) \\
&= \pi(1)P(1,\{2\}) + \pi(2)P(2,\{2\}) + \pi(3 \\
&= \tfrac{1}{6}\times 1 + \tfrac{1}{3}\times 0 + \tfrac{1}{2}\times\tfrac{1}{3} \\
&= \tfrac{1}{3} = \pi(2).
\end{aligned}$$

  - for $A = \{3\}$:

$$\begin{aligned}
&\sum_{x\in\mathcal{X}} \pi(x)P(x,\{3\}) \\
&= \pi(1)P(1,\{3\}) + \pi(2)P(2,\{3\}) + \pi(3 \\
&= \tfrac{1}{6}\times 0 + \tfrac{1}{3}\times\tfrac{1}{2} + \tfrac{1}{2}\times\tfrac{2}{3} \\
&= \tfrac{1}{2} = \pi(3).
\end{aligned}$$

In [22]:
```
import numpy as np
import pdb
a = np.array([[0.9,0.075,0.025], [0.15, 0.8, 0.05], [0.25, 0.25
b = np.array([[0 , 1, 0]])
next_b = b.dot(a)
print(next_b)
for i in range(1,100):
    next_b = b.dot(a)
    #pdb.set_trace()
    b = next_b
print(b)
```

```
[[0.15 0.8  0.05]]
[[0.625  0.3125 0.0625]]
```

The reversibility property

- The last example can be used to show the so-called *reversibility property*.

- *Proposition*. **The reversibility property** (Also called the *detailed balance*)

$$\pi(x)P(x, \{y\}) = \pi(y)P(y, \{x\}).$$

- *Theorem*. **Reversibility implies stationarity** If a Markov chain is reversible with respect to some distribution $\pi(\cdot)$, then $\pi(\cdot)$ is a stationary distribution.

The Metropolis-Hastings algorithm

- The reversibility property can be used to define an important MCMC algorithm, the **Metropolis-Hastings algorithm**.

The Metropolis-Hastings algorithm

- Suppose $\pi(x)$ is the density of a r.v. X from which we want to generate values, but it is difficult to get these values.

- Let $q(y|x)$ be a **proposal distribution** *from which it is easy to draw* instead.

- Let $\alpha(x, y) = \min(1, \frac{\pi(y)q(x|y)}{\pi(x)q(y|x)})$.

- The algorithm works as follows:
  1. Start with x_0, your initial value.
  2. Given x_n, generate y_{n+1} from $q(y|x_n)$.
  3. With probability $\alpha(x_n, y_{n+1})$ we "accept" y_{n+1} and set $x_{n+1} = y_{n+1}$.
  4. With probability $[1 - \alpha(x_n, y_{n+1})]$ we "reject" y_{n+1} and set $x_{n+1} = x_n$.

Main Metropolis-

Main Metropolis Hastings algorithm variations

- Some variations of the MH has been put forward, especially for the choice of the proposal q .

Variation 1: Symmetric M-H

- In this case we have:

$$q(x|y) = q(y|x)$$

and therefore the acceptance probability simplifies to

$$\alpha(x, y) = \min(1, \frac{\pi(y)}{\pi(x)}).$$

Variation 2: Independent M-H Sampler

- In this case we have:

$$q(y|x) = q(y),$$

and

$$q(x|y) = q(x),$$

Therefore the acceptance probability simplifies to

$$\alpha(x, y) = \min(1, \frac{\pi(y)q(x|y)}{\pi(x)q(y|x)}) = \min(1, \frac{\pi(y)q(x)}{\pi(x)q(y)})$$

where $w(y) = \frac{p(y)}{q(y)}$ and
$w(x) = \frac{p(x)}{q(x)}$.

Variation 3: Symmetric Random Walk Metropolis

- This is the case when at time t in the M-H algorithm we can say:

$$y_t = x_{t-1} + \epsilon,$$

i.e.:

$$y_t - x_{t-1} = \epsilon,$$

with ϵ being a simple "random noise", so that the y_t value is a random deviance from x_{t-1}

- So we can state, if the indipendence still holds, from (4):
$$q(y_t|x_{t-1}) = q(y_t - x_{t-1}),$$

and we have a *symmetric random walk M-H* algorithm.

- The acceptance probability at time t becomes, if we consider the random walk increment ϵ_t as symmetric:

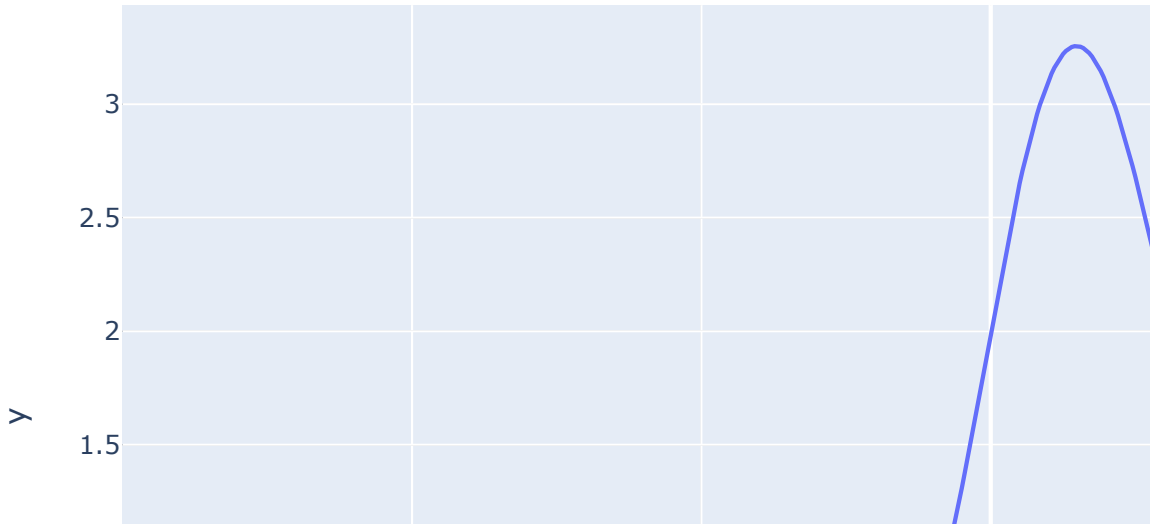
$$\alpha(x_{t-1}, y_t) = \min(1, \frac{\pi(y_t)q(x_{t-1}|y_t)}{\pi(x_{t-1})q(y_t|x_{t-1})}) = min(1, \frac{\pi(y_t)}{\pi(x_{t-1})}$$

Example

Suppose we have a "bad" distribution like the following:

$$\pi(x) \propto \exp(-x^2)(2 + sin(5x) + sin(2x))$$

In [43]:
```
import numpy as np
import plotly.express as px
def targdist (x):
    prob_x = np.exp(-x**2) * (2 + np.sin(5*x) + np.sin(2*x))
    return prob_x
x = np.arange(-3,3,0.01)
y = targdist(x)
px.line(x = x, y = y)
```



In [44]:
```
# This is not a proper density, but we can think of it like the
from scipy.integrate import quad
def integrand(x):
    return np.exp(-x**2) * (2 + np.sin(5*x) + np.sin(2*x))

I = quad(integrand, -3, +3)
#Result with error due to approximation:
I
```

Out[44]: (3.544829393038085, 9.3747309448753e-12)

We need a proposal distribution. Let's think of a normal for the currently generate value x^* :

$$X^*|x_{n-1} \sim N(x_{n-1}, \sigma^2),$$

for which we have to think about a constant value for σ^2.

Therefore, the acceptance probability is given by:

$$\alpha(x^*, x_{n-1}) = \min(1, \frac{\pi(x^*)q(x_{n-1}|x^*)}{\pi(x_{n-1})q(x^*|x_{n-1})}).$$

We can easily get $\frac{\pi(x^*)}{\pi(x_{n-1})}$ by simply plugging x^* and x_{n-1} in the "density" formula.

As for $\frac{q(x_{n-1}|x^*)}{q(x^*|x_{n-1})}$) but the candidate q is a normal and therefore is symmetric, so we have that $\frac{q(x_{n-1}|x^*)}{q(x^*|x_{n-1})}) = 1$ (see below the symmetric M_H).

Using the acceptance probability we have to decide if a newly generated value x^* should be accepted or not as a candidate for x_n.

We do this using a uniform distribution value and comparing it to α.

In [45]:
```
import pandas as pd
import plotly.figure_factory as ff
from ipywidgets import VBox, HBox, Output, Button, IntText, Lab
import matplotlib.pyplot as plt
from matplotlib.transforms import Affine2D
from matplotlib.collections import PathCollection
import matplotlib.animation as animation
```

In [46]:
```
def metropolis_sym (x0, sigma):
    xp = np.random.normal(0,sigma)
    alpha = targdist(xp)/targdist(x0)
    u = np.random.rand()
    if u <= alpha: #if accepted:
        x1=xp
        #a = 1 #flag for the acceptance
    else: #if not accepted:
        x1=x0
        #a = 0 #flag for the rejection
    return x1#, a
```

In [47]:
```
#calling the function
nsamples = 60
X = np.zeros([nsamples,1])
init = 1
sig = 2
for j in range (nsamples):
    x = metropolis_sym(init, sig)
    X[j] = x
df = pd.DataFrame (data = X, columns = ['Generated values'])
df

```

Out[47]:

	Generated values
0	1.000000
1	0.786338
2	0.249329
3	-1.303657
4	-0.146073
5	1.000000
6	1.000000
7	-0.221852
8	1.000000
9	1.000000

10	0.275779
11	-0.099048
12	1.000000
13	1.000000
14	0.348666
15	0.310122
16	1.565077
17	0.227138
18	0.062124
19	1.993310
20	1.000000
21	0.478702
22	-0.427539
23	1.616411
24	0.408681
25	-0.794598
26	1.000000
27	-0.326671
28	1.000000
29	1.000000
30	0.965877
31	-0.649515
32	1.000000
33	1.000000
34	0.689489
35	1.000000
36	1.000000
37	1.048599
38	1.000000
39	1.000000
40	1.000000
41	-1.111314
42	1.000000
43	-1.993214

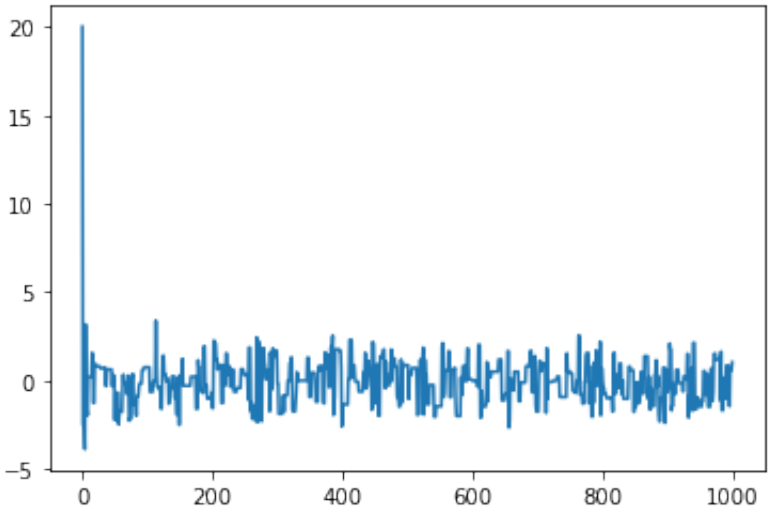
44	-0.117532
45	1.000000
46	-1.204570
47	1.174553
48	1.000000
49	0.491595
50	0.038732
51	1.000000
52	-1.508346
53	1.000000
54	1.000000
55	1.000000
56	-0.143915
57	-0.820256
58	-1.473826
59	0.356016

A quick example for the symmetric random walk (from Hastings, 1970)

- The target density is standard normal.
- The proposal distribution is $U[-\delta, \delta]$.
- The proposed value is accepted with probability $= min(1, \exp(x_{t-1}^2 - y_t^2)/2)$

In [48]:
```
#Example with uniform with ptoposal and N(0,1) as a target (Has
import matplotlib.pyplot as plt
import numpy as np
import math
import random
def MHSampler(initialvalue,k,numiters):
    x = initialvalue
    currentx = initialvalue
    S = np.ones(numiters)
    S[0] = x
    J = np.ones(numiters)
    J[0] = 0
    for j in range (1,numiters):
        y = np.random.uniform(-k, k, 1)
        alpha = math.exp((pow(currentx, 2) - pow(y,2))/2)
        #alpha = math.exp((k-1)*(y-currentx))
        if (alpha>1):
            alpha<-1
        currentx = random. choices(population = [y,x], weights =
        S[j] = currentx
        J[j] = j
        x = currentx
    mean = np.mean(S)
    plt.plot(S)
    plt.show()
```

In [49]:
```
MHSampler(20,4,1000)
```



The Gibbs sampler (as

explained in Casella & George 1992)

- The Gibbs sampler is an MCMC method for generating random variables from conditional distributions indirectly, without having to compute the density.

- Suppose we are given a joint density $f(x, y_1, y_2, \ldots, y_p)$, and are interested in sampling values from the following marginal density in order to obtain some characteristics such as the mean or variance of r.v. X :

$$f(x) = \int \ldots \int f(x, y_1, y_2, \ldots, y_p) dy_1, dy_2, \ldots, d$$

- After computing $f(x)$ we obtain the desired characteristics, but sometimes integrals in (4) are nasty.

- Rather than computing or approximating $f(x)$ directly, the Gibbs sampler allows us to effectively generate a sample $(X_1, \ldots, X_m) \sim f(x)$ without requiring $f(x)$.

- By simulating a large enough sample, the mean, variance, or any other characteristic of $f(x)$

can be calculated to the desired degree of accuracy.

- The original idea was by Geman & Geman (1984), but the statistical potential was overviewed by Gefand & Smith (1990).
- Those interested in the proof of the Gibbs sampling can read the original articles.
- There is a soft proof in Casella & George 1992.

The Gibbs sampler (as explained in Casella & George 1992)

The Gibbs sampler algorithm in the original Gelfand and Smith's (1990) version

- *Task* and *assumptions*: we want to obtain characteristics of $f(x)$ using the conditional distributions $g(y|x)$ and $g(x|y)$ which are supposed known.

- 1. Start with an initial value $Y_0' = y_0'$.

- 2. For $j: 0$ to K , obtain a 'Gibbs sequence' of random variable values: $Y_0', X_0', Y_1', X_1', \dots Y_K', X_K',$ as follows:
  - [2.a] Generate a value from $X_j' \sim g(x|Y_j' = y_j')$;
  - [2.b] Generate a value from $Y_{j+1}' \sim g(y|X_j' = x_j')$.

3. Take the K -th value $X_K' = x_K'$: this represents, as $K \to \infty$, a sample point from $f(x)$.

Suggested reading and references

- Casella, G., George, E. I. (1992). Explaining the Gibbs sampler. *The American Statistician*, 46(3): 167-174.

- Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., Rubin, D. B. (2013). *Bayesian Data Analysis* (Third Edition). Chapman & Hall/CRC Press.

The Gibbs sampler for a vector of parameters (1)

- Suppose we have a two-dimensional vector of parameters $\theta = (\theta_1, \theta_2)$
- For example, when we deal with the normal distribution we have $\theta = (\mu, \sigma^2)$.
- Suppose we are in a Bayesian setting and we want to estimate the joint posterior distribution $p(\theta|X)$.
- Suppose we can easily draw random samples from the two conditional distributions:

$$p(\theta_1|\theta_2, X)$$
$$p(\theta_2|\theta_1, X)$$

- Now we can use the Gibbs sampling and construct the Markov chain by sampling values from the conditional distributions above alternating each parameter θ_i , $i \in \{1, 2\}$ in turn, treating all other parameters as observed.

The Gibbs sampler for a vector of parameters (2)

- Since Bayesian hierarchical models are typically set up as products of conditional distributions, the Gibbs sampler is ubiquitous in Bayesian modeling.
- Where it is difficult to sample from a conditional distribution, we can sample using a Metropolis-Hastings algorithm instead.
- In this context, Gibbs sampling is a type of random walk through parameter space, and hence can be thought of as a Metropolis-Hastings algorithm with a special proposal distribution.
- At each iteration in the cycle, we are drawing a proposal for a new value of a particular parameter, where the proposal distribution *is* the conditional posterior probability of that parameter.
- This means that the proposal move is *always* accepted.

- Hence, if we can draw samples from the conditional distributions, Gibbs sampling can be much more efficient than regular Metropolis-Hastings.

Towards Bayesian hierarchical modeling with the Gibbs sampler: specifying the prior

An important task in Bayesian analysis is the specification of priors.

- There are many types of priors, each of which is chosen for a specific purposes.

- So, we have:
  - *Reference priors*:
    - *Jeffreys' prior.
    - *Improper (flat or uninformative) priors*.
  - *Vague priors*.
  - *Informative priors*.

Towards Bayesian hierarchical modeling with the Gibbs sampler: reference priors

Reference priors are considered "default priors", in the sense that they are considered suitable for the particular model we are implementing.

- They are considered the best priors when there is no subjective idea about the prior information.
- The most used method to get a reference prior is the *Jeffreys' rule*.
- Very frequently reference priors are *improper* (see below).

Jeffreys' priors

- Jeffreys priors are defined in terms of the *Fisher information*, which tells us how much information about an unknown parameter we can get from a sample.

- In other words, Fisher Information tells us how well we can measure a parameter, given a certain amount of data.

- Jeffreys' priors get the priors equal to the square matrix of the determinant of the Fisher information matrix (cannot give the details of this here...).

- The Fisher information is a way of measuring the information about parameters carried out by a sample.

- The partial derivative with respect to the parameter θ of the natural logarithm of the likelihood function is called the *score*.

- So we have this expression for the score:

$$s(\theta) = \frac{\delta}{\delta x} \log f(X; \theta).$$

- The expected value of the score is 0.

- The variance of the score is the Fisher information $\mathcal{I}(\theta)$.

Jeffreys' priors and flat priors

- Generally speaking, flat priors (see below) are used for location parameters (the mean, the median, etc.) indicating the central position of a r.v., whereas inverse priors are Jeffreys' priors for scale parameters (the standard deviation, for example), indicating the spread of the distribution.

- This makes sense because if we are in a situation in which there is no information about the central position of the parameter, giving the same probability to each point could be a good choice.

- This make sense also for scale parameters because inverse priors on scale parameters can be translated in ignorance about this scale parameters. Inverse priors on a scale parameter are equivalent to flat priors on the log scale.

Improper priors

- Generally speaking, improper priors are the ones that formally do not integrate to 1 along their support.
- However, the posterior distribution will result in a proper distribution, integrating to 1 .
- This is done is when you want to give the prior distribution the minimum possible influence with respect to the likelihood.

Improper priors (cont'd)

- To sum up, priors such as $\pi(\mu) = 1$ or $\pi(\sigma) = \frac{1}{\sigma}$ are called improper as they not integrate to 1 (formal requirement for a density).
- The area under these improper priors is therefore equal to ∞ .
- In most cases, improper priors can be used in Bayesian analysis with no major difficulties.
- But in some cases also posteriors result improper (watch this!).

Vague priors

- With *vague priors* we mean priors with large spread (large variance).
- Flat priors can be approximated by vague density priors when some software does not allow the use of flat priors (like WinBUGS).
- The inverse prior $\pi(\sigma)$ can be successfully approximated by a Gamma density with small shape parameter and rate parameter.

Informative priors

- When prior information is available about the parameter θ , it should be included in the prior distribution of θ .
- For example, if the present model form is similar to a previous model form, and the present model is intended to be an updated version based on more 'up-to-date' data, then the posterior distribution of θ from the previous model may be used as the prior distribution of θ for the present model.
- In this way, each version of a model does not start from scratch, based only on the present data, but the cumulative effects of all data, past and present, can be taken into account.
- Sometimes informative prior information might reside in an *expert*.
- In this case, the personal belief about the probability of the event "elicited" by this expert must be taken into account into the form of a proper probability density function.
- This process is called *prior elicitation*.

Hierarchical priors

- Hierarchical priors are two-stage or more than two-stage priors, i.e.a prior is placed on a prior which is placed on another prior and so on.
- Suppose you have a likelihood which is $N(\mu, \sigma^2)$.
- One can put a prior $N(\nu, \tau^2)$ on μ .
- But additionally, one can put priors on the parameters ν and τ^2 to obtain an *hierarchical Bayesian model*.

Hierarchical Bayesian models (1)

- More formally, we can extend a Bayesian model of the form we have seen above:

$$p(\theta|y) \propto p(y|\theta)p(\theta),$$

to be a hierarchical Bayesian model by taking:

$$p(\theta, \phi|y) \propto p(y|\theta)p(\theta|\phi)p(\phi),$$

with $p(\phi)$ being an hyperprior for θ .

Hierarchical Bayesian models (2)

- Less formally, hierarchical Bayesian models with two levels of hierarchy (the most used ones) have the following structure:
  1. first we specify that the data come from a distribution with parameters θ :
  $$X \sim f(X \mid \theta)$$
  2. then we specify that the parameters themselves come from another distribution with hyperparameters λ :
  $$\theta \sim g(\theta \mid \lambda)$$
  3. Finally we specify that λ comes from a prior distribution
  $$\lambda \sim h(\lambda)$$

- More levels of hierarchy are allowed - i.e you can specify hyper-hyperparameters for the distribution of λ and so on.

Problems involving priors

Before the Bayesian model setup and the data collection we can face the following problems:

1. COMPUTATIONAL ISSUES. The posterior distribution has to be computed in closed form. One can use conjugate forms to be sure to obtain a proper posteriors.

2. SENSITIVITY (also called ROBUSTNESS). If we know nothing about the parameters we can choose a series of prior distributions. We can ask ourselves: How is the choice of the prior influencing the conclusion of the study and the predictive power of the Bayesian model?

3. SUBJECTIVITY. Enables the researchers to include in the model their own belief, experiences, and so on.

Elicitation

- The elicitation exercise involves the correct choice of a prior in a Bayesian model to capture the experts' knowledge about one or more uncertain quantities in probabilistic form.
- The purpose of such elicitation is to construct a probability distribution that properly represents the expert's knowledge/uncertainty.
- The person whose knowledge is to be elicited is usually referred to as an "expert", and while in principle there is no particular reason for them to have special knowledge or expertise, the fact that someone deems it worthwhile to carry out the elicitation implies that the expert's knowledge and judgements are worth having!
- This elicitation exercise is performed to *augment* the sample information.
- Therefore, elicitation of prior information is accepted as having a fundamental role in Bayesian statistics.

Step by step examples with OpenBUGS. Example 1: performing frequentist analysis with Bayesian model

- For this example please refer to the following OpenBUGS file:
  - `Regressione Step by Step.odc`

Step by step examples with OpenBUGS. Example 1:

performing frequentist analysis with Bayesian model (cont'd)

- The aim of this example was to build a simple regression model in a frequentist fashion (with no Bayesian inference at all) using WinBUGS/OpenBUGS. The OpenBUGS model was to be implemented on the following data:

RegressionModel-frequentist

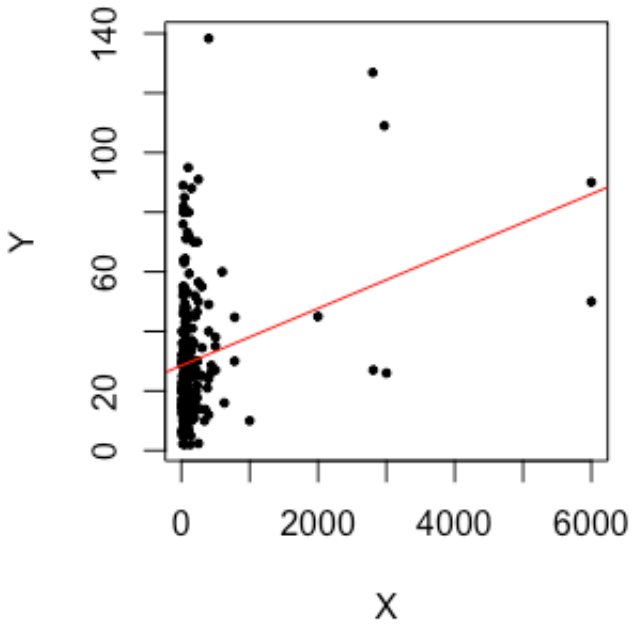
```
#data
list(x=c(33.3,155,30,11.7,  70.3,51.99,30,303.26,140,
147.09,250,44,350,120,25,45,42,50,64,42,79,397.8,5,80,62,23.45,
300,25,208,70,25,19.64,63,98,250,21.14,54.52,91,34,6000,30,53,2
34,340,429,25,20.8,82,600,200,200,30,50,60,20,30,45,36,398,83.2
,103.51,88,500,38,117.16,25.07,12.77,10,78,6,97,105,3000,52,43.
96,107.25,40,55,70,300,100,30,85.8,120,45,16,70.2,62.4,59.87,12
.25,20,80,56.8,80,27,50,58,75,49,20,47,109,250,150,48.09,73,67.
76,20,128.83,83.2,184,39,200,40,71,220,39.35,200,82,36.34,62.4,
42.81,400,110.11,60,50.65,120,39,91.94,12,25,72,30,162,50.6,50,
30,380,12.13,52,5,20,1000,40,58,26,50,200,230,28,25,116,80,117,
100,229.95,100,276,110,209.26,70,100,25,41,100,10,38,18.4,628,5
0,220,100,60,150,141.34,30,113,57,191.48,45,111.23,138.93,60.78
,100,84.6,30,100,112.5,73.05,137.41,60,51.32,70,30,20,24.3,106,
39.21,31.2,23.4,33,43.63,400,5.86,13,70,120,53,79.6,100,10.29,1
80,49.05,400,100,69,500,35.46,62.4,234,30,10.28,89.66,15,87.03,
46.5,60,50,19.76,25,34.77,150,56.05,250,70,2970,500,40.79,20,34
.67,52,75,120,2808,39,2800,90,780,68,124,80,39,15.6,150,33,25,6
5,66,50,50,230,75,35,50,54.9,71,100,50,180,30,40,200,23.4,439,1
10,35,12.09,60,45,150,25,75,50,2000,45,35,780,133.14,112.3,41.68,20,150,40,168,6000,125,30),
y=c(10,25,18,40,29,18,34.5,34.5,16,14,56.5,8.16,13.8,24.5,52,17
,48,20.04,20,63,19,24,15,44,30,15.99,25,5,30,15,25,5,20,36.99,5
0,16,43,53,11,50,24,48.5,70,10,26,25,2.26,25,60,17,15,40,15.4,1
3,22,17,25,30,12,21.88,25,45,35,20.99,15,89,20,20,25.23,32,36,3
1,26,20.81,25.88,25,39,19,40,55,7.59,11,22,13,28,30.35,19,22.5,
19.5,20,30,80,16,19,30,10,20,17.5,2,40,40,80,2.4,16,10.5,30,34.
35,14,29,21,36,20,14.35,15.77,41,20,5,45,29.01,14.31,25,26,138.
34,72,13,54,25,46,26.5,21,30,24.5,15,41,7.84,15,53.5,21,22.49,3
3,20,12,10,30,35,55,49.5,52,46.55,46,30,30,30,30,14,17.78,20,14
,25,25,23.65,30,76,30.91,28,12.6,46,21.22,16,7.14,27,6.8,47.5,1
0,25,18.85,15,21,10.94,47,45,5,9.98,22,25,33,12.2,28,11.5,33,36
.5,23,20,40,5,17,30,12,20.99,14.5,28.5,27.48,49,6.25,36,48,17,1
1.5,24,16,14,70,17,40,95,29.99,38,54.91,23.5,30,35,28.76,12.5,2
3,29,30,14.35,35,25,20.84,64,34,64.41,91,71,109,27,30,30,55,20,
21.5,25,27,33,126.89,73.2,30,10.4,19,45,12.5,17,37,82,15,20,42,
40,11.15,22,28,35,17.5,40,27,45,25,12,80,9.92,26,18,28.6,12.95,
30,17,31,85,25,10,45,30,45,40,34,44.73,2,59.38,30,7.5,88,34,28,90,45,30))
```

- These data contain 297 records taken from the UK Food and Expenditure Survey.
- There are two variables: PensAmt which expresses the amount in pounds of the last installment paid for a private pension and TelBgAmt which contains the amount in pounds of the last bill of the paid telephone.
- We aim at finding the regression line that has TelBgAmt as dependent variable (y) and PensAmt as independent variable (x).
- Therefore it's a toy example with a simple regression model.

- We will show that the Python command (giving frequentist results) will have the same output of a particular Bayesian model implemented in OpenBUGS which can be considered not Bayesian at all!

Example 1 (cont'd)

- Plot of the data gives the following picture:



In [33]:

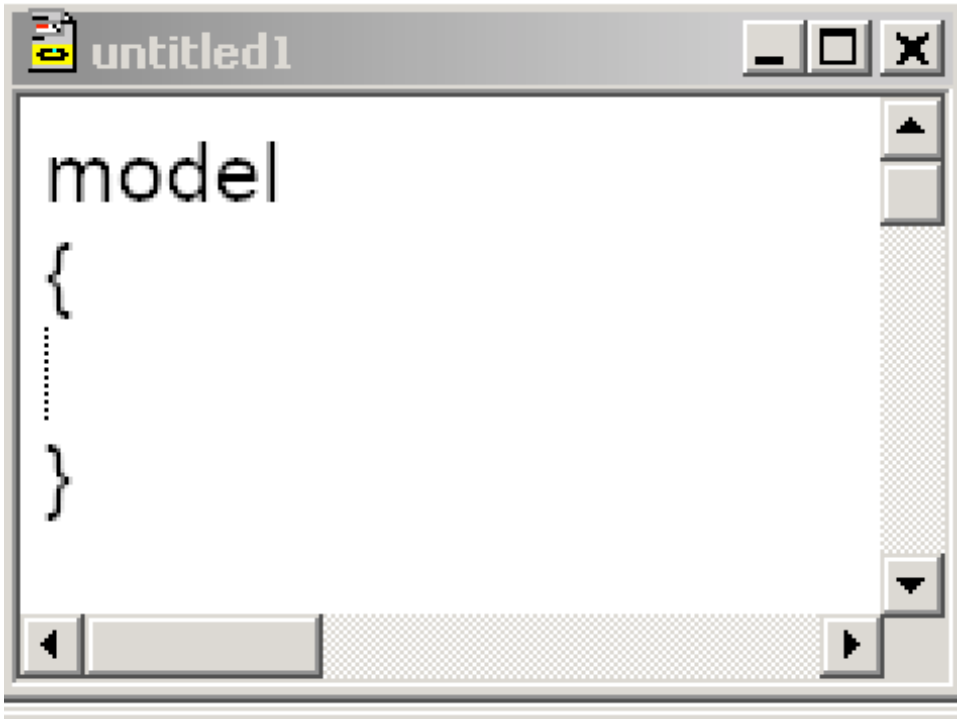
```
from sklearn.linear_model import LinearRegression
import numpy as np

x = np.array([33.3,155,30,11.7, 70.3,51.99,30,303.26,140,
        147.09,250,44,350,120,25,45,42,50,64,42,79,397.8,5,80,
        250,21.14,54.52,91,34,6000,30,53,234,340,429,25,20.8,8
        83.2,103.51,88,500,38,117.16,25.07,12.77,10,78,6,97,10
        30,85.8,120,45,16,70.2,62.4,59.87,12.25,20,80,56.8,80,
        73,67.76,20,128.83,83.2,184,39,200,40,71,220,39.35,200
        120,39,91.94,12,25,72,30,162,50.6,50,30,380,12.13,52,5
        117,100,229.95,100,276,110,209.26,70,100,25,41,100,10,
        57,191.48,45,111.23,138.93,60.78,100,84.6,30,100,112.5
        39.21,31.2,23.4,33,43.63,400,5.86,13,70,120,53,79.6,10
        234,30,10.28,89.66,15,87.03,46.5,60,50,19.76,25,34.77,
        2808,39,2800,90,780,68,124,80,39,15.6,150,33,25,65,66,
        439,110,35,12.09,60,45,150,25,75,50,2000,45,35,780,133

y = np.array([10,25,18,40,29,18,34.5,34.5,16,14,56.5,8.16,13.8,
        25,5,30,15,25,5,20,36.99,50,16,43,53,11,50,24,48.5,70,
        17,25,30,12,21.88,25,45,35,20.99,15,89,20,20,25.23,32,
        11,22,13,28,30.35,19,22.5,19.5,20,30,80,16,19,30,10,20
        20,14.35,15.77,41,20,5,45,29.01,14.31,25,26,138.34,72,
        22.49,33,20,12,10,30,35,55,49.5,52,46.55,46,30,30,30,3
        46,21.22,16,7.14,27,6.8,47.5,10,25,18.85,15,21,10.94,4
        12,20.99,14.5,28.5,27.48,49,6.25,36,48,17,11.5,24,16,1
        14.35,35,25,20.84,64,34,64.41,91,71,109,27,30,30,55,20
        11.15,22,28,35,17.5,40,27,45,25,12,80,9.92,26,18,28.6,
        34,28,90,45,30])


#model = LinearRegression()
model = LinearRegression().fit(x, y)
print('intercept:', model.intercept_)
print('coefficient:', model.coef_)
```

```
intercept: 28.465530287645144
coefficient: [0.00959975]
```

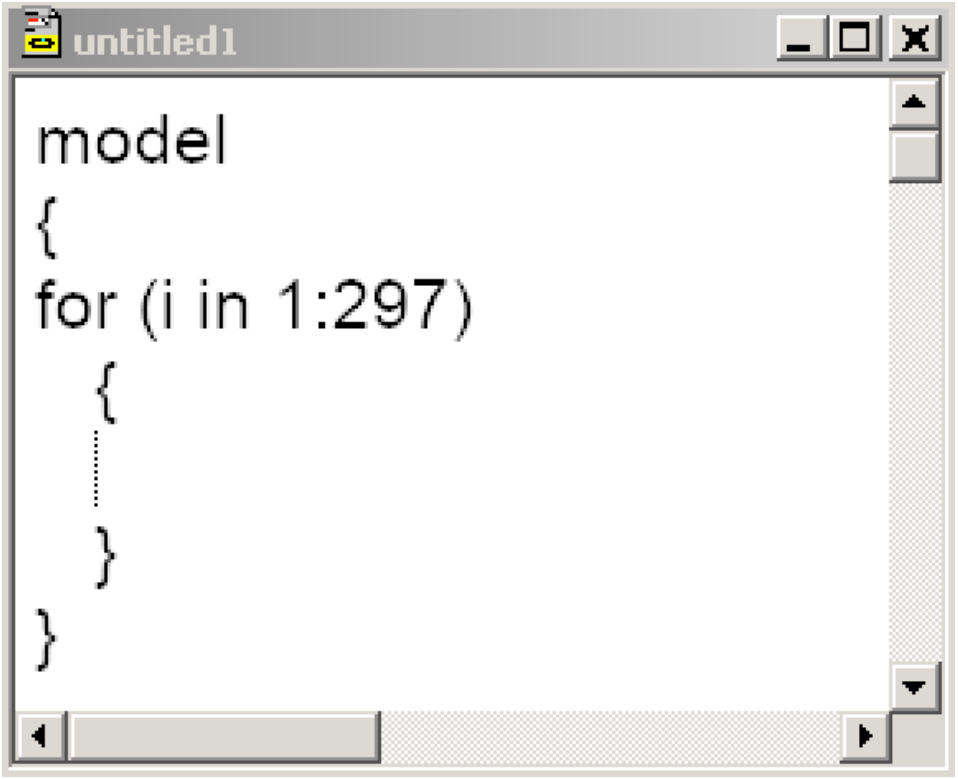
Step-by-step implementation of the simple regression

model in OpenBUGS

- Step 1. Open a new odc file with the sequence *File* + *New* and write the syntax for a model as the following figure shows:

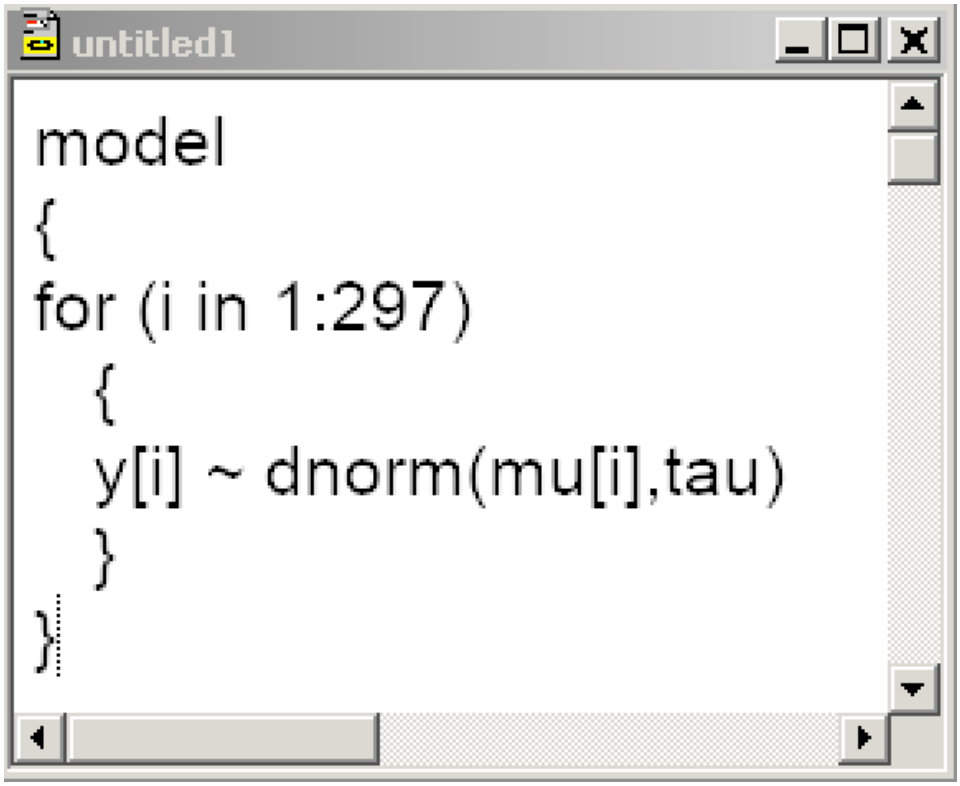


- Step 2. Similarly to the R syntax, write the syntax of a *for{ }* loop:

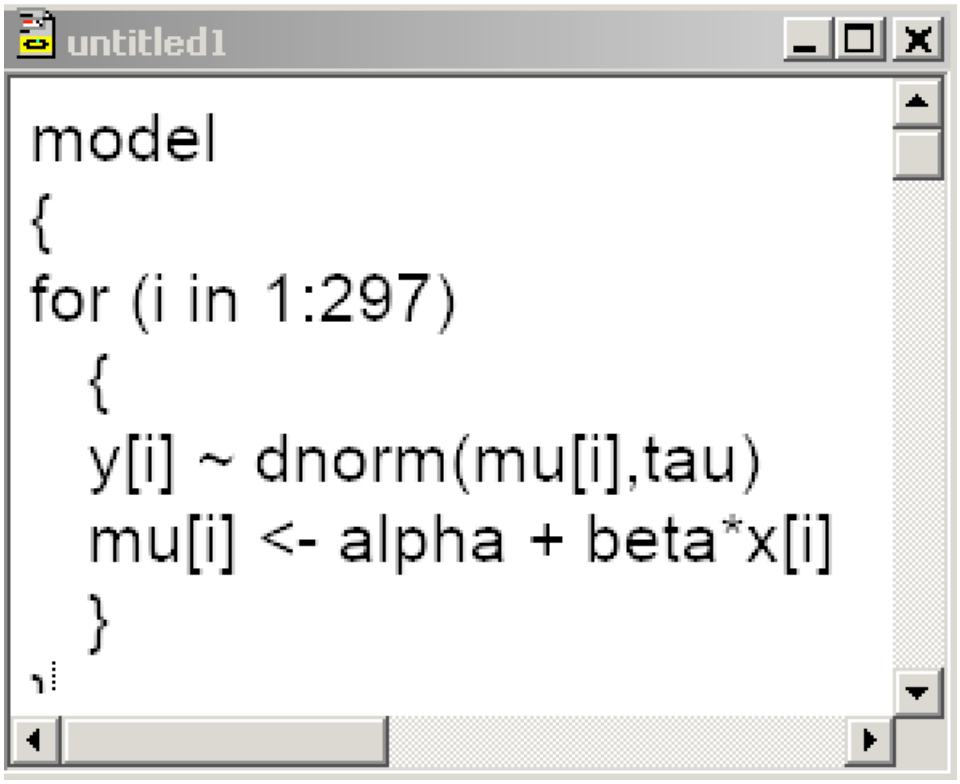


Step-by-step implementation of the simple regression model in OpenBUGS (cont'd)

- Step 3. Insert the distributional hypothesis on Y for each data unit i as follows:

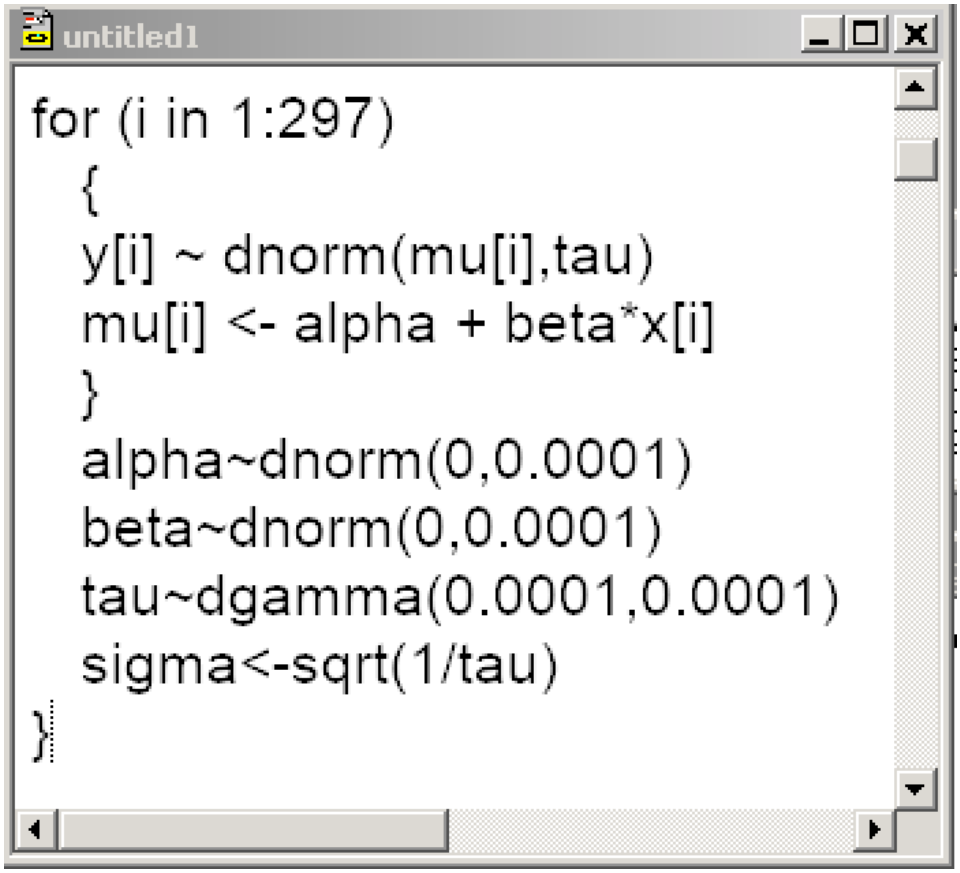


- Step 4. Insert the deterministic part of the model expressing the expected value for Y :



Step-by-step implementation of the simple regression model in OpenBUGS (cont'd)

- Step 5. Insert the vague priors for the nodes of interest (`alpha`, `beta` and `tau`) and the deterministic expression to get the estimation of the standard error (bear in mind that the variance parameter has to be obtained as the square root of the inverse):



- Step 6. Insert the inits for the notes of interest:

```
#init
list(alpha=28,beta=0,tau=1)
```

- Step 7. Insert the data using a list (on the original \texttt{odc} file for the homework they were already present).

Launching the chain after building of the model (1)

- Step 1: *Checking the model*. Open the "Specification Tool", select the "model" word in the `odc` file and click on "check model".
- Step 2: *Loading the data*. Select the "list" word referring to the dataset in the \texttt{odc} file and click on "load data".
- Step 3: *Compiling the model*. Click on "compile".
- Step 4: *Loading the inits*. Select the "list" word referring to the init in the \texttt{odc} file and click on "load inits".

On the bottom left corner of the screen "model is initialized" should appear.

Launching the chain after building of the model (2)

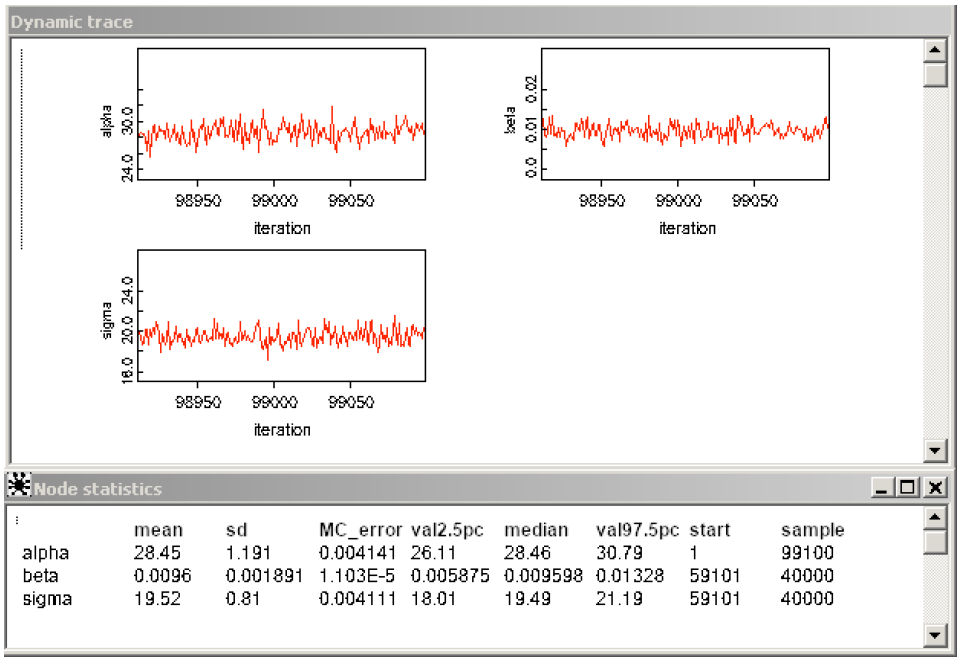
- Step 5: Monitoring the nodes (1). Open the "Sample Monitor Tool", type "alpha" in the "node" textbox and click on "set".
- Step 6: Monitoring the nodes (2). Repeat Step 5 for node "beta".
- Step 7: Monitoring the nodes (3). Repeat Step 5 for node "sigma".
- Step 8: Monitoring the nodes (4). Repeat Step 5 typing "*".

Launching the chain after building of the model (3)

- Step 9: Updating the chain (1). Open the "Update Tool", type "100000" in the "updates" textbox and click on "update"
- Step 10: Right after Step 9 click on "trace" in the "Sample Monitor Tool" to see the dynamic trace of the chain.
- Step 11: After the dynamic trace stops, click on "stats" to see the "Node statistics" window.

Simple regression model in OpenBUGS: the output

- From the "mean" column in the "Node statistics" windows the estimate of alpha is 28.45, the estimate for beta is 0.0096 and the estimate for sigma is 19.52. From the "Dynamic trace" window the chains seem well mixed.



Simple regression model in R and in Python: Final comparison

- Estimate for alpha is 28.45 in OpenBUGS and around 28.47 in Python (which uses the OLS method).
- Estimate for beta is 0.0096 in OpenBUGS and 0.0096 in Python (quite the same estimate!).
- Therefore results are very close. Not too bad!

Bayesian example

This example is taken from Spiegelhalter et al. (2004). Suppose a new HIV test is claimed to have "95% sensitivity and 98% specificity." In a population with an HIV prevalence of $1/1000$, what is the probability that a patient testing positive actually has HIV? We can use Bayes' theorem (3.1) to evaluate this.

Let $A = 1$ if the patient is truly HIV positive and $A = 0$ if they are truly HIV negative. Further, let $B = 1$ if they test positive and $B = 0$ if they test negative. The required probability is then $p(A = 1|B = 1)$. Now, "95% sensitivity" means that $p(B = 1|A = 1) = 0.95$, and "98% specificity" means that $p(B = 1|A = 0) = 1 - 0.98 = 0.02$. Writing $p(B = 1) = p(B = 1|A = 1)p(A = 1) + p(B = 1|A = 0)p(A = 0)$ and applying Bayes' theorem gives

$$p(A = 1|B = 1) = \frac{0.95 \times 0.001}{0.95 \times 0.001 + 0.02 \times 0.999} = 0.045.$$

Thus over 95% of those testing positive will, in fact, *not* have HIV!

This result generally comes as a surprise and illustrates that intuition is often poor when processing probabilistic evidence. The key issue is *how should this test result change our belief that a patient is HIV positive?* The disease prevalence can be thought of as the *prior* probability ($p = 0.001$) of having HIV; observing a positive result causes us to modify or update this to obtain a *posterior probability* of having HIV of $p = 0.045$ — hence the patient is 45 times more likely to have HIV after recording a positive test, but the absolute risk of HIV is still very small.

This result is perhaps better communicated by considering the expected status of a large population, say 100,000 people, in which 100 people are expected to be HIV+, of which 95 will test positive, and 99,900 will be HIV−, of which 1998 (2%) will also (erroneously) test positive. Thus, out of 2093 positive tests, only 95 (4.5%) will be truly HIV+ (Figure 3.1).

Bayesian example (cont'd)

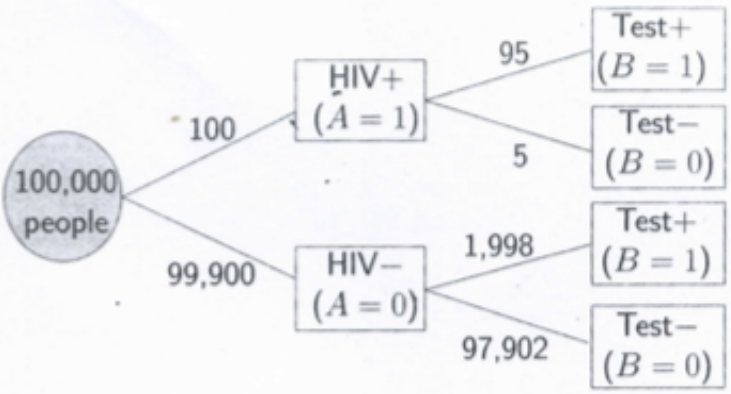


FIGURE 3.1
Bayes' theorem in HIV testing.

Example with the conjugacy property

- Recall the conjugacy property we saw in previous classes.
- Here is an example on how to build a model in the banking mortgage area:
- Suppose a branch of a bank received 20 requests of mortgages in the last month, 15 of them were successful.
- Suppose that in the past the "rate of acceptance" for mortgages (or, if you want, the rate of the entire bank) has been around 40.9% (9 out of 22).
- Construct an OpenBUGS model for the posterior acceptance rate using the beta-binomial conjugacy (*Suggestion*: remember that the expected value of a beta distribution with parameters α and β is $\frac{\alpha}{\alpha+\beta}$).

Example with the conjugacy property (2)

- *Log-likelihood*: is given by a binomial X random variable, i.e.:
  $x \sim \text{Bin}(\theta, n = 20)$.
- The actual sample is formed by $n = 20$ requests of mortgages, among which 15 were successful.
- In these terms, the "sample data" is simply $x = 15$.
- We have to put a prior on θ .
- Prior: If we want that the Bayes model is a a beta-binomial conjugacy, the prior must be a beta, so that also the posterior is a beta.
- Prior settings: If the past acceptance rate is $9/22$, then $\alpha = 9$ and $\beta = 13$. So the prior is
  $\theta \sim \text{Beta}(\alpha = 9, \beta = 13)$.

Example with the conjugacy property (3)

- The model will monitor the θ parameter and will be the following:

BetaBinomial

```
model{
theta ~ dbeta(9, 13) #Prior distribution for theta
y ~ dbin(theta, 20)
}

#init:
list(theta=0.5)
#"Data":
list(y=15)
```

- ...and will have the following output for θ, after $100,000$ updates:

Node statistics

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
theta	0.5715	0.07537	2.389E-4	0.4211	0.5729	0.7152	1	100000

- ...so the acceptance rate has passed from 40.9 % of the prior and 75 % ($15/20$) of the likelihood to 57.15 % in the posterior.

Example with logistic regression

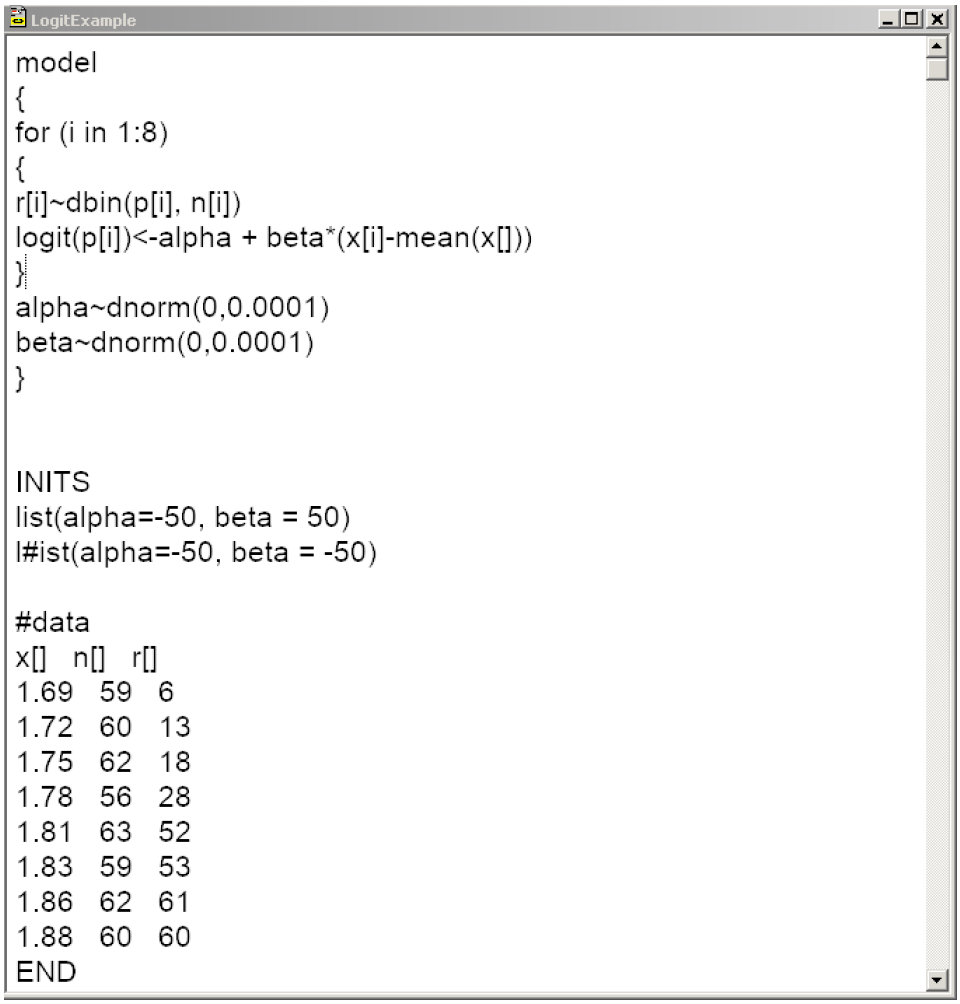
- Suppose you have the following data showing interest rates on ordinary accounts, the number of surveyed customers and "positive responses" with respect to open an account in 8 different bank branches offering 8 different interest rates (file `Branch-data.csv`):

Branch	Interest rate (x_i)	No. of customers (n_i)	No. of pos. responses (r_i)
1	1.69	59	6
2	1.72	60	13
3	1.75	62	18
4	1.78	56	28
5	1.81	63	52
6	1.83	59	53
7	1.86	62	61
8	1.88	60	60

- In `OpenBUGS` , model $p_i = P(Y = 1)$ (with $Y = 1$ denoting a positive response) as response variable of a logistic regression with independent variable given by (centered) x_i (i.e. $x_i - \bar{x}$).

Example with logistic

regression: OpenBUGS model (with data loaded from xls)

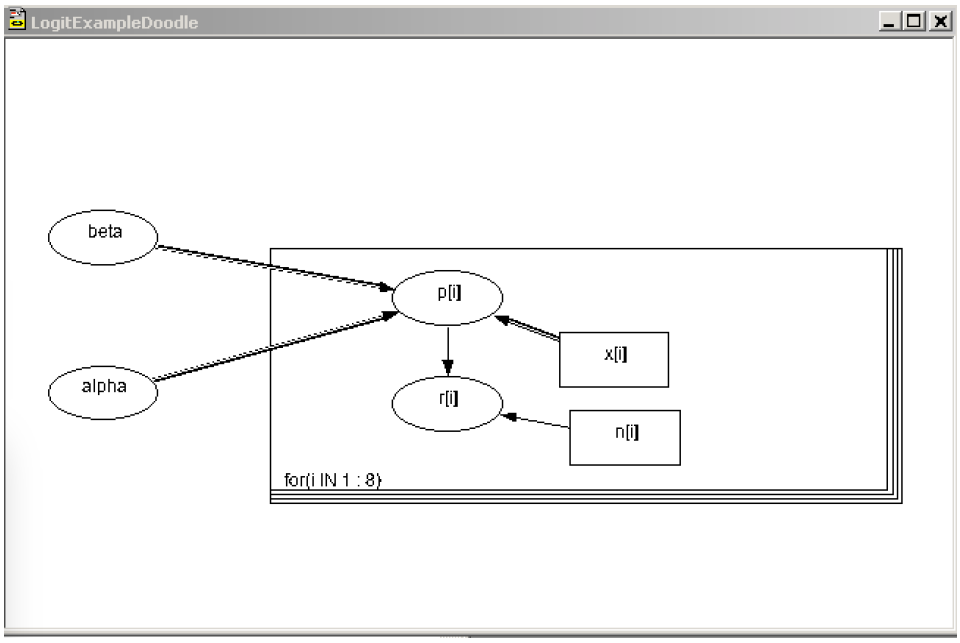


Example with logistic regression: OpenBUGS model (with data loaded from xls)

Node statistics

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
alpha	0.7666	0.1406	5.553E-4	0.4943	0.7647	1.05	1	100000
beta	34.7	2.937	0.01282	29.13	34.63	40.68	1	100000

Using Doodles



In [ ]:

The two-sample T-test on the difference between two means

- Let's $\mathbf{x}_1 = (x_{11}, \dots, x_{1n_1})$ and $\mathbf{x}_2 = (x_{21}, \dots, x_{2n_2})$ be two iid samples with $n_1 \neq n_2$.
- Consider the following statistic:

$$u_i := \bar{x}_1 - \bar{x}_2.$$

- This is the difference between two sample means.
- We can use this statistic to test whether the difference in the population means $\mu_1 - \mu_2$ is significantly different from zero.
- Therefore, we can write the following hypothesis test:

$$\begin{cases} H_0 \colon \mu_1 - \mu_2 = 0 \\ H_1 \colon \mu_1 - \mu_2 \neq 0 \end{cases}$$

and work with standard frequentist analysis.

The two-sample T-test on the difference between two means: Bayesian approach + OpenBUGS implementation

- Let's have the following two iid samples:

  $\mathbf{x}_1$=(118.86,122.02,118.04,127.52,122.59,118.10,123.20,124.18

  $\mathbf{x}_2$=(125.86,125.23,121.55,123.94,126.71,127.01,122.05,124.67
  representing $n_1 = 10$ measurements of cholesterol values in 10 men and $n_2 = 11$ cholesterol values in 11 women. Cholesterol is measured in mg/dl.

- Samples are supposed to be normally distributed around the mean values for each group in the population, so that we can write:

  $$x_{1i} \sim N(\mu_1, \sigma^2),$$

  and

  $$x_{2i} \sim N(\mu_2, \sigma^2),$$

  therefore hypothesizing the same variance.

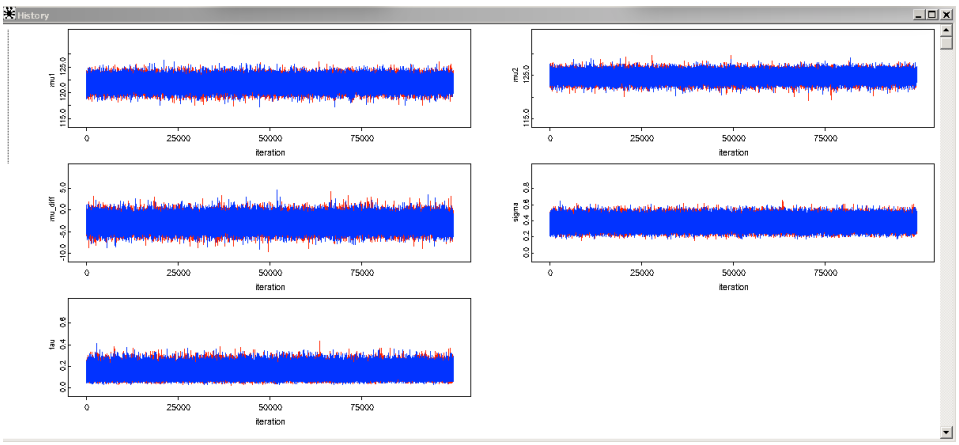
A T-test example

ANOVA

```
model
{
   for (i in 1:N1)
   {
      x1[i] ~ dnorm(mu1,tau)
   }
   for (i in 1:N2)
   {
      x2[i] ~ dnorm(mu2,tau)
   }
#Prior for tau
unif1 <- exp(-20)
unif2 <- exp(20)
tau~dunif(unif1, unif2)
sigma <- sqrt(tau)
#Very vague priors for mus
mu1 ~ dnorm(0, 0.000001)
mu2 ~ dnorm(0, 0.000001)
mu_diff <- mu1-mu2
}
#initial values
list(mu1 = 122, mu2=125, tau=0.2)
list(mu1 = 121, mu2=122, tau=0.1)
#data
list(x1=c(118.86, 122.02, 118.04, 127.52, 122.59, 118.10, 123.20, 124.18, 123.55,
120.11),
x2=c(125.86, 125.23, 121.55, 123.94, 126.71, 127.01, 122.05, 124.67, 120.76, 125.00,
129.22),N1=10, N2=11)
```

A T-test example (cont'd)

- The chains are very well mixed:



A T-test example (cont'd)

- The 2.5 and 97.5 percentile range does not include zero, so we can reject the null hypothesis:

Node statistics

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
mu1	121.8	0.8844	0.001985	120.1	121.8	123.6	1	200000
mu2	124.7	0.8424	0.001873	123.1	124.7	126.4	1	200000
mu_diff	-2.911	1.222	0.002846	-5.331	-2.909	-0.4999	1	200000
sigma	0.3722	0.05772	1.424E-4	0.2638	0.3706	0.4899	1	200000
tau	0.1419	0.04375	1.069E-4	0.06957	0.1374	0.24	1	200000

A T-test example (cont'd)

- The two priors given in the first model above do not allow for letting the two means be *exactly* equal to zero.
- We may think μ_1 might exactly equal μ_2 .
- A way to do so is proposed by Brewer (page 79).
- So we can define the prior for μ_1 as above, but as for μ_2 we can find a way to let it have a 50% chance to be equal to μ_1 .
- When μ_2 is not equal to μ_1 we give it artificially a “bi-exponential” distribution centered around μ_1 .

ANOVA2

```
model{
  for (i in 1:N1) {
    x1[i] ~ dnorm(mu1,tau)}
  for (j in 1:N2) {
    x2[j] ~ dnorm(mu2,tau)}
#Prior for tau
unif1 <- exp(-20)
unif2 <- exp(20)
tau~dunif(unif1, unif2)
sigma <- sqrt(tau)
#Very vague prior for mu1
mu1 ~ dnorm(0, 0.000001)
#mu2 ~ dnorm(0, 0.000001)
u~ dunif(-1,1)
L <- 5
size_of_difference <- step(u) * (-L * log(1 - u))
C~ dbin(0.5,1)
difference <- (2*C - 1) * size_of_difference
mu2 <- mu1 + difference
mu_diff <- mu1-mu2
}
#initial values
list(mu1 = 122, u=0, tau=0.2, C = 0)
list(mu1 = 121, u=0.01, tau=0.1, C=0)
#data
list(x1=c(118.86, 122.02, 118.04, 127.52, 122.59, 118.10, 123.20,
124.18, 123.55, 120.11),
x2=c(125.86, 125.23, 121.55, 123.94, 126.71, 127.01, 122.05,
124.67, 120.76, 125.00, 129.22),N1=10, N2=11)
```

In [2]:

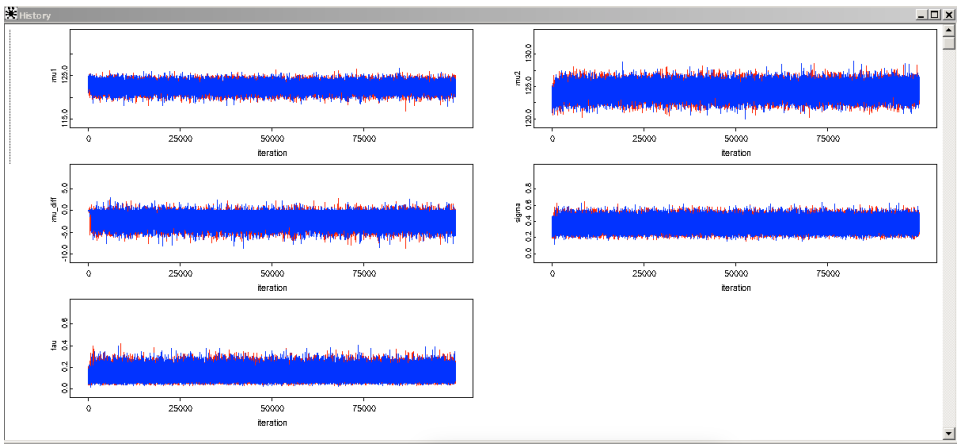
```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
np.random.seed(19680801)
n_bins = 50
L = 5
dimvect =1000
values = np.zeros(dimvect)
mu1 = np.random.normal(0,1000**4,size = dimvect)

for i in range(0,dimvect):
    uniform1 = np.random.uniform(low = -1,high = 1,size = 1)
    if (uniform1 < 0):
        size_of_difference = L * np.log(1-uniform1)
    else:
        size_of_difference = (-1)*(L * np.log(1-uniform1))
    C = np.random.binomial(1,0.5,1)
    difference = (2*C - 1) * size_of_difference
    values[i] = difference

```

```
mu2 = mu1 + values
df = pd.DataFrame({ 'values' : values,
                    'mu1' : mu1,
                    'mu2' : mu2,
                    'mu2-mu1' : mu2-mu1,
                    })
fig, (ax1, ax2, ax3,ax4) = plt.subplots(1, 4)
ax1.hist(df['values'], bins=n_bins)  # no
ax2.hist(df['mu1'], bins=n_bins)   # nothing
ax3.hist(df['mu2'], bins=n_bins)
ax4.hist(df['mu2-mu1'], bins=n_bins)


#plt.hist(values, bins = [-40, -30, -20, -10, 0, 10, 20, 30, 40
#plt.hist(mu1, bins = 50)
#par(mfrow=c(2,2))
#hist(values)
#hist(mu1)
#hist(mu2)
#hist(mu1-mu2)

```

Out[2]:
```
(array([  1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   1.,   0.,   1.,   2.,   2.,   5.,   4.,   8.,
          4.,  11.,  15.,   9.,  16.,  33.,  65., 172., 146., 125., 171.,
        110.,  34.,  25.,  10.,  12.,   6.,   2.,   1.,   2.,   2.,   0.,
          3.,   1.,   0.,   0.,   0.,   1.]),
 array([-45.49597168, -44.03353027, -42.57108887, -41.10864746,
        -39.64620605, -38.18376465, -36.72132324, -35.25888184,
        -33.79644043, -32.33399902, -30.87155762, -29.40911621,
        -27.9466748 , -26.4842334 , -25.02179199, -23.55935059,
        -22.09690918, -20.63446777, -19.17202637, -17.70958496,
        -16.24714355, -14.78470215, -13.32226074, -11.85981934,
        -10.39737793,  -8.93493652,  -7.47249512,  -6.01005371,
         -4.5476123 ,  -3.0851709 ,  -1.62272949,  -0.16028809,
          1.30215332,   2.76459473,   4.22703613,   5.68947754,
          7.15191895,   8.61436035,  10.07680176,  11.53924316,
         13.00168457,  14.46412598,  15.92656738,  17.38900879,
         18.8514502 ,  20.3138916 ,  21.77633301,  23.23877441,
         24.70121582,  26.16365723,  27.62609863]),
 <a list of 50 Patch objects>)
```

A T-test example (cont'd)

- Again the chains are very well mixed:



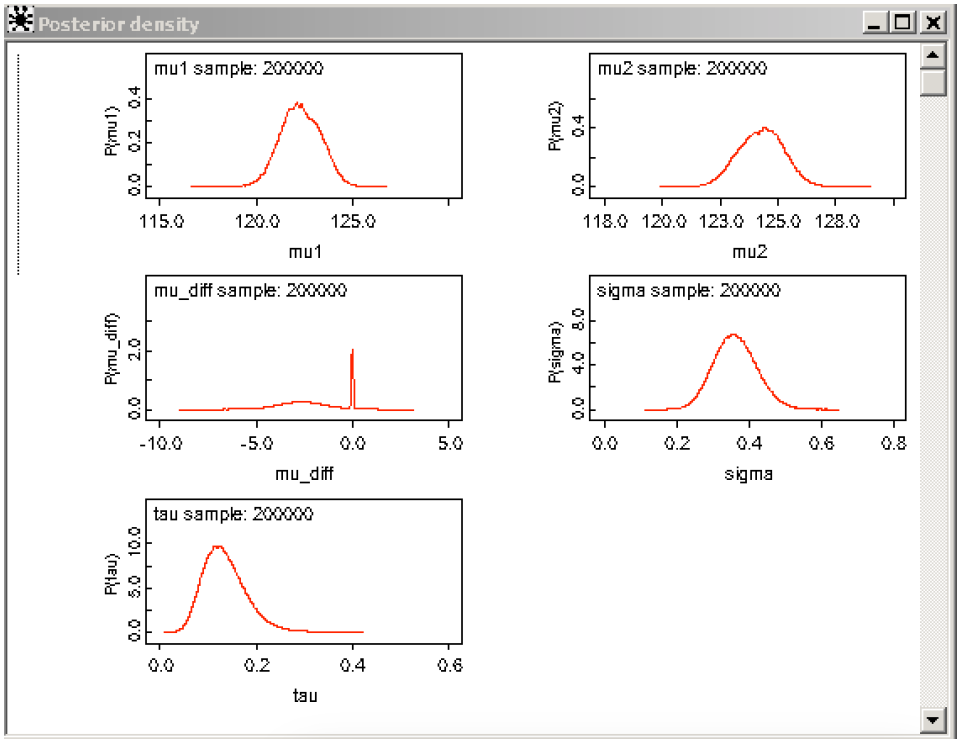
A T-test example (cont'd)

- This time the 97.5 percentile is equal to zero, so we cannot reject the nullnull hypothesis:

Node statistics

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
mu1	121.8	0.8844	0.001985	120.1	121.8	123.6	1	200000
mu2	124.7	0.8424	0.001873	123.1	124.7	126.4	1	200000
mu_diff	-2.911	1.222	0.002846	-5.331	-2.909	-0.4999	1	200000
sigma	0.3722	0.05772	1.424E-4	0.2638	0.3706	0.4899	1	200000
tau	0.1419	0.04375	1.069E-4	0.06957	0.1374	0.24	1	200000

- However, from inspecting the posterior density for mu_diff we notice that we now allow the means to be equal at 50%.



A T-test example (cont'd): A Bayesian hierarchical model

- The last prior given before is not completely understandable.
- If we're comparing the two samples we can think about the two parameters μ_1 and μ_2 not to be equal, but *similar* in value.
- In other words, we shouldn't worry so much about the prior probablity of $\mu_1 = \mu_2$, but we should at least make sure there's a moderate prior probability that $\mu_1 \approx \mu_2$.
- One way we could do this is by applying a normal prior to both μ_1 and μ_2 with some mean (let's call it the *overall mean*) and some standard deviation (let's call it the *divergence").
- That way, μ_1 and μ_2 would both be likely to be somewhere *around* the overall mean, and they would likely be different by roughly the size of the divergence.
- The challenge now seems to be the choice of appropriate values for the overall mean and the

divergence.

- Fortunately we can do this by applying priors for them instead.
- This is an example of a *hierarchical model* to be applied even in a test situation.
- Again, recall that in a hierarchical model, instead of directly assigning priors to our parameters, we imagine that we knew the values of some other parameters (called “hyperparameters”), and assign our prior for the parameters given the hyperparameters.
- Then we assign a prior for they hyperparameters as well, to complete the model.

A T-test example: A Bayesian hierarchical model (cont'd)

- The model could be the following:

ANOVA3

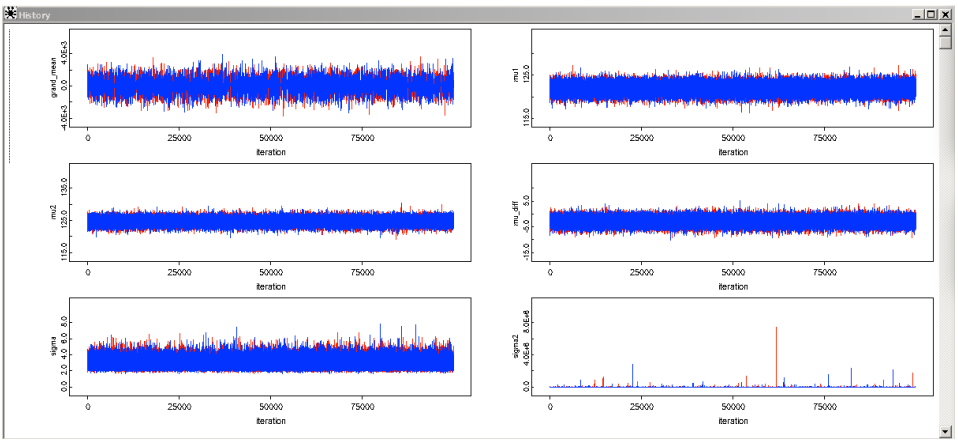
```
model{
  for (i in 1:N1) {
    x1[i] ~ dnorm(mu1,tau)}
  for (j in 1:N2) {
    x2[j] ~ dnorm(mu2,tau)}
#Prior for tau

sigma ~ dunif(0, 100000000)
tau <- 1/pow(sigma,2)
#Hyperpriors for the parameters of the likelihood
mu1 ~ dnorm(grand_mean, tau2)
mu2 ~ dnorm(grand_mean, tau2)
#Hyperparameters to build up a hierarchical model
grand_mean ~ dnorm(0, 0.000001)
sigma2~dunif(0, 100000000)
tau2 <- 1/pow(sigma2,2)
mu_diff <- mu1-mu2
}
#initial values
list(mu1 = 122, mu2 = 124, sigma=3, grand_mean = 106, sigma2 =
2376)
list(mu1 = 121, mu2 = 125, sigma=3.1, grand_mean = 108, sigma2
= 2375)
#data
list(x1=c(118.86, 122.02, 118.04, 127.52, 122.59, 118.10, 123.20,
124.18, 123.55, 120.11),
x2=c(125.86, 125.23, 121.55, 123.94, 126.71, 127.01, 122.05,
124.67, 120.76, 125.00, 129.22),N1=10, N2=11)
```

A T-test example (cont'd): Comments on the implementation of the Bayesian hierarchical model

- With hierarchical models, the choice of priors is important.
- Vague priors for location parameters can be from the normal "family".
- Vage priors for scale parameters can be uniforms, gammas, etc.; in any case must have positive values.
- In this case the choice of initial values is crucial, as there can be no convergence even after hundreds of thousands iterations.
- Or, if we generate these initial values randomly, we have to cut the left-hand side of the chains in an approriate way (by using "beg" in OpenBUGS).

A T-test example: A Bayesian hierarchical model (cont'd)

- Again the chains are very well mixed:



A T-test example (cont'd)

- We can reject the null hypothesis:

Node statistics

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
grand_mean	106.5	368.6	0.2341	-823.8	123.1	918.2	100000	2800002
mu1	121.9	0.9686	7.263E-4	120.0	121.9	123.8	100000	2800002
mu2	124.7	0.9242	7.102E-4	122.8	124.7	126.5	100000	2800002
mu_diff	-2.808	1.348	0.001144	-5.465	-2.817	-0.1073	100000	2800002
sigma	2.998	0.5349	4.1E-4	2.168	2.925	4.246	100000	2800002
sigma2	2376.0	153600.0	241.4	1.159	68.96	7489.0	100000	2800002

Exchangeability

- Assume you have n units in your sample indexed by i .
- Each of them could depend on a parameter θ_i .
- If we assume that under broad conditions (de Finetti (1931)) these units are *similar* and we can also *exchange* their labels, and each of their θ_i depends on a population parameter ϕ , we talk about *exchengeability*.
- In this way we think about the θ_i as being *similar* but not equal.
- Example. Suppose we want to monitor the results of the same operation carried out in multiple hospitals and be interested in the mortility rates of this operation across the multiple hospitals.
- Supposing that hospitals A, B, C, have certain rates, you can think of the hospital D to have a mortality rate that is similar to those of other hospitals.
- This is reasonable and we can think that it is unlikely that all the hospitals have the *same* mortality rate, but it is plausible to think about

similar *mortality rates*.

- So, our natural invlination would be to assume something in between a situation where the unit-specific parameters are identical and a situation where all the units are completely identical.

Exchangeability (cont'd): schematic

- We have here a basic hierarchical model (left), an independent-parameters model (middle) and an identical-parameters model (right):

Bayesian hierarchical model with hospitals data

- Consider these data referring to mortality in England' most important 50 hospitals (more than 60,000 admissions each year - data are referred to 2018-19):

	A	B	C	D	E	F	G	H
1	Location	Site name			SHMI value	SHMI banding	Number of spells	Observed deaths
2	CAMBRIDGE	ADDENBROOKE'S HOSPITAL			0.8351	3	63,170	2,100
3	LONDON	THE ROYAL LONDON HOSPITAL			0.8420	3	68,470	1,170
4	BASILDON	BASILDON UNIVERSITY HOSPITAL			1.0076	2	74,710	1,965
5	LONDON	QUEEN'S HOSPITAL			0.9819	2	66,805	2,220
6	LIVERPOOL	UNIVERSITY HOSPITAL AINTREE			0.9456	2	63,215	2,045
7	BOLTON	ROYAL BOLTON HOSPITAL			1.1800	1	60,425	1,775
8	BRADFORD	BRADFORD ROYAL INFIRMARY			0.8768	2	64,940	1,425
9	STEVENAGE	LISTER HOSPITAL			0.9870	2	65,270	2,090
10	BLACKBURN	ROYAL BLACKBURN HOSPITAL			1.0357	2	64,710	2,605
11	PORTSMOUTH	FRIMLEY PARK HOSPITAL			0.9468	2	65,680	1,780
12	WREXHAM	WEXHAM PARK HOSPITAL			0.9738	2	62,700	1,765
13	GLOUCESTER	GLOUCESTERSHIRE ROYAL HOSPITAL			1.0887	2	66,840	1,895
14	SWINDON	THE GREAT WESTERN HOSPITAL			0.9222	2	62,505	1,690
15	LONDON	ST THOMAS' HOSPITAL			0.7628	3	74,845	1,075
16	HULL	HULL ROYAL INFIRMARY			1.0606	2	64,510	2,480
17	LONDON	KING'S COLLEGE HOSPITAL (DENMARK HILL)			0.8937	2	67,460	1,530
18	PRESTON	ROYAL PRESTON HOSPITAL			0.9987	2	63,815	1,535
19	LEEDS	ST JAMES'S UNIVERSITY HOSPITAL			1.0840	2	67,775	3,185
20	HARROW	NORTHWICK PARK HOSPITAL			0.8203	3	79,055	1,745
21	LUTON	LUTON & DUNSTABLE HOSPITAL			1.0258	2	82,050	1,725
22	MANCHESTER	WYTHENSHAWE HOSPITAL			0.9229	2	69,190	1,740
23	WAKEFIELD	PINDERFIELDS GENERAL HOSPITAL			1.0680	2	83,275	2,550
24	NORWICH	NORFOLK & NORWICH UNIVERSITY HOSPITAL			1.0989	2	89,185	3,460
25	BRISTOL	SOUTHMEAD HOSPITAL			0.9128	2	89,860	2,230
26	NORTHAMPTON	NORTHAMPTON GENERAL HOSPITAL (ACUTE)			0.9775	2	75,375	1,900
27	CRAMLINGTON	NORTHUMBRIA SPECIALIST EMERGENCY CARE HOSPITAL			0.9909	2	69,535	2,745
28	NOTTINGHAM	NOTTINGHAM UNIVERSITY HOSPITALS NHS TRUST - QUEEN'S MEDICAL CENTRE CAMPUS			1.1068	2	90,910	2,885
29	OXFORD	JOHN RADCLIFFE HOSPITAL			0.8970	2	89,025	2,075
30	PORTSMOUTH	QUEEN ALEXANDRA HOSPITAL			1.0300	2	83,660	3,005
31	LONDON	ROYAL BERKSHIRE HOSPITAL			1.0897	2	68,980	2,030
32	TRURO	ROYAL CORNWALL HOSPITAL (TRELISKE)			0.9461	2	69,510	2,280
33	EXETER	ROYAL DEVON & EXETER HOSPITAL (WONFORD)			1.0355	2	62,455	2,175
34	BATH	ROYAL UNITED HOSPITAL			0.9833	2	60,015	1,905
35	MIDDLESBROUGH	THE JAMES COOK UNIVERSITY HOSPITAL			1.0795	2	76,160	2,190
36	LONDON	ST GEORGE'S HOSPITAL (TOOTING)			0.8187	3	88,440	1,915
37	PRESCOT	WHISTON HOSPITAL			1.0429	2	79,720	2,320
38	DUDLEY	RUSSELLS HALL HOSPITAL			1.1290	2	62,385	2,390
39	NEWCASTLE UPON TYNE	THE ROYAL VICTORIA INFIRMARY			0.9650	2	73,300	1,505
40	WOLVERHAMPTON	NEW CROSS HOSPITAL			1.1524	2	68,545	2,670
41	SOUTHAMPTON	SOUTHAMPTON GENERAL HOSPITAL			0.8633	2	81,145	2,490
42	SUTTON COLDFIELD	GOOD HOPE HOSPITAL			0.9514	2	65,975	1,730
43	BIRMINGHAM	HEARTLANDS HOSPITAL			0.9586	2	99,255	2,035
44	BIRMINGHAM	QUEEN ELIZABETH HOSPITAL BIRMINGHAM			1.0887	2	75,165	2,785
45	COVENTRY	UNIVERSITY HOSPITAL (COVENTRY)			1.1001	2	96,970	2,860
46	DERBY	ROYAL DERBY HOSPITAL			0.9788	2	85,805	3,185
47	LEICESTER	LEICESTER ROYAL INFIRMARY			1.0729	2	89,795	3,185
48	STOKE-ON-TRENT	ROYAL STOKE UNIVERSITY HOSPITAL			1.0061	2	132,805	3,745
49	PLYMOUTH	DERRIFORD HOSPITAL			1.1558	2	72,420	2,600
50	WIRRAL	ARROWE PARK HOSPITAL			1.0752	2	63,760	2,190
51	YORK	YORK HOSPITAL			0.9591	2	60,295	1,885

Bayesian hierarchical

model with hospitals data (cont'd)

- We can use a binomial distribution for the data as usual, but modeling θ_i hierarchically using a link function (a logit function on θ_i) in combination with a normal distribution, that is:

$$\text{logit}\theta_i \sim N(\mu, \omega^2),$$

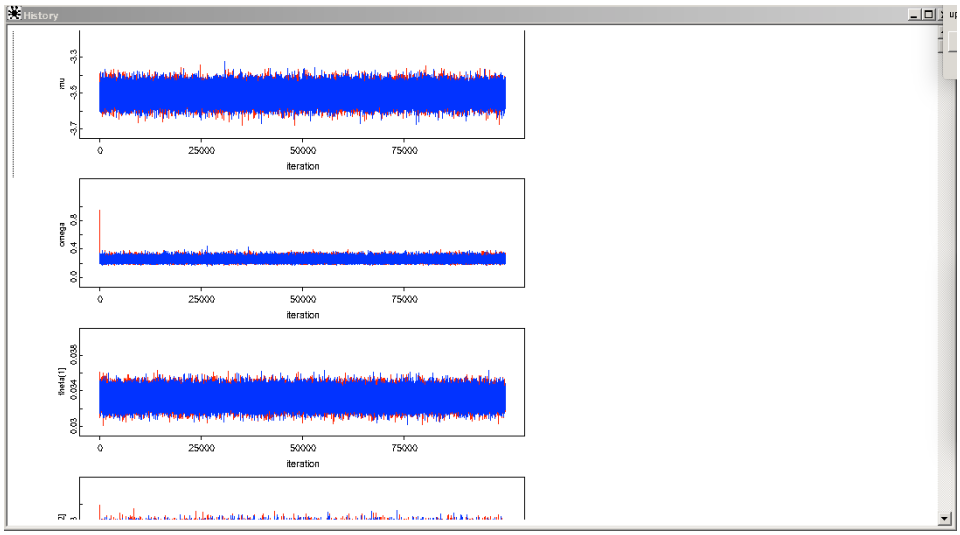
and choose vague hyperpriors for μ and ω .

- Therefore the model in OpenBUGS will appear as follows:

```
model{for (i in 1:50){
   y[i] ~ dbin(theta[i], n[i])
   logit(theta[i]) <- logit.theta[i]
   logit.theta[i] ~ dnorm(mu, inv.omega.squared)}
   inv.omega.squared <- 1/pow(omega,2)
   omega ~ dunif(0,100)
   mu ~ dunif(-100,100)}
#Data
list(y=c(2100, 1170, 1965, 2220, 2045, 1775, 1425, 2090, 2605,
1780, 1765, 1895, 1690, 1075, 2480, 1530, 1535, 3185, 1745,
1725, 1740, 2550, 3460, 2230, 1900, 2745, 2885, 2075, 3005,
2030, 2280, 2175, 1905, 2190, 1915, 2320, 2390, 1505, 2670,
2490, 1730, 2035, 2785, 2860, 3185, 3185, 3645, 2600, 2190,
1885),
n=c(63170, 68470, 74710, 66805, 63215, 60425, 64940, 65270,
64710, 65680, 62700, 66840, 62505, 74845, 64510, 67460, 63815,
67775, 79055, 82050, 69190, 83275, 89185, 89860, 75375, 69535,
90910, 89025, 83660, 68980, 69510, 62455, 60015, 76160, 88440,
79720, 62385, 73300, 68545, 81145, 65975, 99255, 75165, 96970,
85805, 89795, 132805, 72420, 63670, 60295))
#Initial values
list(mu = -2, omega = 0.1,
logit.theta = c(0.3,
0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,
0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,
0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3))
list(mu = -3, omega = 0.3,
logit.theta= c(0.3,
0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,
0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,
0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3))
```

Bayesian hierarchical model with hospitals data (cont'd)

- Again the chains are very well mixed:



Bayesian hierarchical model with hospitals data (cont'd)

- Mortality rates estimates for these 50 hospitals are the following:

omega	0.2501	0.02643	6.461E-5	0.2047	0.2478	0.3079	1	200000
theta[1]	0.03321	7.113E-4	1.479E-6	0.03183	0.03321	0.03462	1	200000
theta[2]	0.01722	4.946E-4	1.142E-6	0.01626	0.01721	0.0182	1	200000
theta[3]	0.02632	5.851E-4	1.313E-6	0.02519	0.02632	0.02748	1	200000
theta[4]	0.0332	6.905E-4	1.583E-6	0.03186	0.03319	0.03456	1	200000
theta[5]	0.03232	7.002E-4	1.615E-6	0.03097	0.03232	0.03371	1	200000
theta[6]	0.02937	6.838E-4	1.491E-6	0.02805	0.02936	0.03073	1	200000
theta[7]	0.02202	5.734E-4	1.3E-6	0.02091	0.02201	0.02315	1	200000
theta[8]	0.032	6.862E-4	1.543E-6	0.03066	0.03199	0.03335	1	200000
theta[9]	0.04017	7.703E-4	1.654E-6	0.03868	0.04017	0.0417	1	200000
theta[10]	0.02712	6.315E-4	1.482E-6	0.02589	0.02711	0.02837	1	200000
theta[11]	0.02816	6.571E-4	1.417E-6	0.02688	0.02816	0.02946	1	200000
theta[12]	0.02836	6.418E-4	1.375E-6	0.02711	0.02835	0.02963	1	200000
theta[13]	0.02706	6.46E-4	1.53E-6	0.02581	0.02705	0.02834	1	200000
theta[14]	0.01452	4.346E-4	9.61E-7	0.01369	0.01451	0.01539	1	200000
theta[15]	0.03837	7.539E-4	1.83E-6	0.03691	0.03837	0.03987	1	200000
theta[16]	0.02274	5.716E-4	1.242E-6	0.02164	0.02274	0.02387	1	200000
theta[17]	0.0241	6.039E-4	1.281E-6	0.02294	0.0241	0.0253	1	200000
theta[18]	0.04687	8.116E-4	1.813E-6	0.04529	0.04687	0.04847	1	200000
theta[19]	0.02213	5.188E-4	1.137E-6	0.02113	0.02213	0.02316	1	200000
theta[20]	0.02109	4.999E-4	1.097E-6	0.02013	0.02109	0.02208	1	200000
theta[21]	0.02518	5.944E-4	1.261E-6	0.02403	0.02518	0.02636	1	200000
theta[22]	0.03061	5.945E-4	1.292E-6	0.02945	0.03061	0.03179	1	200000
theta[23]	0.03874	6.432E-4	1.405E-6	0.03749	0.03874	0.04001	1	200000
theta[24]	0.02485	5.174E-4	1.16E-6	0.02384	0.02484	0.02587	1	200000
theta[25]	0.02524	5.688E-4	1.257E-6	0.02414	0.02524	0.02637	1	200000
theta[26]	0.0394	7.371E-4	1.635E-6	0.03796	0.0394	0.04086	1	200000
theta[27]	0.03172	5.8E-4	1.327E-6	0.03059	0.03172	0.03286	1	200000
theta[28]	0.02335	5.037E-4	1.179E-6	0.02237	0.02335	0.02435	1	200000
theta[29]	0.03587	6.408E-4	1.513E-6	0.03463	0.03587	0.03715	1	200000
theta[30]	0.02943	6.405E-4	1.56E-6	0.02819	0.02942	0.0307	1	200000
theta[31]	0.03277	6.734E-4	1.507E-6	0.03147	0.03277	0.0341	1	200000
theta[32]	0.03478	7.323E-4	1.648E-6	0.03335	0.03477	0.03623	1	200000
theta[33]	0.03172	7.152E-4	1.568E-6	0.03033	0.03171	0.03313	1	200000
theta[34]	0.02876	6.027E-4	1.309E-6	0.02758	0.02875	0.02994	1	200000
theta[35]	0.02171	4.882E-4	1.088E-6	0.02076	0.02171	0.02268	1	200000
theta[36]	0.02911	5.927E-4	1.314E-6	0.02795	0.0291	0.03028	1	200000
theta[37]	0.03823	7.671E-4	1.644E-6	0.03674	0.03823	0.03975	1	200000
theta[38]	0.02061	5.214E-4	1.238E-6	0.01961	0.02061	0.02164	1	200000
theta[39]	0.03888	7.375E-4	1.688E-6	0.03745	0.03888	0.04034	1	200000
theta[40]	0.03067	6.028E-4	1.354E-6	0.02951	0.03067	0.03187	1	200000
theta[41]	0.02625	6.186E-4	1.455E-6	0.02505	0.02624	0.02748	1	200000
theta[42]	0.02056	4.486E-4	9.885E-7	0.01969	0.02056	0.02145	1	200000
theta[43]	0.037	6.859E-4	1.597E-6	0.03566	0.03699	0.03835	1	200000
theta[44]	0.02949	5.418E-4	1.195E-6	0.02844	0.02949	0.03056	1	200000
theta[45]	0.03707	6.434E-4	1.458E-6	0.03582	0.03707	0.03835	1	200000
theta[46]	0.03543	6.146E-4	1.371E-6	0.03424	0.03543	0.03665	1	200000
theta[47]	0.02745	4.452E-4	9.298E-7	0.02659	0.02745	0.02833	1	200000
theta[48]	0.03585	6.888E-4	1.581E-6	0.03452	0.03585	0.03721	1	200000
theta[49]	0.03435	7.191E-4	1.608E-6	0.03295	0.03435	0.03577	1	200000
theta[50]	0.03125	7.053E-4	1.559E-6	0.02988	0.03124	0.03264	1	200000

More complex models

Journal of the Royal Statistical Society
SERIES A Statistics in Society

J. R. Statist. Soc. A (2011)
174, *Part* 1, *pp.* 31–50

Modelling bias in combining small area prevalence estimates from multiple surveys

Giancarlo Manzi

Medical Research Council Biostatistics Unit, Cambridge, UK, and Università degli Studi di Milano, Italy

David J. Spiegelhalter and Rebecca M. Turner,

Medical Research Council Biostatistics Unit, Cambridge, UK

Julian Flowers

Eastern Region Public Health Observatory, Cambridge, UK

and Simon G. Thompson

Medical Research Council Biostatistics Unit, Cambridge, UK

[Received February 2009. Revised February 2010]

Summary. Combining information from multiple surveys can improve the quality of small area estimates. Customary approaches, such as the multiple-frame and statistical matching methods, require individual level data, whereas in practice often only multiple aggregate estimates are available. Commercial surveys usually produce such estimates without clear description of the methodology that is used. In this context, bias modelling is crucial, and we propose a series of Bayesian hierarchical models which allow for additive biases. Some of these models can also be fitted in a classical context, by using a mixed effects framework. We apply these methods to obtain estimates of smoking prevalence in local authorities across the east of England from seven surveys. All the surveys provide smoking prevalence estimates and confidence intervals at the local authority level, but they vary by time, sample size and transparency of methodology. Our models adjust for the biases in commercial surveys but incorporate information from all the sources to provide more accurate and precise estimates.

Keywords: Bias modelling; Hierarchical models; Meta-analysis; Mixed effects models; Multiple survey data; Small area estimation; Smoking prevalence

1. Introduction

More complex models (cont'd)

The basic model proposed is

$$\begin{aligned} y_{ij}|\delta_{ij} &\sim N(\theta_i + \delta_{ij}, \sigma_{ij}^2), \\ \delta_{ij} &\sim N(\mu_j, \tau_j^2). \end{aligned} \tag{1}$$

We shall in general adopt a Bayesian approach with vague priors

$$\begin{aligned} \mu_j &\sim N(0, 100^2), \\ \tau_j &\sim \text{Unif}(0, 100) \end{aligned}$$

where μ_j and τ_j^2 represent respectively the mean bias and the bias variance for data source j.

Writing model (1) marginally, we obtain

$$y_{ij} \sim N(\theta_i + \mu_j, \sigma_{ij}^2 + \tau_j^2) \tag{2}$$

and an identifiability issue becomes obvious because, for example, all θ_i could be increased by an arbitrary constant and all μ_j decreased by the same constant without changing the fit of the model; we note that model (1) resembles a weighted two-way analysis of variance (ANOVA) (Eberly and Carlin, 2000). To implement the model, we therefore need to provide additional information about the θs in addition to equations (1). We introduce an overall smoking prevalence for the east of England and provide this parameter on the basis of external information. It is taken as 23% from the UK General Household Survey for the east of England in 2005 (Goddard, 2006): $\bar{\theta} = 23$, i.e.

$$\begin{aligned} \theta_i &\sim \text{Unif}(0, 100), \qquad i = 1, \ldots, 47, \\ \theta_{48} &= \bar{\theta} \times 48 - \sum_{i=1}^{47} \theta_i. \end{aligned}$$

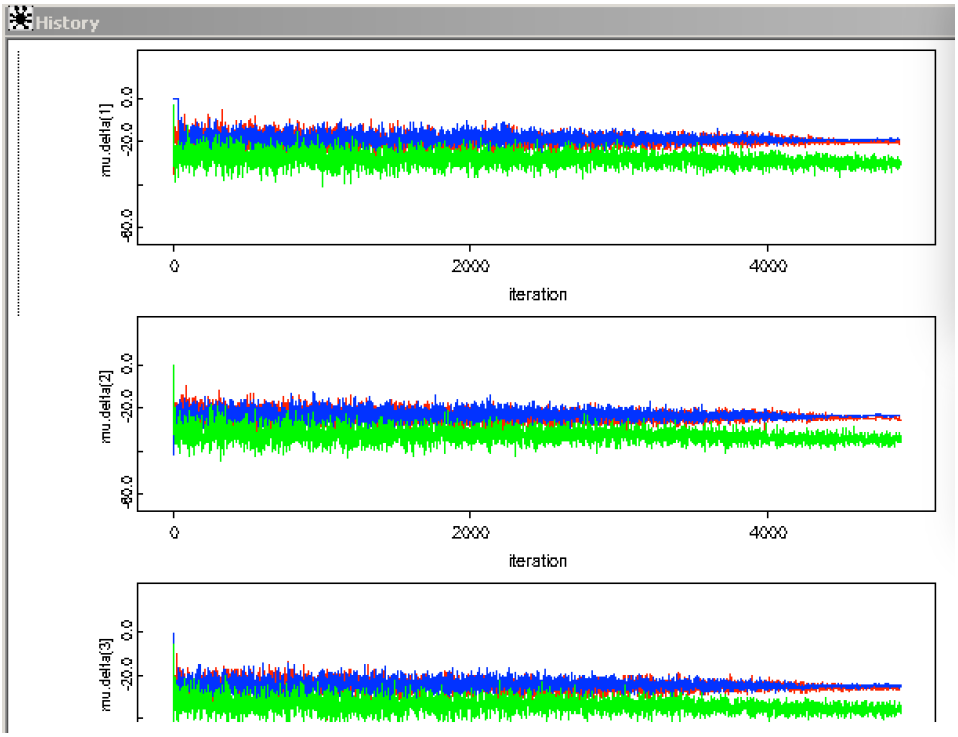
We refer to this assumption, together with equations (1), as *model 1*.

More complex models (cont'd)

Lecture12ExampleJRSSA

```
 model{

# Data sources indexed by j, areas indexed by i
for (j in 1:m) {
   for (i in 1:n) {
      y[i,j] ~ dnorm(mu[i,j],tau.y[i,j])
      mu[i,j] <- theta[i] + delta[i,j]
      # Source-specific bias represented by delta
      delta[i,j] ~ dnorm(mu.delta[j],tau.delta[j])
      tau.y[i,j] <- 1/pow(sd.y[i,j],2)
      sd.y[i,j] <- (uci[i,j]-lci[i,j])/(2*1.96)
      }
     mu.delta[j] ~ dnorm(0,0.0001)
     mu.delta.adj[j]<-mu.delta[j]+thetabar23
     #mu.delta.adj[j]<-mu.delta[j]-thetabar+23
     tau.delta[j] <- 1/pow(sd.delta[j],2)
     sd.delta[j] ~ dunif(0,100)
}

# Priors for area-specific prevalences. Average prevalence across region fixed at 23%.
for (i in 1:(n)) {
    theta[i] ~ dunif(0,100)
    theta.adj[i]<-theta[i]-thetabar23
}
thetabar<-mean(theta[])
thetabar23<-thetabar-23
}
```

More complex models (cont'd)



Suggested references and reading

- Brewer, B.J., Introduction to Bayesian Statistics. Course notes. University of Auckland.
- De Finetti, B (1931). Funzione caratteristica di un fenomeno aleatorio. Accademia dei Lincei, Roma.

In [ ]:

Using MCMC-Metropolis Hastings to approximate parameters for a posterior probability

The goal of this nb is to approximate a posterior probability function for a runner to be at max intensity heart rate, as a function of time.

Import packages

In [2]:
```
pip install pymc3
```

```
Collecting pymc3
  Downloading pymc3-3.11.5-py3-none-any.whl (872 kB)
     |████████████████████████████████| 872 kB 875 kB/s eta 0:00:0
1
Collecting theano-pymc==1.1.2
  Downloading Theano-PyMC-1.1.2.tar.gz (1.8 MB)
     |████████████████████████████████| 1.8 MB 154 kB/s eta 0:00:0
1
Collecting deprecat
  Downloading deprecat-2.1.1-py2.py3-none-any.whl (9.8 kB)
Requirement already satisfied: pandas>=0.24.0 in /opt/anaconda3/li
```

```
b/python3.8/site-packages (from pymc3) (1.3.4)
Collecting fastprogress>=0.2.0
  Downloading fastprogress-1.0.2-py3-none-any.whl (12 kB)
Collecting arviz>=0.11.0
  Downloading arviz-0.12.1-py3-none-any.whl (1.6 MB)
     |████████████████████████████████| 1.6 MB 136 kB/s eta 0:00:0
1
Collecting cachetools>=4.2.1
  Downloading cachetools-5.2.0-py3-none-any.whl (9.3 kB)
Requirement already satisfied: numpy<1.22.2,>=1.15.0 in /opt/anaco
nda3/lib/python3.8/site-packages (from pymc3) (1.20.2)
Collecting semver>=2.13.0
  Downloading semver-2.13.0-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: typing-extensions>=3.7.4 in /opt/an
aconda3/lib/python3.8/site-packages (from pymc3) (3.7.4.3)
Requirement already satisfied: patsy>=0.5.1 in /opt/anaconda3/lib/
python3.8/site-packages (from pymc3) (0.5.1)
Collecting scipy<1.8.0,>=1.7.3
  Downloading scipy-1.7.3-cp38-cp38-macosx_10_9_x86_64.whl (33.0 M
B)
     |████████████████████████████████| 33.0 MB 1.7 MB/s eta 0:00:
01
Collecting dill
  Downloading dill-0.3.5.1-py2.py3-none-any.whl (95 kB)
     |████████████████████████████████| 95 kB 1.4 MB/s eta 0:00:01
Requirement already satisfied: filelock in /opt/anaconda3/lib/pyth
on3.8/site-packages (from theano-pymc==1.1.2->pymc3) (3.0.12)
Collecting netcdf4
  Downloading netCDF4-1.5.8-cp38-cp38-macosx_10_9_x86_64.whl (4.2
MB)
     |████████████████████████████████| 4.2 MB 1.2 MB/s eta 0:00:0
1
Collecting xarray>=0.16.1
  Downloading xarray-2022.3.0-py3-none-any.whl (870 kB)
     |████████████████████████████████| 870 kB 721 kB/s eta 0:00:0
1
Requirement already satisfied: matplotlib>=3.0 in /opt/anaconda3/l
ib/python3.8/site-packages (from arviz>=0.11.0->pymc3) (3.3.4)
Requirement already satisfied: packaging in /opt/anaconda3/lib/pyt
hon3.8/site-packages (from arviz>=0.11.0->pymc3) (20.9)
Collecting xarray-einstats>=0.2
  Downloading xarray_einstats-0.2.2-py3-none-any.whl (33 kB)
Requirement already satisfied: setuptools>=38.4 in /opt/anaconda3/
lib/python3.8/site-packages (from arviz>=0.11.0->pymc3) (52.0.0.po
st20210125)
Requirement already satisfied: python-dateutil>=2.1 in /opt/anacon
da3/lib/python3.8/site-packages (from matplotlib>=3.0->arviz>=0.11
.0->pymc3) (2.8.1)
Requirement already satisfied: cycler>=0.10 in /opt/anaconda3/lib/
python3.8/site-packages (from matplotlib>=3.0->arviz>=0.11.0->pymc
3) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=
2.0.3 in /opt/anaconda3/lib/python3.8/site-packages (from matplotl
ib>=3.0->arviz>=0.11.0->pymc3) (2.4.7)
```

```
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/anaconda3
/lib/python3.8/site-packages (from matplotlib>=3.0->arviz>=0.11.0-
>pymc3) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in /opt/anaconda3/lib
/python3.8/site-packages (from matplotlib>=3.0->arviz>=0.11.0->pym
c3) (8.2.0)
Requirement already satisfied: six in /opt/anaconda3/lib/python3.8
/site-packages (from cycler>=0.10->matplotlib>=3.0->arviz>=0.11.0-
>pymc3) (1.15.0)
Requirement already satisfied: pytz>=2017.3 in /opt/anaconda3/lib/
python3.8/site-packages (from pandas>=0.24.0->pymc3) (2021.1)
Collecting numpy<1.22.2,>=1.15.0
  Downloading numpy-1.22.1-cp38-cp38-macosx_10_9_x86_64.whl (17.6
MB)
     |████████████████████████████████| 17.6 MB 1.1 MB/s eta 0:00:
01
Requirement already satisfied: wrapt<2,>=1.10 in /opt/anaconda3/li
b/python3.8/site-packages (from deprecat->pymc3) (1.12.1)
Collecting cftime
  Downloading cftime-1.6.0-cp38-cp38-macosx_10_9_x86_64.whl (226 k
B)
     |████████████████████████████████| 226 kB 1.1 MB/s eta 0:00:0
1
Building wheels for collected packages: theano-pymc
  Building wheel for theano-pymc (setup.py) ... done
  Created wheel for theano-pymc: filename=Theano_PyMC-1.1.2-py3-no
ne-any.whl size=1529946 sha256=382505738fce2281b0ec6e3db76ce5ae0b4
3b8c5965e3ff7e2c180e2317d83ae
  Stored in directory: /Users/giancarlomanzi/Library/Caches/pip/wh
eels/0e/41/d2/82c7b771236f987def7fe2e51855cce22b270327f3fedec57c
Successfully built theano-pymc
Installing collected packages: numpy, xarray, scipy, cftime, xarra
y-einstats, netcdf4, theano-pymc, semver, fastprogress, dill, depr
ecat, cachetools, arviz, pymc3
  Attempting uninstall: numpy
    Found existing installation: numpy 1.20.2
    Uninstalling numpy-1.20.2:
      Successfully uninstalled numpy-1.20.2
  Attempting uninstall: scipy
    Found existing installation: scipy 1.6.2
    Uninstalling scipy-1.6.2:
      Successfully uninstalled scipy-1.6.2
  Attempting uninstall: cachetools
    Found existing installation: cachetools 4.2.0
    Uninstalling cachetools-4.2.0:
      Successfully uninstalled cachetools-4.2.0
ERROR: pip's dependency resolver does not currently take into acco
unt all the packages that are installed. This behaviour is the sou
rce of the following dependency conflicts.
yellowbrick 1.3.post1 requires numpy<1.20,>=1.16.0, but you have n
umpy 1.22.1 which is incompatible.
tensorflow 2.4.0 requires grpcio~=1.32.0, but you have grpcio 1.41
.1 which is incompatible.
```

```
tensorflow 2.4.0 requires numpy~=1.19.2, but you have numpy 1.22.1
which is incompatible.
tensorboard 2.4.0 requires google-auth<2,>=1.6.3, but you have goo
gle-auth 2.3.2 which is incompatible.
google-auth 2.3.2 requires cachetools<5.0,>=2.0.0, but you have ca
chetools 5.2.0 which is incompatible.
Successfully installed arviz-0.12.1 cachetools-5.2.0 cftime-1.6.0
deprecat-2.1.1 dill-0.3.5.1 fastprogress-1.0.2 netcdf4-1.5.8 numpy
-1.22.1 pymc3-3.11.5 scipy-1.7.3 semver-2.13.0 theano-pymc-1.1.2 x
array-2022.3.0 xarray-einstats-0.2.2
WARNING: You are using pip version 21.2.2; however, version 22.1.2
is available.
You should consider upgrading via the '/opt/anaconda3/bin/python -
m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

In [24]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython.core.pylabtools import figsize

from datetime import datetime
```

Import heart-rate and workout-data

In [25]:

```
heart_rate = pd.read_csv("Heartrate.csv")

work_out = pd.read_csv("Workout.csv")
```

In [26]:
```
1 work_out
```

Out[26]:

	sourceName	sourceVersion	device	creationDate	startDate	endDate	
0	Strava	18214	NaN	2019-09-04 16:35:52 +0100	2019-09-04 14:52:44 +0100	2019-09-04 15:29:27 +0100	HKWorkou
1	Strava	18214	NaN	2019-09-05 20:34:06 +0100	2019-09-05 16:29:52 +0100	2019-09-05 17:02:09 +0100	HKWorkou
2	Filips Apple Watch	5.3	<<HKDevice: 0x282f7c640>, name:Apple Watch, ma...	2019-09-05 19:01:44 +0100	2019-09-05 18:36:01 +0100	2019-09-05 19:01:42 +0100	HKWorko
3	Filips Apple Watch	5.3	<<HKDevice: 0x282f7c780>, name:Apple Watch, ma...	2019-09-05 22:28:10 +0100	2019-09-05 22:12:09 +0100	2019-09-05 22:28:09 +0100	HKWorko
4	Filips Apple Watch	5.3	<<HKDevice: 0x282f7c8c0>, name:Apple Watch, ma...	2019-09-06 09:39:43 +0100	2019-09-06 09:01:23 +0100	2019-09-06 09:39:38 +0100	HKWorko
...	...	...	...	...	...	...	
82	Filips Apple Watch	5.3.1	<<HKDevice: 0x282f7dcc0>, name:Apple Watch, ma...	2019-11-19 18:34:28 +0100	2019-11-19 17:53:53 +0100	2019-11-19 18:34:25 +0100	HKWorko
83	Strava	18518	NaN	2019-11-21 13:04:26 +0100	2019-11-21 11:57:28 +0100	2019-11-21 12:14:29 +0100	HKWorkou
84	Filips Apple Watch	5.3.1	<<HKDevice: 0x282f7de00>, name:Apple Watch, ma...	2019-11-23 13:02:42 +0100	2019-11-23 11:42:15 +0100	2019-11-23 13:02:36 +0100	HKWorko
85	Strava	18518	NaN	2019-11-24 12:07:10 +0100	2019-11-24 09:36:21 +0100	2019-11-24 11:11:40 +0100	HKWorkou
86	Strava	18518	NaN	2019-11-25 15:33:38 +0100	2019-11-25 14:50:03 +0100	2019-11-25 15:19:33 +0100	HKWorkou

87 rows × 13 columns

Manipulate work-out data and exctract time-ranges for runs

In [27]:
```
# remove all non-run workouts
work_out = work_out.loc[work_out.sourceName == "Strava"]
# convert to datetime-object
work_out['sd'] = pd.to_datetime(work_out['startDate'])
work_out['ed'] = pd.to_datetime(work_out['endDate'])
# remove timezone-info
work_out.sd = work_out.sd.apply(lambda x: x.replace(tzinfo=None
work_out.ed = work_out.ed.apply(lambda x: x.replace(tzinfo=None
# extract data as lists for easier use later
wo_sd = work_out.sd.tolist() # work out start-date
wo_ed = work_out.ed.tolist() # work out end-date
```

```
<ipython-input-27-bc1f66e1f0be>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-
a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/in
dexing.html#returning-a-view-versus-a-copy)
  work_out['sd'] = pd.to_datetime(work_out['startDate'])
<ipython-input-27-bc1f66e1f0be>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-
a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/in
dexing.html#returning-a-view-versus-a-copy)
  work_out['ed'] = pd.to_datetime(work_out['endDate'])
/opt/anaconda3/lib/python3.8/site-packages/pandas/core/generic.py:
5516: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-
a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/in
dexing.html#returning-a-view-versus-a-copy)
  self[name] = value
```

In [28]:
```
1 work_out
```

Out[28]:

	sourceName	sourceVersion	device	creationDate	startDate	endDate	workou
0	Strava	18214	NaN	2019-09-04 16:35:52 +0100	2019-09-04 14:52:44 +0100	2019-09-04 15:29:27 +0100	HKWorkoutActivity
1	Strava	18214	NaN	2019-09-05 20:34:06 +0100	2019-09-05 16:29:52 +0100	2019-09-05 17:02:09 +0100	HKWorkoutActivity
9	Strava	18214	NaN	2019-09-08 16:11:10 +0100	2019-09-08 15:00:43 +0100	2019-09-08 15:47:33 +0100	HKWorkoutActivity
12	Strava	18214	NaN	2019-09-09 15:04:01 +0100	2019-09-09 13:59:03 +0100	2019-09-09 14:26:04 +0100	HKWorkoutActivity
13	Strava	18214	NaN	2019-09-09 15:05:32 +0100	2019-09-09 14:55:07 +0100	2019-09-09 15:05:19 +0100	HKWorkoutActivity
...	...	...	...	...	...	...	
80	Strava	18518	NaN	2019-11-19 15:56:18 +0100	2019-11-19 13:42:57 +0100	2019-11-19 14:22:08 +0100	HKWorkoutActivity
81	Strava	18518	NaN	2019-11-19 15:56:18 +0100	2019-11-19 14:24:22 +0100	2019-11-19 14:36:53 +0100	HKWorkoutActivity
83	Strava	18518	NaN	2019-11-21 13:04:26 +0100	2019-11-21 11:57:28 +0100	2019-11-21 12:14:29 +0100	HKWorkoutActivity
85	Strava	18518	NaN	2019-11-24 12:07:10 +0100	2019-11-24 09:36:21 +0100	2019-11-24 11:11:40 +0100	HKWorkoutActivity
86	Strava	18518	NaN	2019-11-25 15:33:38 +0100	2019-11-25 14:50:03 +0100	2019-11-25 15:19:33 +0100	HKWorkoutActivity

67 rows × 15 columns

Make heart-rate observation time column the index for easier manipulation

In [29]:
```
heart_rate['sd'] = pd.to_datetime(heart_rate['startDate'])
heart_rate.sd = heart_rate.sd.apply(lambda x: x.replace(tzinfo=
heart_rate.set_index('sd',inplace=True)
```

In [30]:
```
# rename value-column to heart_rate
heart_rate.rename(columns={'value': 'heart_rate'}, inplace=True
```

Create new column run_id to categorize heart-rate data into different runs

In [31]:
```
heart_rate['run_id'] = np.nan
new_dataframe = pd.DataFrame()

for i in range(len(wo_sd)):
    temp_df = heart_rate.loc[wo_sd[i] : wo_ed[i]]
    temp_df.run_id=i
    new_dataframe = new_dataframe.append(temp_df)


# make original df the new one and delete new one to save space
heart_rate = new_dataframe.copy()
del(new_dataframe)
```

```
<ipython-input-31-486071692fb4>:5: FutureWarning: Value based part
ial slicing on non-monotonic DatetimeIndexes with non-existing key
s is deprecated and will raise a KeyError in a future Version.
  temp_df = heart_rate.loc[wo_sd[i] : wo_ed[i]]
```

In [32]:
```
heart_rate
```

Out[32]:

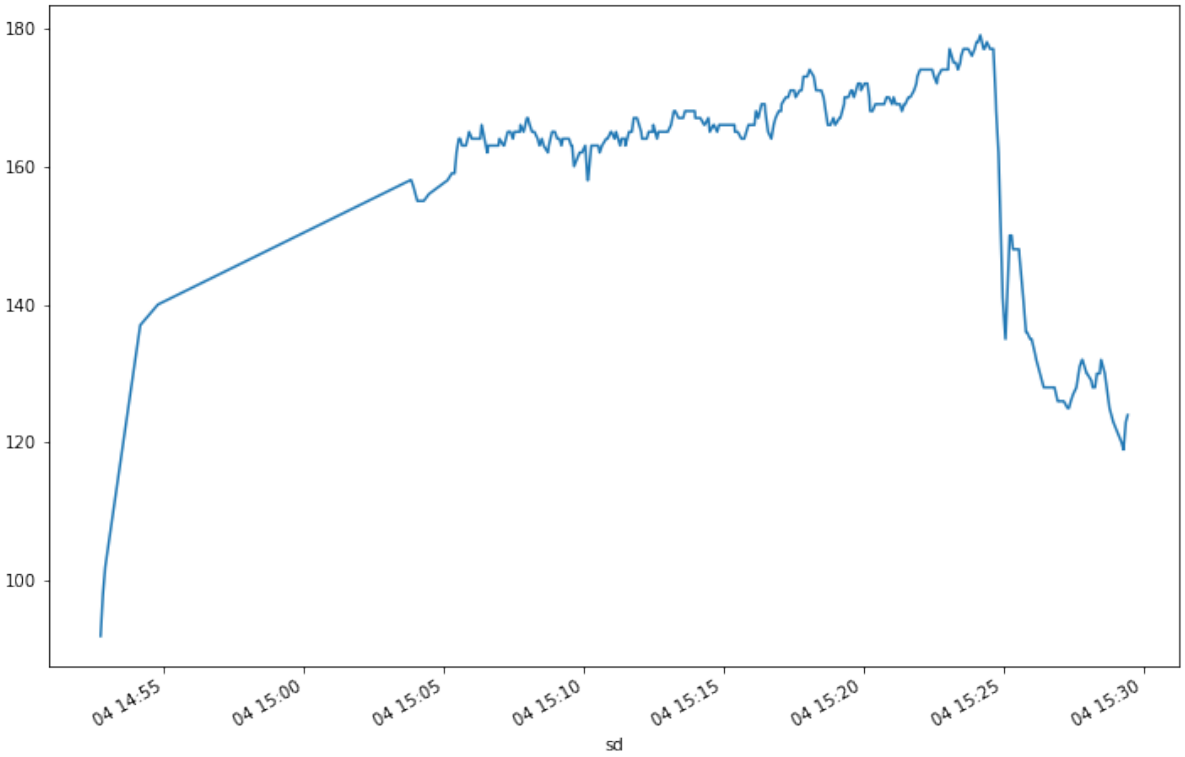
	sourceName	sourceVersion	device	type	unit	creationDate	sta
sd							
2019-09-04 14:52:44	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2ad0>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:52:49 +0100	20 1
2019-09-04 14:52:49	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2e40>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:52:54 +0100	20 1
2019-09-04 14:52:54	Filips Apple Watch	5.3	<<HKDevice: 0x282cf0eb0>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:52:59 +0100	20 1
2019-09-04 14:54:09	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2620>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:54:13 +0100	20 1
2019-09-04 14:54:47	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2530>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:54:49 +0100	20 1
...	...	...	...	...	...	...	
2019-11-25 15:19:12	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafb10>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:13 +0100	20 1
2019-11-25 15:19:15	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafbb0>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:18 +0100	20 1
2019-11-25 15:19:21	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafc50>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:23 +0100	20 1
2019-11-25 15:19:23	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafcf0>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:28 +0100	20 1
2019-11-25 15:19:32	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafd90>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:33 +0100	20 1

12229 rows × 10 columns

Plot to se heart-rate data from one run

In [33]:
```
heart_rate[heart_rate.run_id==0].heart_rate.plot()
```

Out[33]: <AxesSubplot:xlabel='sd'>



Manipulate data further to create timedelta (from start of run to finish)

In [33]:
```
heart_rate['time_delta'] = np.nan
heart_rate['sd'] = heart_rate.index

new_df = pd.DataFrame()
for rid in heart_rate.run_id.unique():
    temp_df = heart_rate.loc[heart_rate.run_id==rid]
    start = temp_df.iloc[0].sd
    temp_df['time_delta'] = temp_df.sd.apply(lambda x: x-start)
    new_df = new_df.append(temp_df)

heart_rate = new_df.copy()
del(new_df)
heart_rate
```

```
<ipython-input-33-1ad3007572e1>:8: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  temp_df['time_delta'] = temp_df.sd.apply(lambda x: x-start)
```

Out[33]:

sd	sourceName	sourceVersion	device	type	unit	creationDate	sta
2019-09-04 14:52:44	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2ad0>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:52:49 +0100	20 1
2019-09-04 14:52:49	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2e40>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:52:54 +0100	20 1
2019-09-04 14:52:54	Filips Apple Watch	5.3	<<HKDevice: 0x282cf0eb0>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:52:59 +0100	20 1
2019-09-04 14:54:09	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2620>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:54:13 +0100	20 1
2019-09-04 14:54:47	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2530>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:54:49 +0100	20 1
...	...	...	...	...	...	...	
2019-11-25 15:19:12	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafb10>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:13 +0100	20 1
2019-11-25 15:19:15	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafbb0>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:18 +0100	20 1
2019-11-25 15:19:21	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafc50>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:23 +0100	20 1
2019-11-25 15:19:23	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafcf0>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:28 +0100	20 1
2019-11-25	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafd90>, name:Apple	HeartRate	count/min	2019-11-25 15:19:33	20 1

15:19:32 Watch, ma... +0100

12229 rows × 12 columns

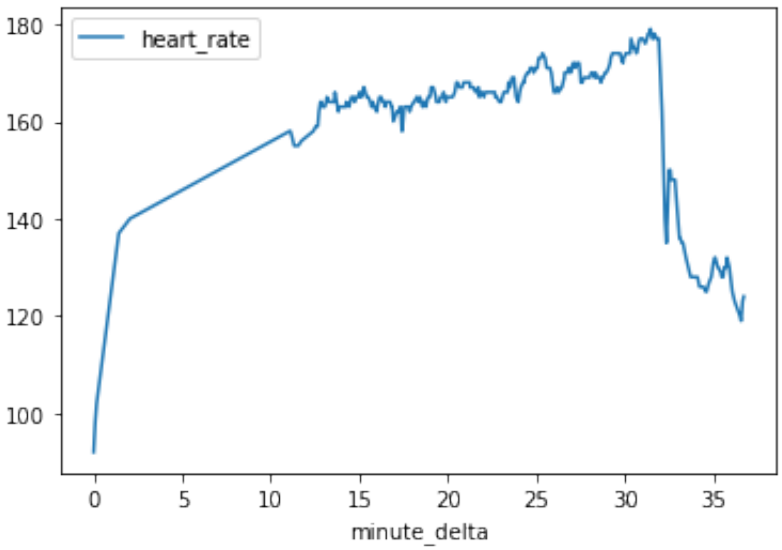
Create additional time-delta column for minutes (instead of seconds)

In [34]:
```
heart_rate['minute_delta'] = heart_rate.time_delta.apply(lambda
```

Plot to see heart-rate data from one run with timedelta as x-axis

In [35]:
```
heart_rate[heart_rate.run_id==0].plot(x='minute_delta', y='hear
```

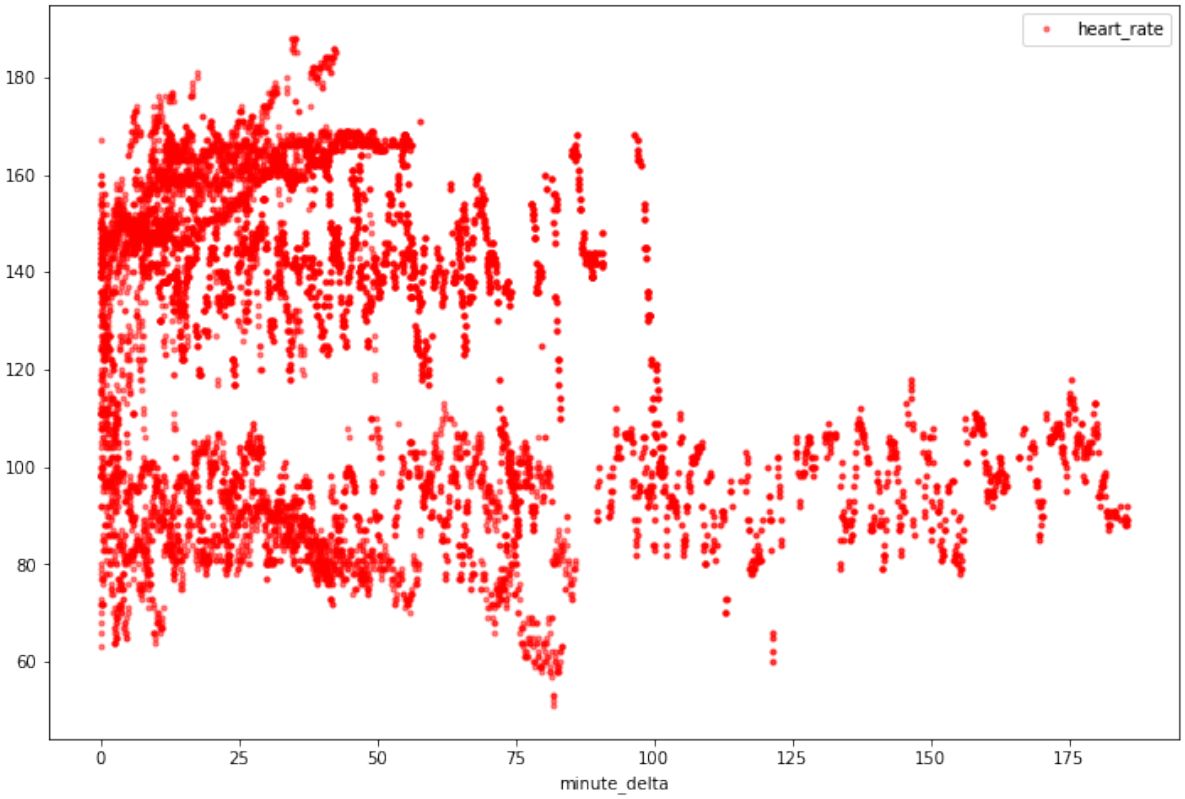
Out[35]: <AxesSubplot:xlabel='minute_delta'>



Scatter plot for all runs

In [36]:
```
figsize(12,8)
heart_rate.plot(x='minute_delta', y ='heart_rate', linestyle='N
#plt.savefig('scattered_hr.png')
```

Out[36]: <AxesSubplot:xlabel='minute_delta'>



- it becomes apparent that a divide into short, medium and long runs is needed in order to find proper functions

Determine ranges for heart-rate max:

In [37]:
```
heart_rate.heart_rate.describe()
```

Out[37]:
```
count    12229.000000
mean       124.527925
std         30.878813
min         51.000000
25%         95.000000
50%        134.000000
75%        150.000000
max        188.000000
Name: heart_rate, dtype: float64
```

- Above 85% of max-heart rate is usually seen as max-intensity
- $0.85*188 \approx 160$

In [38]:
```
def max_inten(x):
    if x>160:
        return 1
    else:
        return 0
```

In [40]:
```
heart_rate['max_intensity'] = heart_rate.heart_rate.apply(max_i
heart_rate
```

Out[40]:

	sourceName	sourceVersion	device	type	unit	creationDate	sta
sd							
2019-09-04 14:52:44	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2ad0>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:52:49 +0100	2(1
2019-09-04 14:52:49	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2e40>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:52:54 +0100	2(1
2019-09-04 14:52:54	Filips Apple Watch	5.3	<<HKDevice: 0x282cf0eb0>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:52:59 +0100	2(1
2019-09-04 14:54:09	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2620>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:54:13 +0100	2(1
2019-09-04 14:54:47	Filips Apple Watch	5.3	<<HKDevice: 0x282cf2530>, name:Apple Watch, ma...	HeartRate	count/min	2019-09-04 14:54:49 +0100	2(1
...	...	...	...	...	...	...	
2019-11-25 15:19:12	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafb10>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:13 +0100	2(1
2019-11-25 15:19:15	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafbb0>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:18 +0100	2(1
2019-11-25 15:19:21	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafc50>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:23 +0100	2(1
2019-11-25 15:19:23	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafcf0>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:28 +0100	2(1
2019-11-25 15:19:32	Filips Apple Watch	5.3.1	<<HKDevice: 0x282cafd90>, name:Apple Watch, ma...	HeartRate	count/min	2019-11-25 15:19:33 +0100	2(1

12229 rows × 14 columns

Divide into short/medium/long runs

In [41]:
```
short_runs = pd.DataFrame()
medium_runs = pd.DataFrame()
long_runs = pd.DataFrame()

for rid in heart_rate.run_id.unique().tolist():
    temp = heart_rate.loc[heart_rate.run_id==rid]
    secs = temp.iloc[-1].time_delta.total_seconds()
    if secs<3000:
        short_runs = short_runs.append(temp)
    elif secs >= 3000 and secs < 6000:
        medium_runs = medium_runs.append(temp)
    else:
        long_runs = long_runs.append(temp)


```

In [42]:
```
### sort short-runsdd based on time from start
short_runs.sort_values('minute_delta', inplace=True)
```

Plot short runs

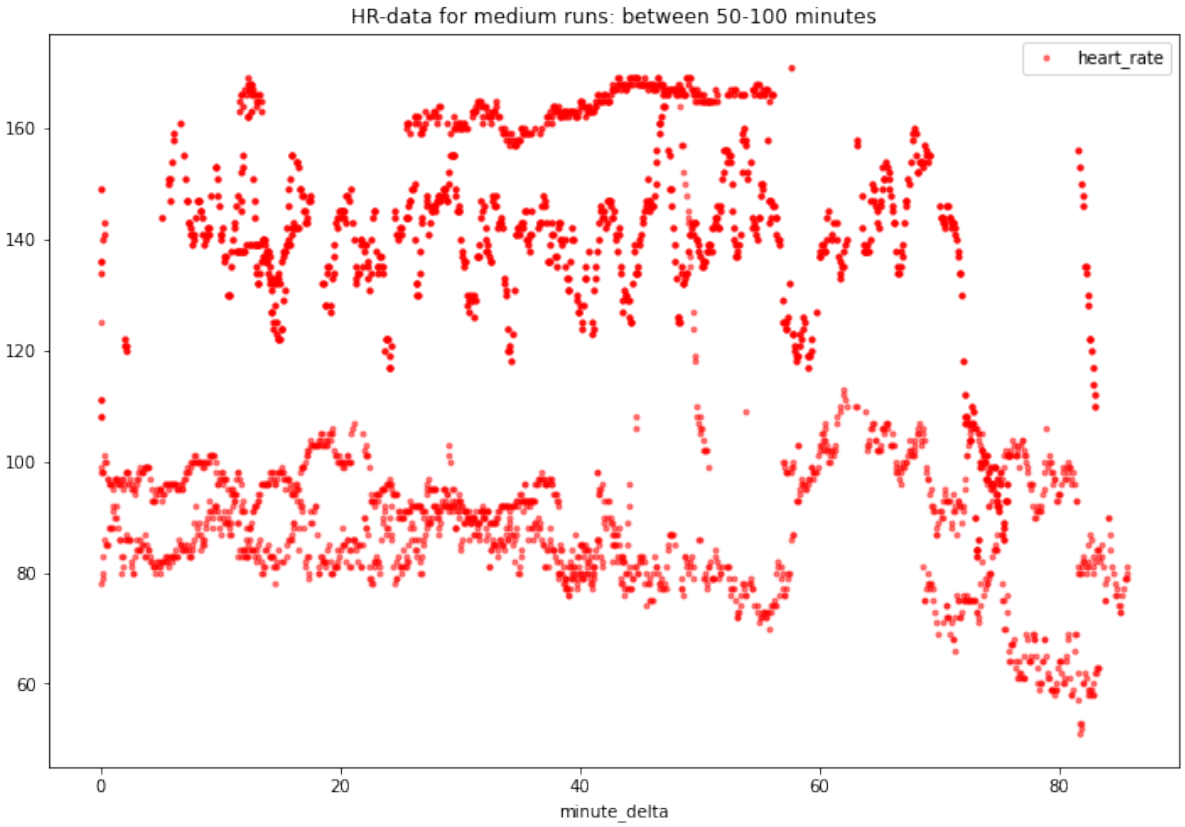
In [43]:
```
short_runs.plot(x='minute_delta', y ='heart_rate', linestyle='N
#plt.savefig('short_run_scatter.png')
```

Out[43]:
```
<AxesSubplot:title={'center':'Hr-data for short runs: <50 minutes'
}, xlabel='minute_delta'>
```

Medium runs

In [44]:
```
medium_runs.plot(x='minute_delta', y ='heart_rate', linestyle='
#plt.savefig('medium_scatter.png')
```

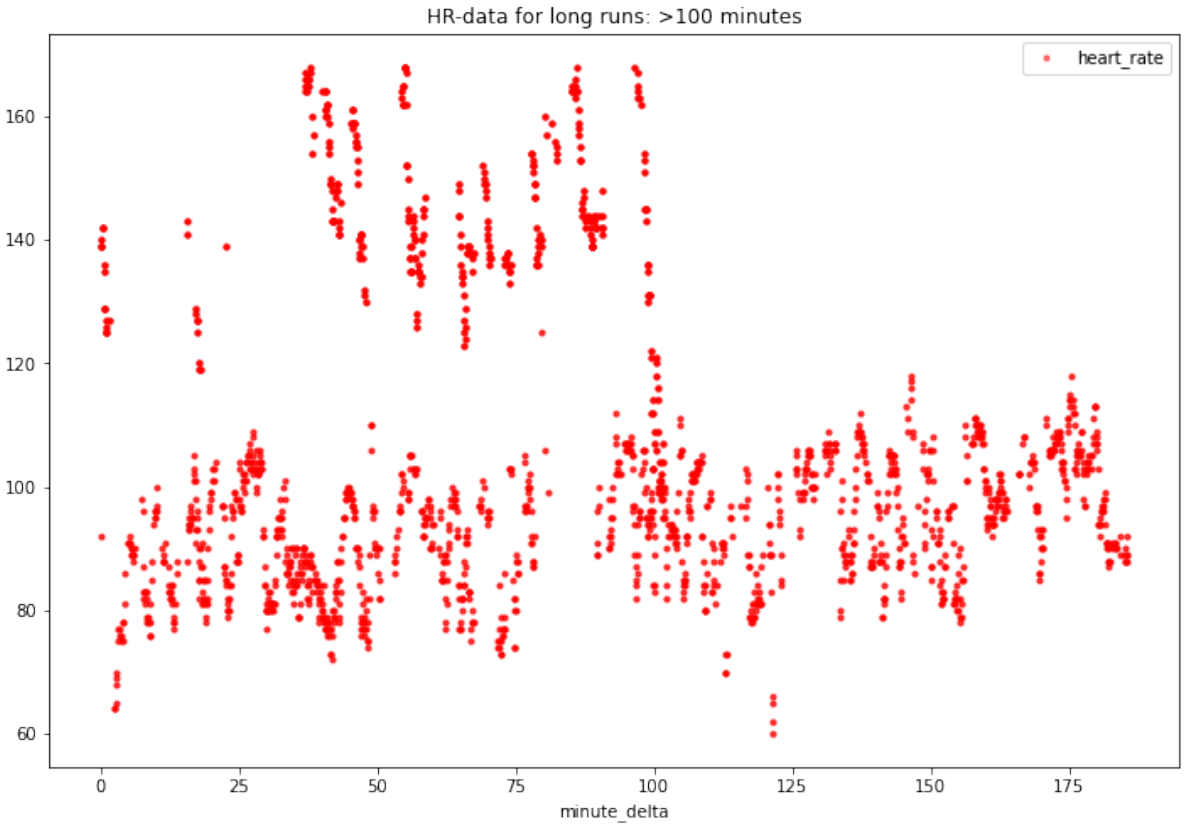
Out[44]:
```
<AxesSubplot:title={'center':'HR-data for medium runs: between 50-
100 minutes'}, xlabel='minute_delta'>
```



Long runs

In [45]:
```
long_runs.plot(x='minute_delta', y ='heart_rate', linestyle='No
#plt.savefig('long_scatter.png')
```

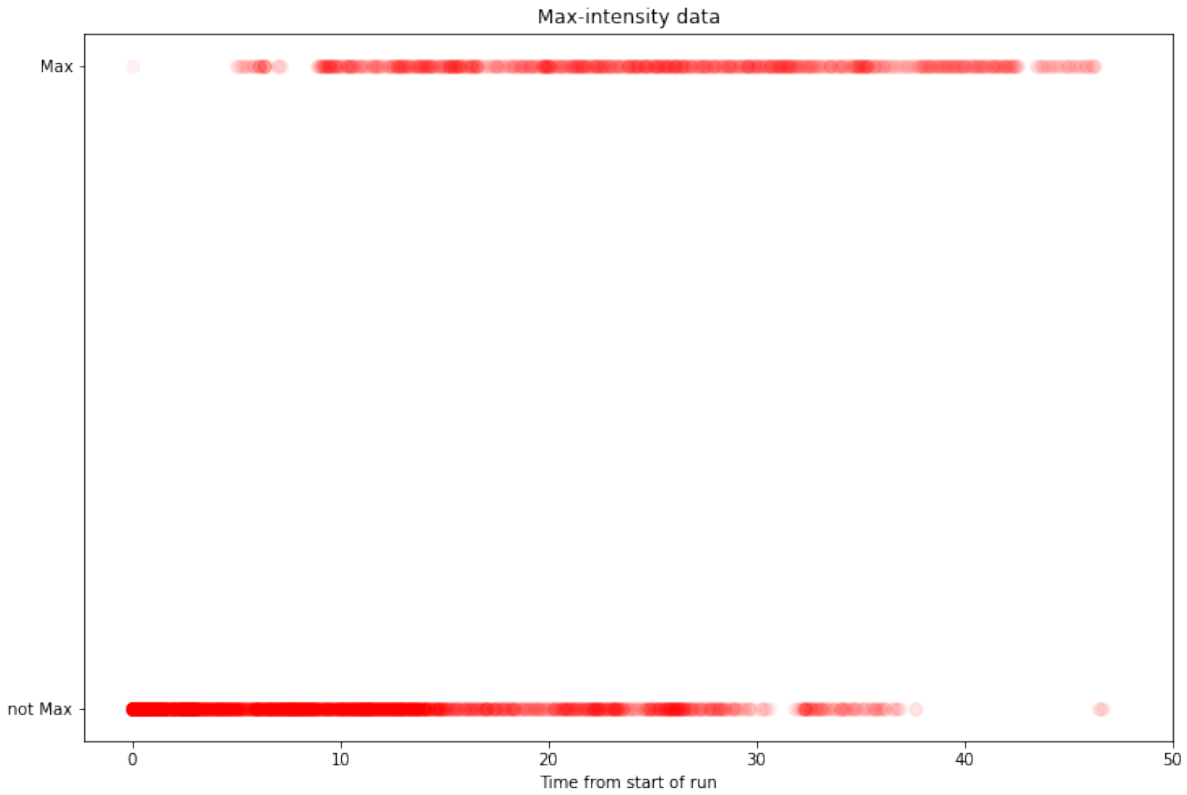
Out[45]:
```
<AxesSubplot:title={'center':'HR-data for long runs: >100 minutes'
}, xlabel='minute_delta'>
```



Plot max-intensity data

In [46]:
```
plt.scatter(short_runs['minute_delta'], short_runs['max_intensi
plt.yticks([0,1], ['not Max', 'Max'])
plt.xlabel('Time from start of run')
plt.xticks([0,10,20,30,40,50])
plt.title('Max-intensity data')
#plt.show()
#plt.savefig('max_intensity_short.png')
```

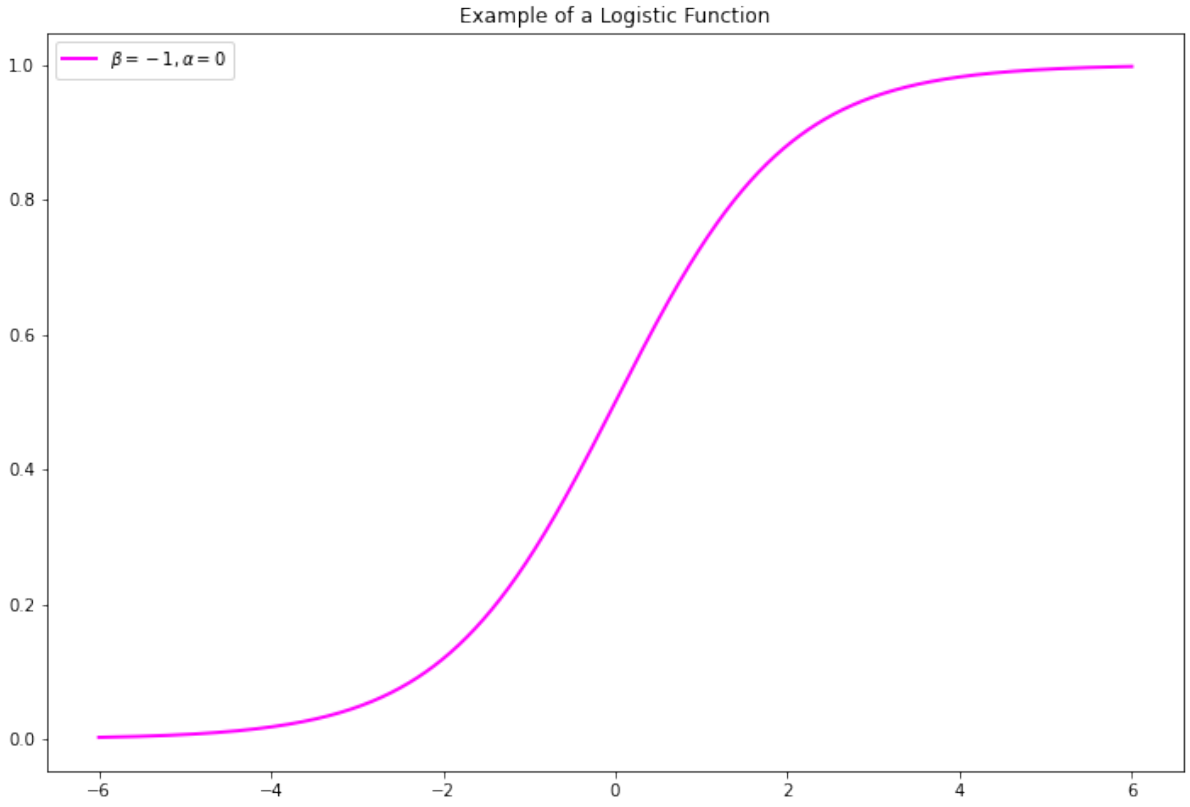
Out[46]: Text(0.5, 1.0, 'Max-intensity data')



Determine function to model the behaviour of the data

- logistic function seems viable

In [47]:
```
# Logistic function with arguments x, alpha and beta
# np.dot is a dot product of two arrays
def logistic(x, beta, alpha=0):
    return 1.0 / (1.0 + np.exp(np.dot(beta, x) + alpha))

x = np.linspace(-6, 6, 1000)
plt.plot(x, logistic(x, beta=-1), label=r"$\beta = -1, \alpha =
plt.legend()
plt.title(r'Example of a Logistic Function');
#plt.savefig('logistic.png')
```



Build MCMC-model

In [48]:
```
import pymc3 as pm
import theano.tensor as tt
```

In [51]:
```
#x-value
time = np.array(short_runs.loc[:,'minute_delta'])

#y-value
hr_obs = np.array(short_runs.loc[:, 'max_intensity'])
unique, counts = np.unique(hr_obs, return_counts=True)
print(np.asarray((unique, counts)).T)
```

```
[[   0 2209]
 [   1  959]]
```

In [67]:
```
N_SAMPLES = 100

with pm.Model() as model:
    # draw parameters for function from normal dist
    alpha = pm.Normal('alpha', mu=0.0, tau=0.01, testval=0.0)
    beta = pm.Normal('beta', mu=0.0, tau=0.01, testval=0.0)

    # create probability from the logistic function:
    p = pm.Deterministic('p', 1. / (1. + tt.exp(beta * time + a

    # model the observed max-heart rates as a Bernoulli RV:
    observed = pm.Bernoulli('obs', p, observed=hr_obs)

    # choose MCMC-algo:
    step = pm.Metropolis()

    # model chain(s)
    hr_trace = pm.sample(N_SAMPLES, step=step, tune = 50, chain

```

```
Only 100 samples in chain.
Multiprocess sampling (2 chains in 4 jobs)
CompoundStep
>Metropolis: [beta]
>Metropolis: [alpha]
```

100.00% [300/300 04:29<00:00

Sampling 2 chains, 0 divergences]

```
Sampling 2 chains for 50 tune and 100 draw iterations (100 + 200 d
raws total) took 275 seconds.
The rhat statistic is larger than 1.4 for some parameters. The sam
pler did not converge.
The number of effective samples is smaller than 10% for some param
eters.
```

In [70]:
```
len(hr_trace["alpha"])
```

Out[70]: 200

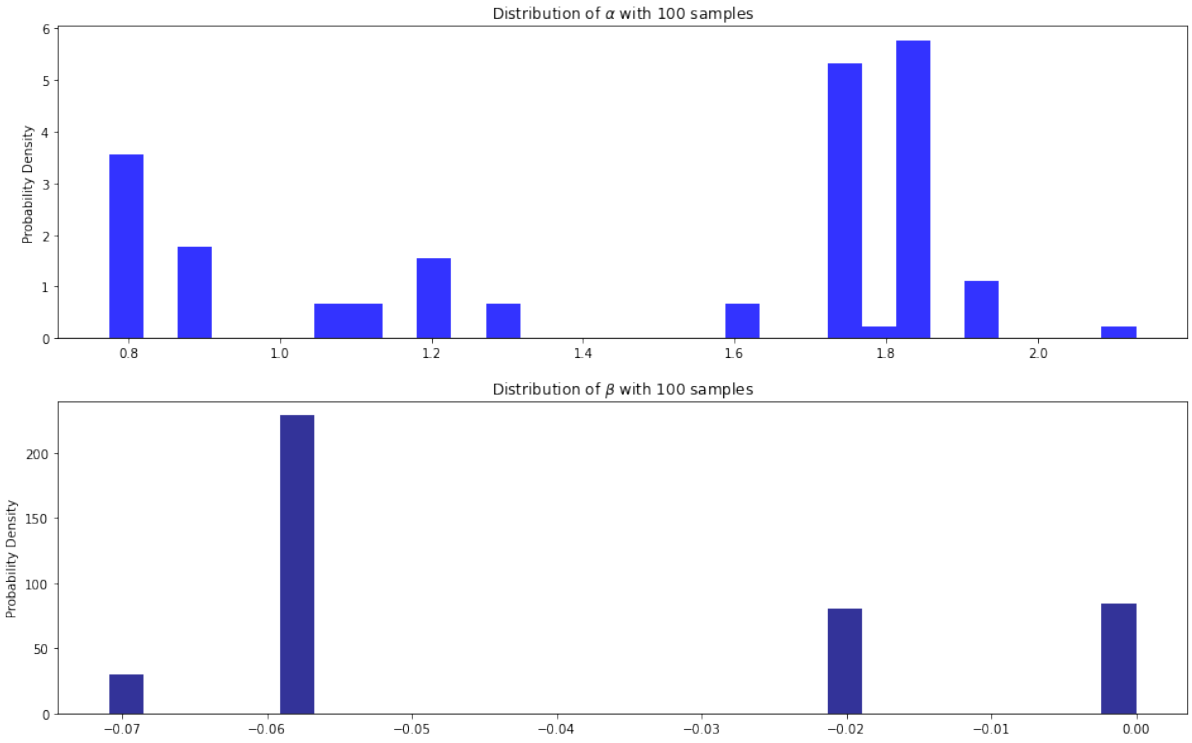
In [74]:
```
hr_trace["alpha"]
```

Out[74]:
```
array([1.7805651 , 1.7805651 , 1.7805651 , 1.7805651 , 1.7805651 ,
       1.7805651 , 1.7805651 , 1.8152321 , 1.8152321 , 1.8152321 ,
       1.8152321 , 1.8152321 , 1.8152321 , 1.8152321 , 1.8152321 ,
       1.8152321 , 1.8152321 , 1.8152321 , 1.8152321 , 1.8152321 ,
       1.8152321 , 1.8152321 , 1.8152321 , 1.8152321 , 1.8152321 ,
       1.8152321 , 1.8152321 , 1.90705331, 1.90705331, 1.90705331,
       1.90705331, 1.90705331, 1.90705331, 1.90705331, 1.90705331,
       1.90705331, 1.90705331, 1.90705331, 1.90705331, 1.90705331,
       2.21029777, 2.21029777, 2.21029777, 2.21029777, 2.21029777,
       2.15610786, 2.15610786, 2.15610786, 2.15610786, 2.15610786,
       1.99344801, 1.99344801, 1.99344801, 1.99344801, 1.99344801,
       1.99344801, 1.99344801, 1.99344801, 1.99344801, 1.99344801,
       1.99344801, 1.99344801, 2.03799539, 2.03799539, 2.0649635 ,
       2.0649635 , 2.0649635 , 2.0649635 , 2.0649635 , 2.0649635 ,
       2.0649635 , 2.0649635 , 2.0649635 , 2.0649635 , 2.0649635 ,
       2.0649635 , 2.0649635 , 2.0649635 , 2.0649635 , 2.0649635 ,
       2.0649635 , 2.0649635 , 2.0649635 , 2.0649635 , 2.0649635 ,
       2.04875249, 2.04875249, 2.04875249, 2.04875249, 2.04875249,
       2.04875249, 2.04875249, 2.04875249, 2.04875249, 2.04875249,
       2.04875249, 2.04875249, 2.04875249, 2.04875249, 2.04875249,
       0.89893347, 0.89893347, 0.89893347, 0.89893347, 0.89893347,
       0.89893347, 0.89893347, 0.89893347, 0.77497768, 0.77497768,
       0.77497768, 0.77497768, 0.77497768, 0.77497768, 0.77497768,
       0.77497768, 0.77497768, 0.77497768, 0.77497768, 0.77497768,
       0.77497768, 0.77497768, 0.77497768, 0.77497768, 1.27617798,
       1.27617798, 1.27617798, 1.0752777 , 1.0752777 , 1.0752777 ,
       1.09598549, 1.09598549, 1.09598549, 1.19456542, 1.19456542,
       1.19456542, 1.19456542, 1.19456542, 1.19456542, 1.19456542,
       1.62142041, 1.62142041, 1.62142041, 1.83522078, 1.83522078,
       1.83522078, 1.83522078, 1.83522078, 1.83522078, 1.83522078,
       1.83522078, 1.84893376, 1.84893376, 1.84893376, 1.84893376,
       1.84893376, 1.84893376, 1.84893376, 1.84893376, 1.84893376,
       1.84893376, 1.84893376, 1.84893376, 1.84893376, 1.84893376,
       1.84893376, 1.84893376, 1.84893376, 1.84893376, 1.74843594,
       1.73684399, 1.73684399, 1.73684399, 1.73684399, 1.73684399,
       1.73684399, 1.73684399, 1.73684399, 1.73684399, 1.73684399,
       1.73684399, 1.73684399, 1.73684399, 1.73684399, 1.73684399,
       1.73684399, 1.73684399, 1.73684399, 1.73684399, 1.73684399,
       1.73684399, 1.73684399, 1.73684399, 1.78743775, 2.12949971,
       1.91955381, 1.91955381, 1.91955381, 1.91955381, 1.91955381]
)
```

In [71]:
```
# Extract the last half of the alpha and beta samples
alpha_samples = hr_trace["alpha"][100:, None]
beta_samples = hr_trace["beta"][100:, None]
```

In [72]:

```
from IPython.core.pylabtools import figsize
figsize(16, 10)
plt.subplot(211)
plt.title(r"""Distribution of $\alpha$ with %d samples""" % N_S

plt.hist(alpha_samples, histtype='stepfilled',
         color = 'blue', bins=30, alpha=0.8, density=True);
plt.ylabel('Probability Density')


plt.subplot(212)
plt.title(r"""Distribution of $\beta$ with %d samples""" % N_SA
plt.hist(beta_samples, histtype='stepfilled',
         color = 'navy', bins=30, alpha=0.8, density=True)
plt.ylabel('Probability Density');
#plt.savefig('prob_density.png')
```



In [75]:

```
# Time values for probability prediction
time_est = np.linspace(time.min()- 5, time.max() + 5, 1000)[:,

# Take most likely parameters to be mean values
alpha_est = alpha_samples.mean()
beta_est = beta_samples.mean()

# Probability at each time using mean values of alpha and beta
hr_est = logistic(time_est, beta=beta_est, alpha=alpha_est)
```

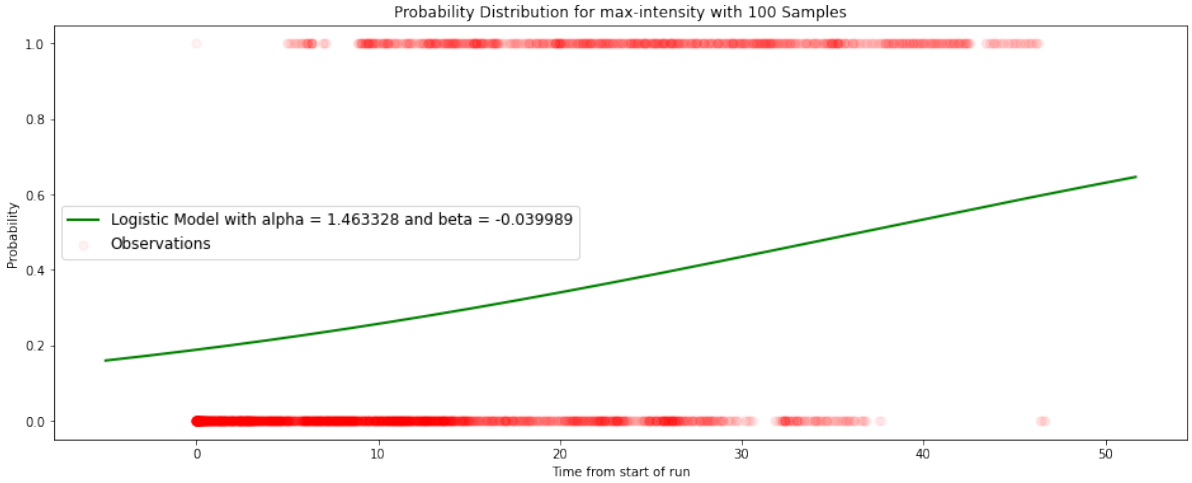
In [76]:
```
alpha_est
```

Out[76]: 1.4633278157251406

In [77]:
```
beta_est
```

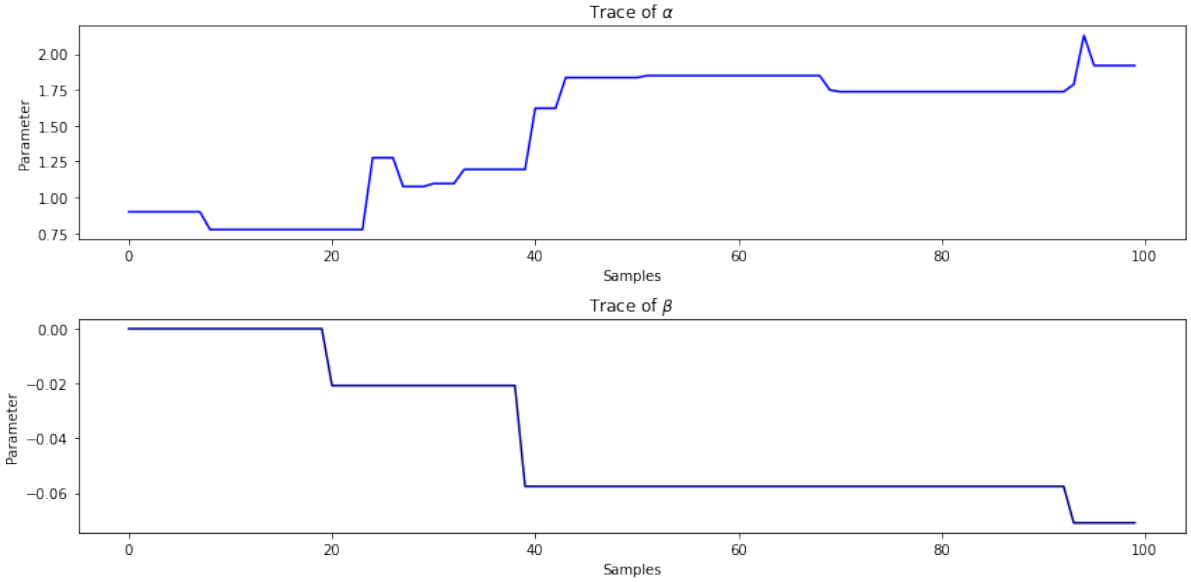
Out[77]: -0.03998920143513101

In [78]:
```
figsize(16, 6)

plt.plot(time_est, hr_est, color = 'green',
         lw=2, label="Logistic Model with alpha = %f and beta =
plt.scatter(time, hr_obs, edgecolor = 'red',
            s=50, alpha=0.05, label='Observations', facecolor='
plt.title('Probability Distribution for max-intensity with %d S
plt.legend(prop={'size':12})
plt.ylabel('Probability')
plt.xlabel('Time from start of run');
plt.xticks([0, 10, 20, 30, 40, 50]);
#plt.savefig('final_model.png')
```



Model convergence: trace

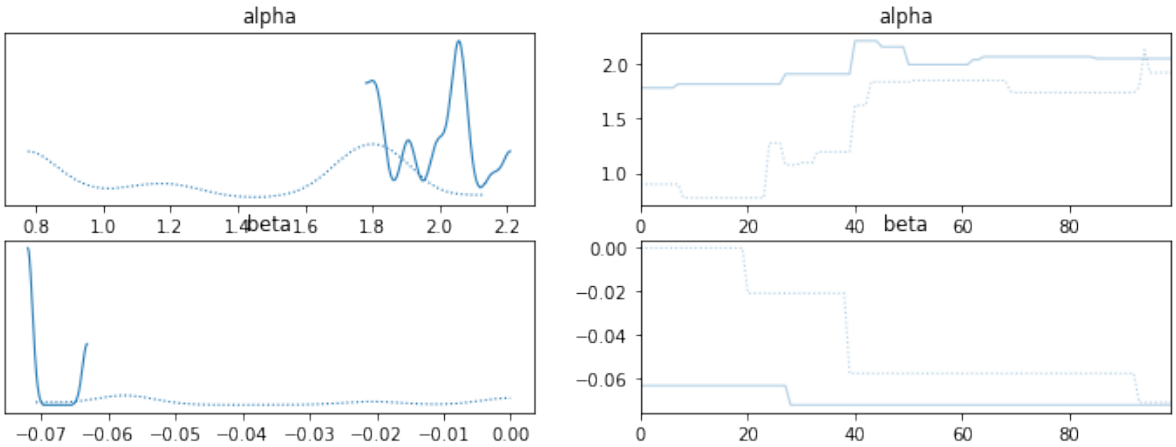
In [79]:
```
figsize(12, 6)

# Plot alpha trace
plt.subplot(211)
plt.title(r'Trace of $\alpha$')
plt.plot(alpha_samples, color = 'blue')
plt.xlabel('Samples'); plt.ylabel('Parameter');

# Plot beta trace
plt.subplot(212)
plt.title(r'Trace of $\beta$')
plt.plot(beta_samples, color='navy')
plt.xlabel('Samples'); plt.ylabel('Parameter');
plt.tight_layout(h_pad=0.8)
#plt.savefig('trace.png')

```



Model convergence: checking the shape and the mixing of the densities and the chains generated

In [80]:
```
figsize(20, 12)
pm.traceplot(hr_trace, ['alpha', 'beta']);
#plt.savefig('trace2.png')
```

```
<ipython-input-80-944eaaec0d5b>:2: DeprecationWarning: The functio
n `traceplot` from PyMC3 is just an alias for `plot_trace` from Ar
viZ. Please switch to `pymc3.plot_trace` or `arviz.plot_trace`.
  pm.traceplot(hr_trace, ['alpha', 'beta']);
Got error No model on context stack. trying to find log_likelihood
in translation.
/opt/anaconda3/lib/python3.8/site-packages/arviz/data/io_pymc3_3x.
py:98: FutureWarning: Using `from_pymc3` without the model will be
deprecated in a future release. Not using the model will return le
ss accurate and less useful results. Make sure you use the model a
rgument or call from_pymc3 within a model context.
  warnings.warn(
Got error No model on context stack. trying to find log_likelihood
in translation.
```



In [60]:
```
1 pip install arviz
```

```
Requirement already satisfied: arviz in /opt/anaconda3/lib/python3
.8/site-packages (0.12.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/
anaconda3/lib/python3.8/site-packages (from arviz) (3.7.4.3)
Requirement already satisfied: numpy>=1.12 in /opt/anaconda3/lib/p
ython3.8/site-packages (from arviz) (1.22.1)
Requirement already satisfied: setuptools>=38.4 in /opt/anaconda3/
lib/python3.8/site-packages (from arviz) (52.0.0.post20210125)
Requirement already satisfied: matplotlib>=3.0 in /opt/anaconda3/l
ib/python3.8/site-packages (from arviz) (3.3.4)
Requirement already satisfied: pandas>=0.23 in /opt/anaconda3/lib/
python3.8/site-packages (from arviz) (1.3.4)
Requirement already satisfied: xarray>=0.16.1 in /opt/anaconda3/li
b/python3.8/site-packages (from arviz) (2022.3.0)
Requirement already satisfied: netcdf4 in /opt/anaconda3/lib/pytho
n3.8/site-packages (from arviz) (1.5.8)
Requirement already satisfied: xarray-einstats>=0.2 in /opt/anacon
da3/lib/python3.8/site-packages (from arviz) (0.2.2)
Requirement already satisfied: scipy>=0.19 in /opt/anaconda3/lib/p
ython3.8/site-packages (from arviz) (1.7.3)
Requirement already satisfied: packaging in /opt/anaconda3/lib/pyt
hon3.8/site-packages (from arviz) (20.9)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=
2.0.3 in /opt/anaconda3/lib/python3.8/site-packages (from matplotl
ib>=3.0->arviz) (2.4.7)
Requirement already satisfied: cycler>=0.10 in /opt/anaconda3/lib/
python3.8/site-packages (from matplotlib>=3.0->arviz) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /opt/anacon
da3/lib/python3.8/site-packages (from matplotlib>=3.0->arviz) (2.8
.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/anaconda3
/lib/python3.8/site-packages (from matplotlib>=3.0->arviz) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in /opt/anaconda3/lib
/python3.8/site-packages (from matplotlib>=3.0->arviz) (8.2.0)
Requirement already satisfied: six in /opt/anaconda3/lib/python3.8
/site-packages (from cycler>=0.10->matplotlib>=3.0->arviz) (1.15.0
)
Requirement already satisfied: pytz>=2017.3 in /opt/anaconda3/lib/
python3.8/site-packages (from pandas>=0.23->arviz) (2021.1)
Requirement already satisfied: cftime in /opt/anaconda3/lib/python
3.8/site-packages (from netcdf4->arviz) (1.6.0)
WARNING: You are using pip version 21.2.2; however, version 22.1.2
is available.
You should consider upgrading via the '/opt/anaconda3/bin/python -
m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

In [11]:
```
pip install gif
```

```
Collecting gif
  Downloading gif-3.0.0.tar.gz (5.0 kB)
Requirement already satisfied: Pillow>=7.1.2 in /opt/anaconda3/lib
/python3.8/site-packages (from gif) (8.2.0)
Building wheels for collected packages: gif
  Building wheel for gif (setup.py) ... done
  Created wheel for gif: filename=gif-3.0.0-py3-none-any.whl size=
4817 sha256=2087513a34819559c0fb0c941b8349a3ba4ba5ece31cbf26ece92a
1fcc015008
  Stored in directory: /Users/giancarlomanzi/Library/Caches/pip/wh
eels/d8/db/4e/e0ce5209665322902834f4e88fa987f4dfa0c08271368ef098
Successfully built gif
Installing collected packages: gif
Successfully installed gif-3.0.0
WARNING: You are using pip version 21.2.2; however, version 22.1.2
is available.
You should consider upgrading via the '/opt/anaconda3/bin/python -
m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

In [16]:
```
#Example taken from
#https://github.com/mr-easy/Gibbs-Sampling-Visualized/blob/mast
import numpy as np
import matplotlib.pyplot as plt
import gif
from IPython.display import Image
from random import random
```

In [17]:
```
# For plotting Gaussian contours
from plotting_util import plot_gaussian_from_points
from plotting_util import plot_gaussian_from_parameters
```

In [18]:
```
def conditional_sampler(sampling_index, current_x, mean, cov):
    conditioned_index = 1 - sampling_index
    a = cov[sampling_index, sampling_index]
    b = cov[sampling_index, conditioned_index]
    c = cov[conditioned_index, conditioned_index]

    mu = mean[sampling_index] + (b * (current_x[conditioned_ind
    sigma = np.sqrt(a-(b**2)/c)
    new_x = np.copy(current_x)
    new_x[sampling_index] = np.random.randn()*sigma + mu
    return new_x
```

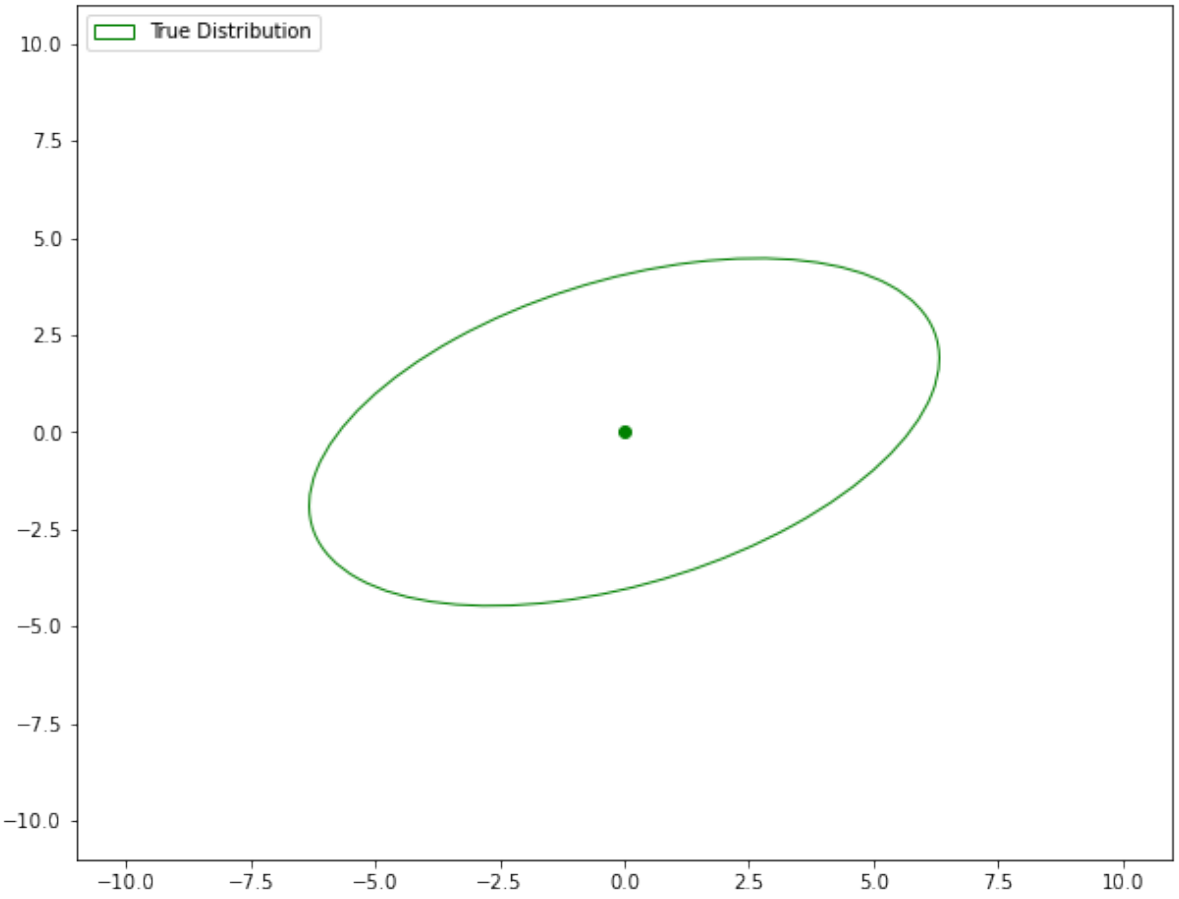
In [19]:

```
@gif.frame
def plot_samples(samples, num_samples, tmp_points, num_tmp, tit
    fig = plt.figure(figsize=(10, 8))
    ax = fig.gca()

    # Plot the true distribution
    plot_gaussian_from_parameters(mean, cov, ax, n_std=2, edgec

    # Plot sampled points
    ax.scatter(samples[:num_samples, 0], samples[:num_samples,
    ax.scatter(samples[0, 0], samples[0, 1], marker='*', c='g',

    # Plot samples from conditional distribution
    ax.scatter(tmp_points[:num_tmp, 0], tmp_points[:num_tmp, 1]

    # Keeping the axes scales same for good GIFS
    ax.set_xlim(xlims)
    ax.set_ylim(ylims)

    # Plot lines
    if(num_tmp > 0):
        ax.plot([samples[num_samples-1, 0], tmp_points[num_tmp-
                 [samples[num_samples-1, 1], tmp_points[num_tmp-
        # Plot estimated Gaussian, ignoring the starting point
        if(num_samples > 2):
            plot_gaussian_from_points(samples[1:num_samples, 0]
                                      ax, n_std=2, edgecolor='b

    ax.legend(loc='upper left')
    ax.set_title(title)
```

In [20]:
```
def gibbs_sampler(initial_point, num_samples, mean, cov, create

    frames = []  # for GIF
    point = np.array(initial_point)
    samples = np.empty([num_samples+1, 2])  #sampled points
    samples[0] = point
    tmp_points = np.empty([num_samples, 2]) #inbetween points

    for i in range(num_samples):

        # Sample from p(x_0|x_1)
        point = conditional_sampler(0, point, mean, cov)
        tmp_points[i] = point
        if(create_gif):
            frames.append(plot_samples(samples, i+1, tmp_points

        # Sample from p(x_1|x_0)
        point = conditional_sampler(1, point, mean, cov)
        samples[i+1] = point
        if(create_gif):
            frames.append(plot_samples(samples, i+2, tmp_points

    if(create_gif):
        return samples, tmp_points, frames
    else:
        return samples, tmp_points
```

In [21]:
```
mean = np.array([0, 0])
cov = np.array([[10, 3],
                [3, 5]])
```

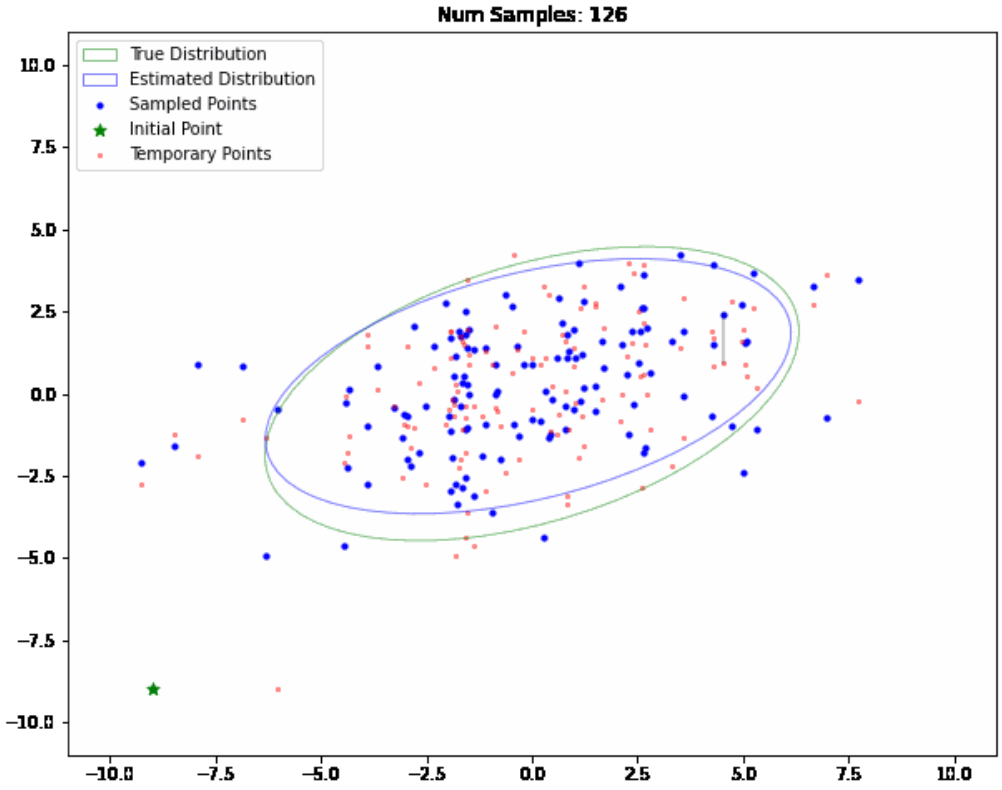
In [22]:
```
# Plot true distribution
fig = plt.figure(figsize=(10, 8))
ax = fig.gca()
plot_gaussian_from_parameters(mean, cov, ax, n_std=2, edgecolor
ax.scatter(mean[0], mean[1], c='g')
ax.set_xlim((-11, 11))
ax.set_ylim((-11, 11))
ax.legend(loc='upper left')
plt.show()
```



In [23]:
```
initial_point = [-9.0, -9.0]
num_samples = 500
samples, tmp_points, frames = gibbs_sampler(initial_point, num_
```

In [24]:
```
# Creating the GIF
gif.save(frames, "gibbs.gif", duration=150)
Image(filename="gibbs.gif")
```

Out[24]:



In [1]:

```
# To run slideshow type jupyter nbconvert /Users/giancarlomanzi
# from terminal
#/Users/giancarlomanzi/Documents/Box Sync BackUp PC Lavoro 2406

#This is to let you have larger fonts...
from IPython.core.display import HTML
HTML("""
<style>

div.cell { /* Tunes the space between cells */
margin-top:1em;
margin-bottom:1em;
}

div.text_cell_render h1 { /* Main titles bigger, centered */
font-size: 2.2em;
line-height:1.4em;
text-align:center;
}

div.text_cell_render h2 { /*  Parts names nearer from text */
margin-bottom: -0.4em;
}


div.text_cell_render { /* Customize text cells */
font-family: 'Times New Roman';
font-size:1.5em;
line-height:1.4em;
padding-left:3em;
padding-right:3em;
}
</style>
""")
```

Out[1]:

In [ ]:

```

```