Introduction to Bayesian Inference in R

Robert Dodier

This document and associated material is on Github: robert-dodier/bayesian-inference-r-2016

What is Bayesian inference?

- Bayesian inference = laws of probability applied to any proposition
- Proposition = statement about uncertain variables
- Variables = observable quantities, model parameters, latent (hidden) variables, hypotheses, etc
- Fundamental operation is to compute conditional probabilities
- i.e. probability of an interesting proposition ...
    - given (conditional on) some data (i.e. whatever is fixed, either by observation or assumption)
- Bayesian inference is a framework within which we construct a way to handle any problem involving uncertain propositions

What to do about uninteresting variables

- What about variables which are neither interesting nor given??
- Laws of probability dictate the Right Thing To Do:
  - namely, integrate joint probability of "interesting" and "not interesting" over "not interesting"
  - formally,

$$P(\text{interesting}|\text{given}) = \int P(\text{interesting}, \text{uninteresting}|\text{given}) d\text{uninteresting}$$

(where integral is a summation if uninteresting is discrete)

- This is the motivation for the computational stuff we'll see later

Some problems involving uncertain propositions

- Missing data
- Forecasts
- Diagnosis
- Model selection
- Model parameters
- "What if" scenarios

Reasonable, non-Bayesian approaches

- Missing data: plug in estimate of missing variable
- Forecasts: output forecast = best guess
- Diagnosis: output diagnosis = best guess
- Model selection: choose "the best" model
- Model parameters: choose best parameters
- "What if" scenarios: plug in example inputs into model and get output

Approaches as enriched by a Bayesian perspective

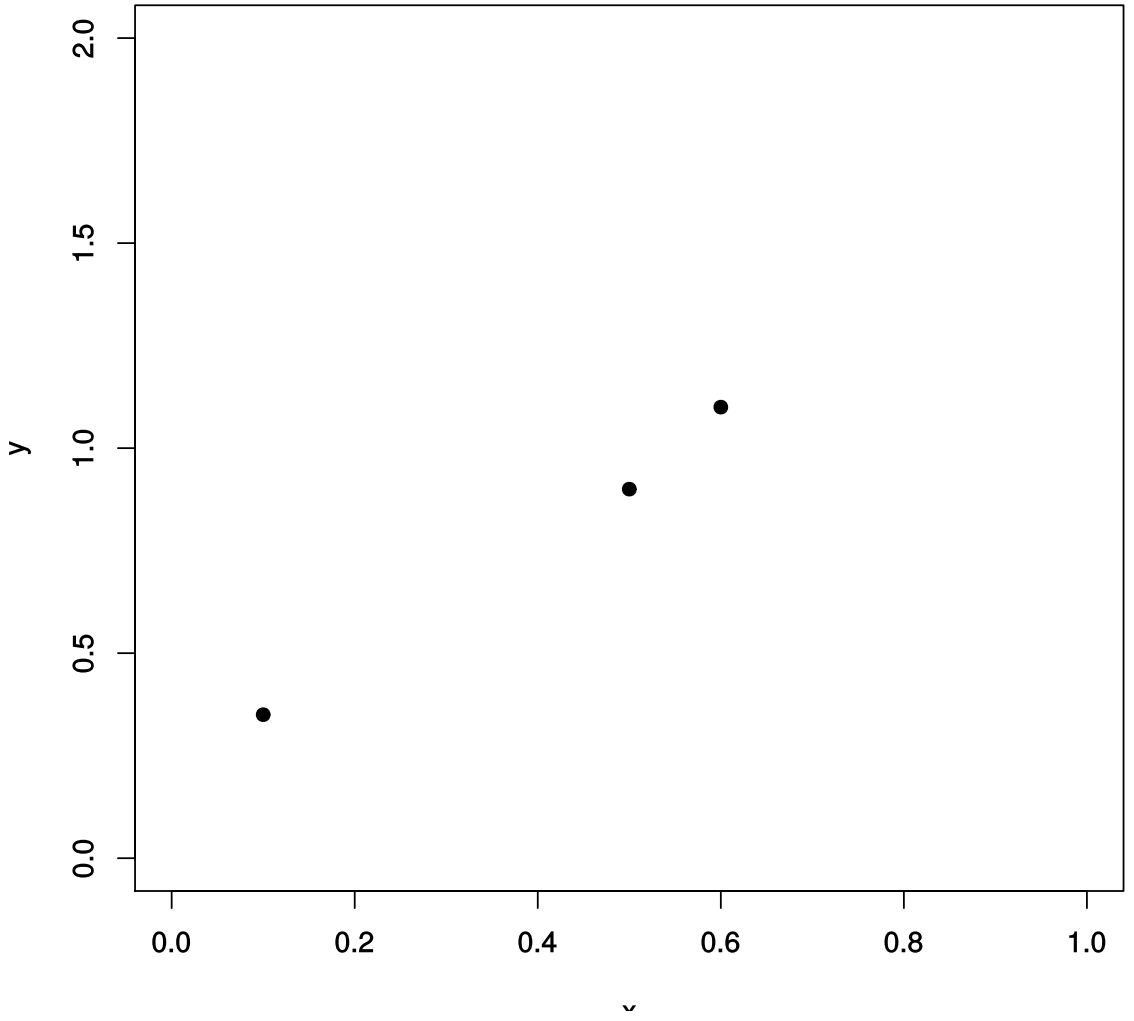
- Missing data: consider all possible values of missing variable
- Forecasts: consider all possible output values
- Diagnosis: consider all possible diagnoses
- Model selection: consider all possible models
- Model parameters: consider all possible parameters
- "What if" scenarios: consider all possible results

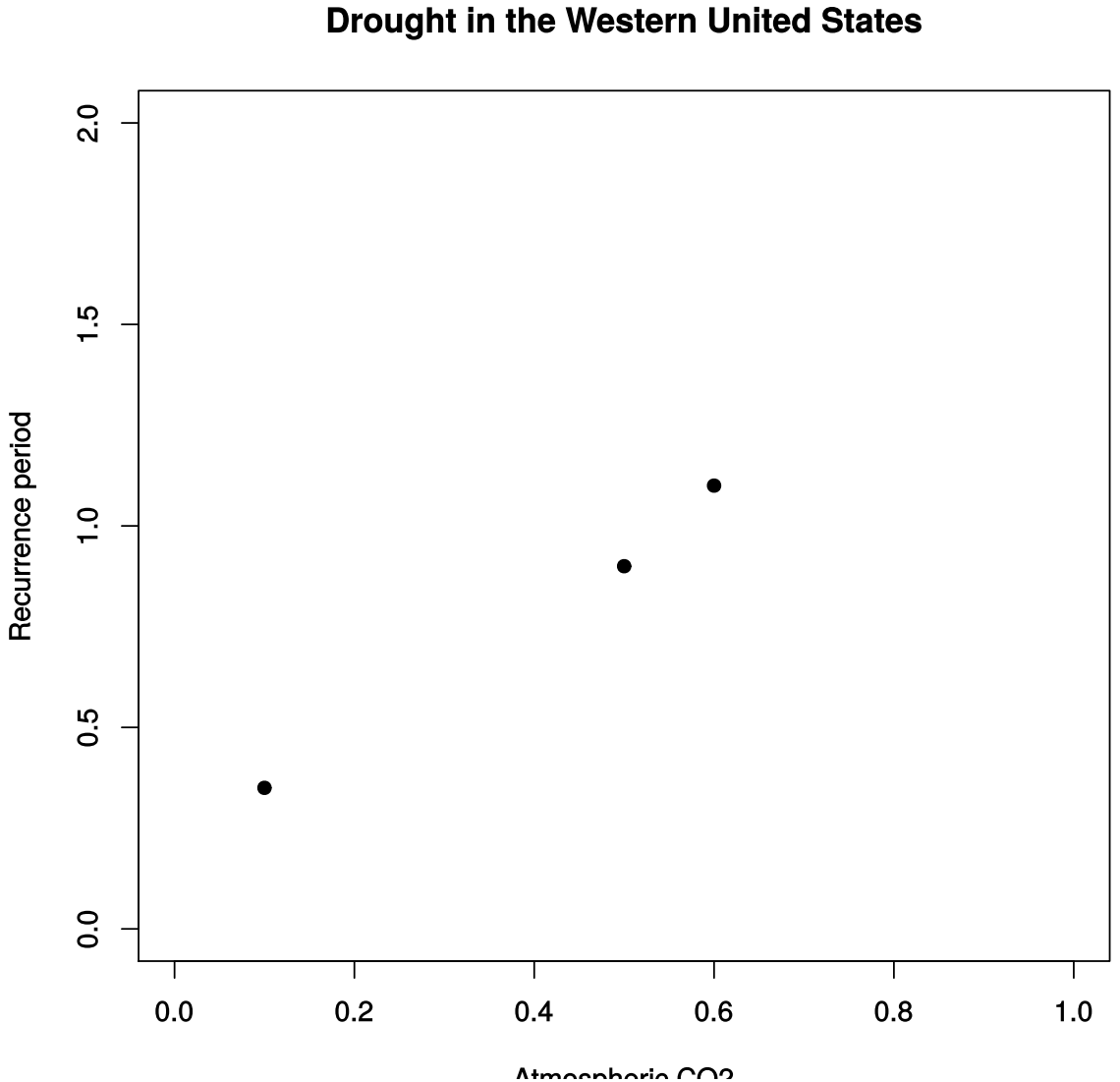
A closer look at the model parameters problem

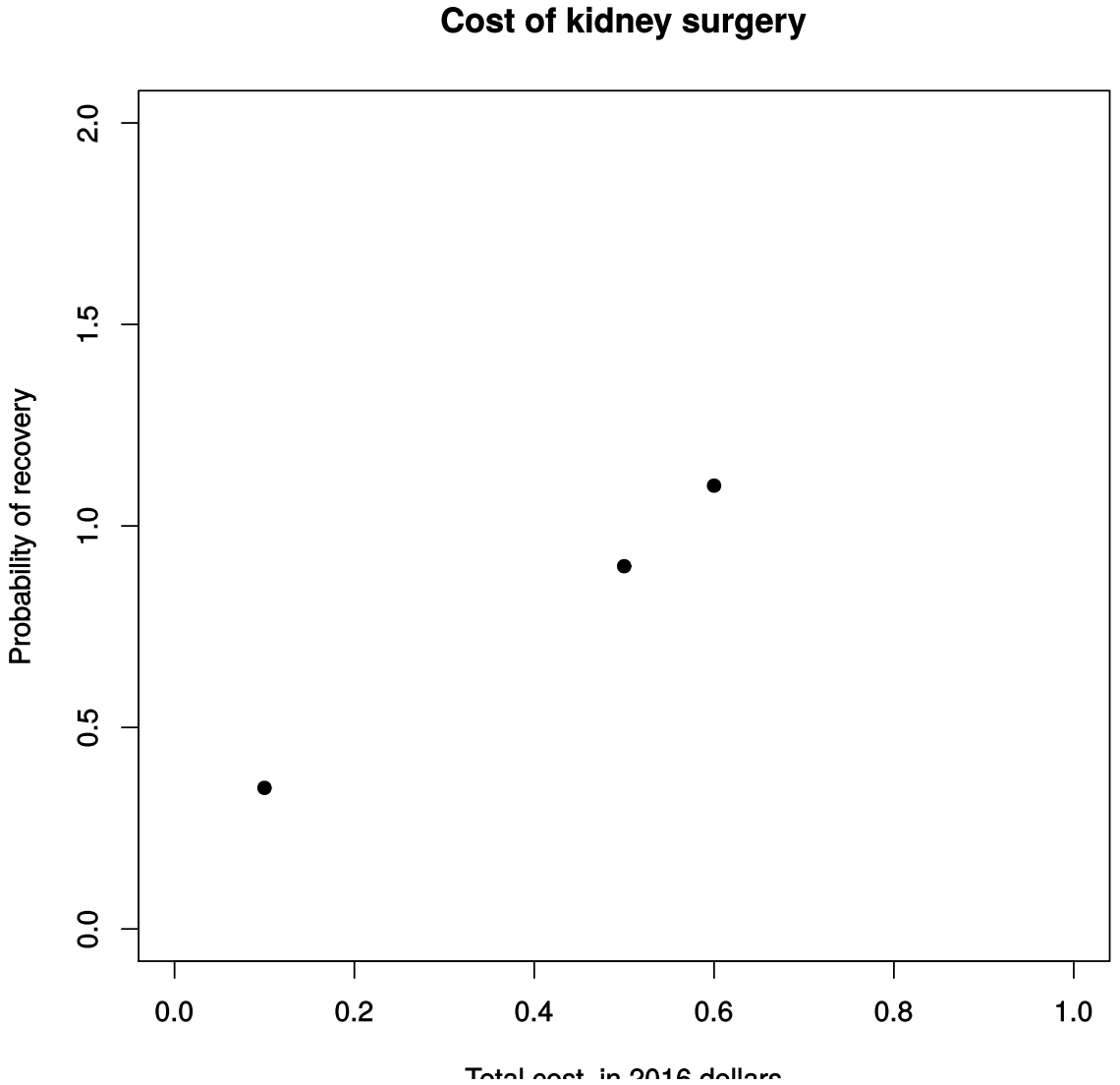
- Goal is to construct posterior distribution of parameters given data
- In some cases an exact distribution of parameters can be constructed exactly or approximately
- In other cases, can't construct it explicitly but can construct an algorithm to sample from posterior
- ... then to produce e.g. forecasts, sample from posterior and use parameters to generate output
- ... and the result is a posterior distribution over the output

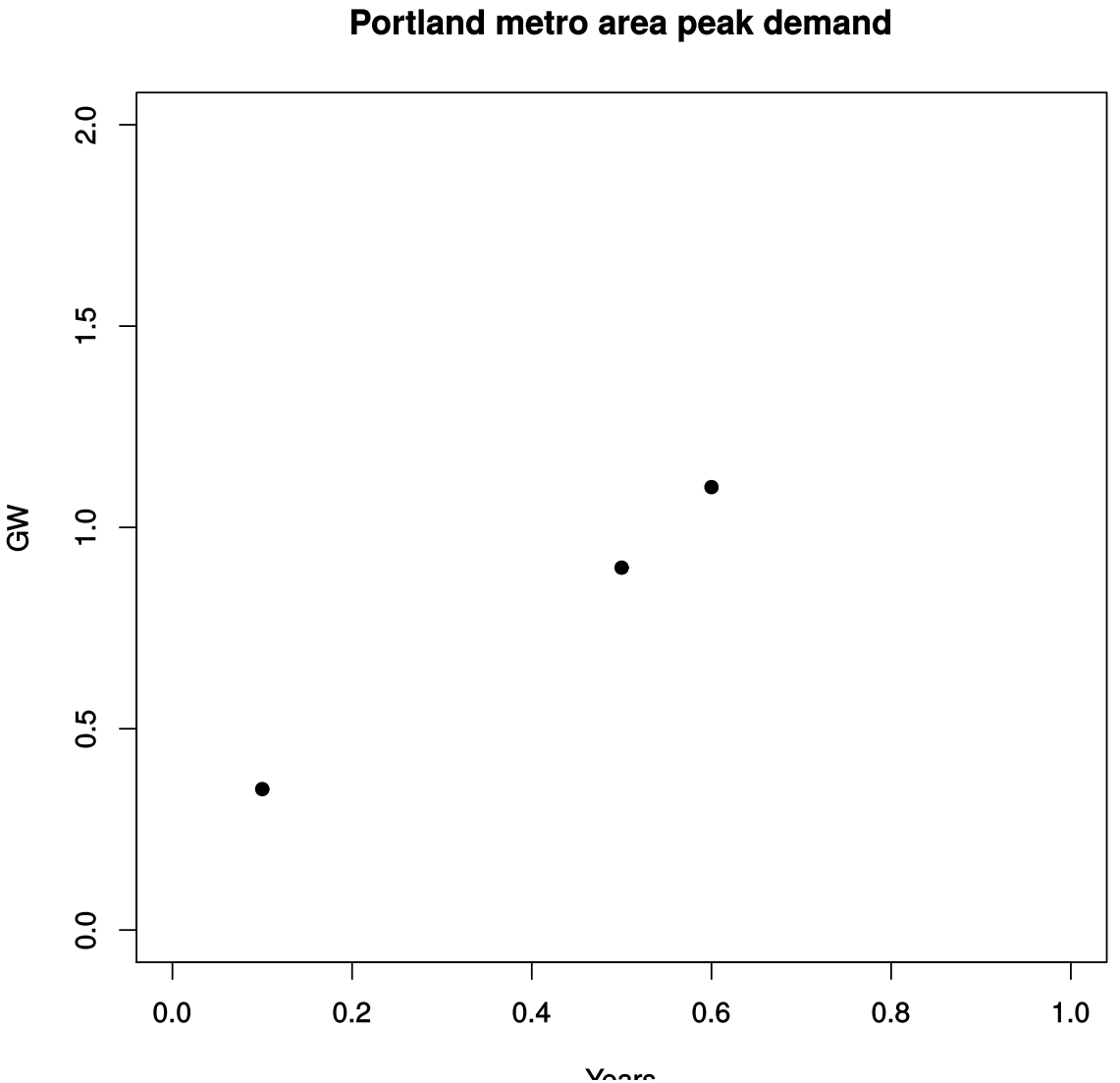
Linear regression example

- We have a small data sample
- We'll assume the data are described by a linear model $y = a_0 + a_1 x + \text{noise}$
- We'd like to compute a forecast at a given point
- ... taking uncertainty in the model parameters into account

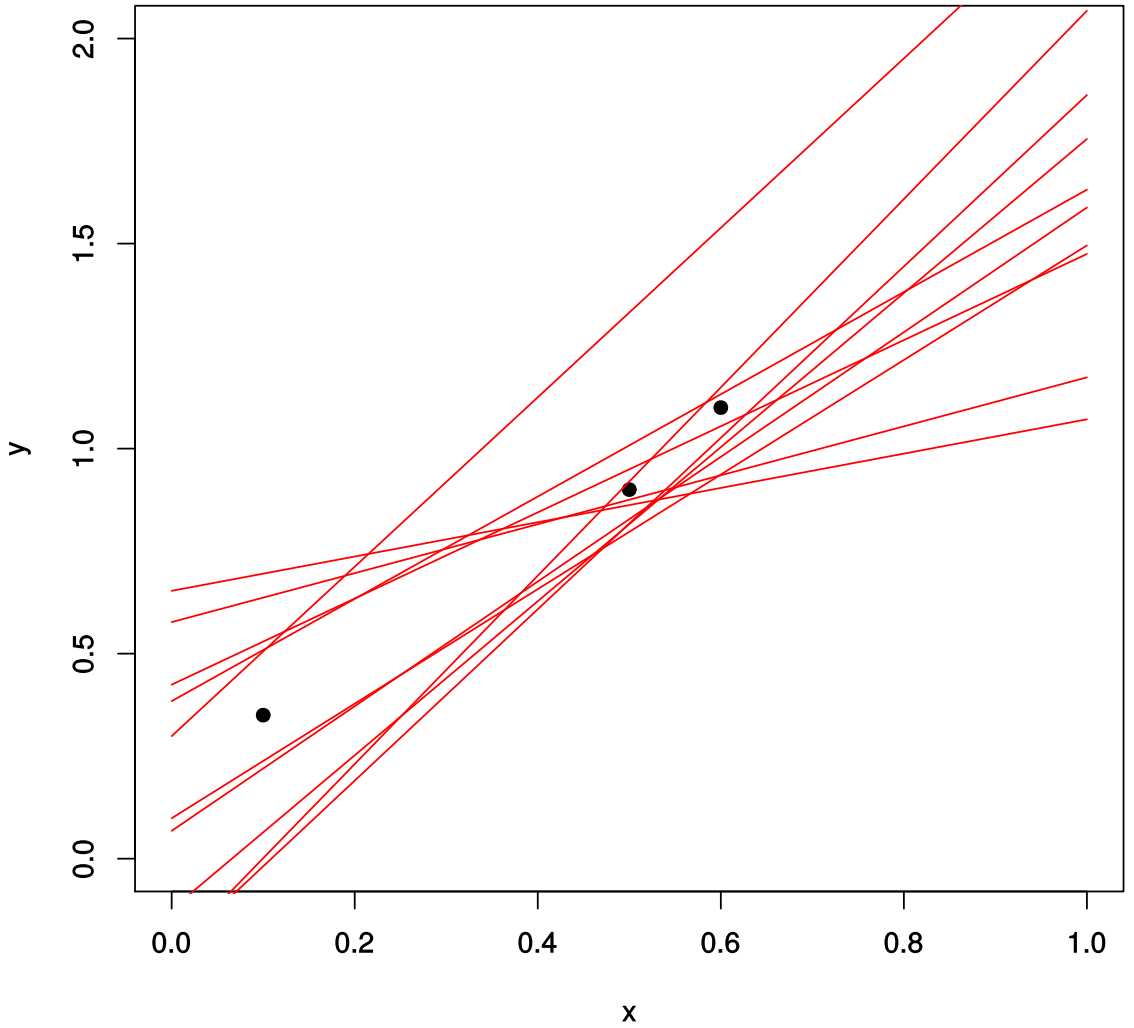

y
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Drought in the Western United States
Recurrence period
0.0
0.5
1.0
1.5
2.0
0.0
0.2
0.4
0.6
0.8
1.0
Atmospheric CO2


Cost of kidney surgery
Probability of recovery
0.0
0.5
1.0
1.5
2.0
0.0
0.2
0.4
0.6
0.8
1.0


Portland metro area peak demand
GW
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Years

There are many possible lines, some more plausible, some less



Let's set our goal to compute the posterior distribution $p(y|x, \text{data})$

- For that, we need to average over the variables we don't know, namely a_0 and a_1.
- Specifically we need to compute

  $$\int\int p(y|x, a_0, a_1)p(a_0, a_1|\text{data})p(a_0, a_1)da_0 da_1$$

Computing integrals -- (1) symbolic

- Stuff like: $\int_0^x \alpha e^{-\alpha t} \beta e^{-\beta(x-t)}\, dt = \frac{\alpha\beta}{\beta-\alpha}(e^{-\alpha x} - e^{-\beta x})$
- Gives insight if assumptions are satisfied
- Limited applicability
- In R, maybe rsympy or maybe ryacas which are glue code for computer algebra systems

Computing integrals -- (2) quadrature formulas

- stats::integrate adaptive subdivision based on rules exact for polynomials

Linear regression example using quadrature

`py.given.x` computes $p(y|x, \text{data})$

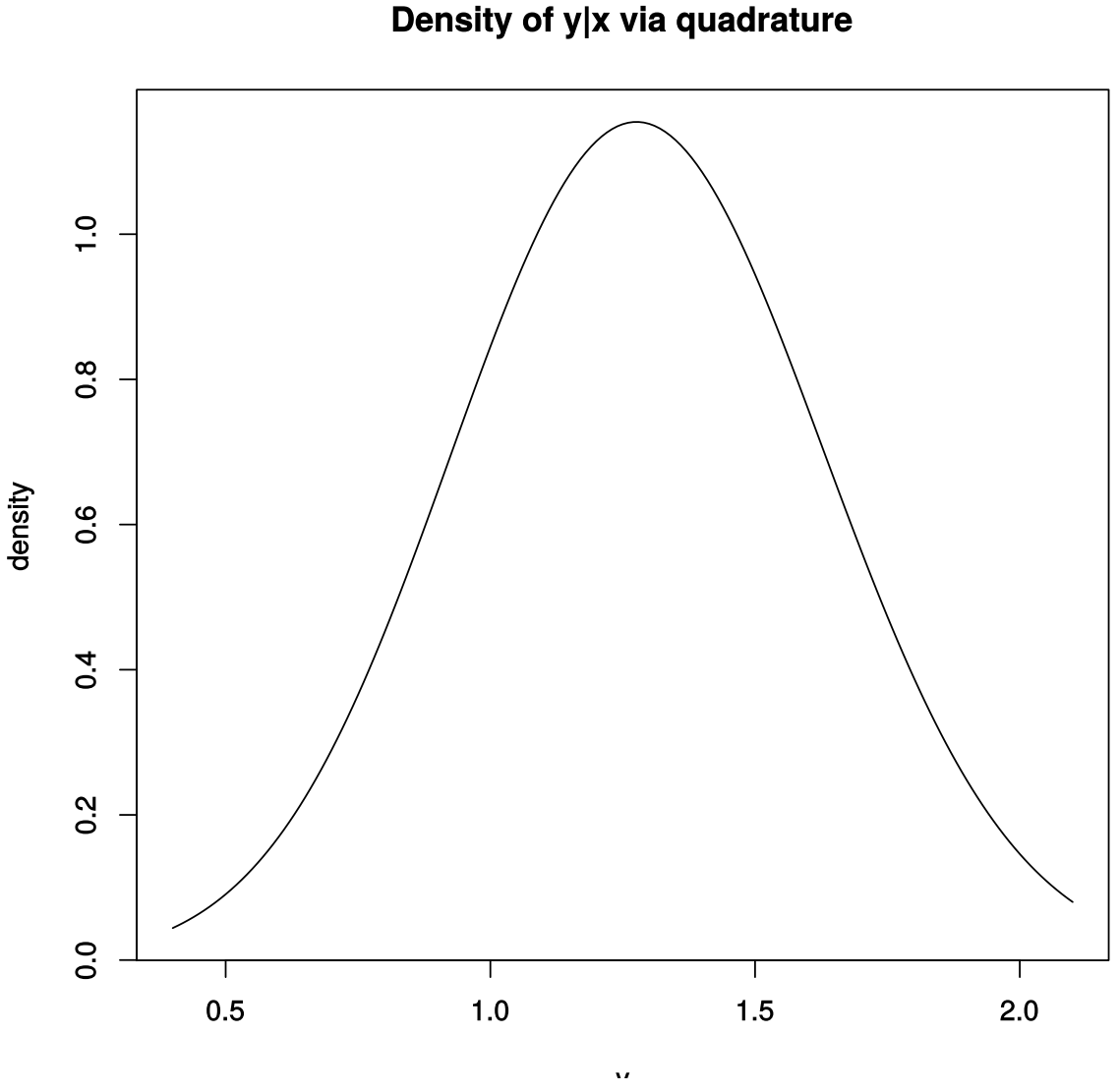
```
py.given.x <-
  function (y, x) {
    integrate (
      function (a.1) {
        sapply (a.1,
                function (a.1) {
                  integrate (function (a.0) {
                               py.given.x.a0.a1.sigma.e (y, x, a.0, a.1, 0.25) *
                                 P2 (a.0, a.1)},
                             -Inf, Inf)$value})},
      0, Inf)}
```

Integrand is $p(y|x, a_0, a_1)p(a_0, a_1|\text{data})p(a_0, a_1)$

Calculating a posterior distribution via quadrature

```
yy <- seq (from=0.4, to=2.1, by=0.01)
sapply (yy, function (y) {py.given.x (y, 0.75) $ value}) -> pp
I <- sum(pp)*0.01
pp <- pp/I

plot (x=yy, y=pp, type="l")
```


Density of y|x via quadrature
density
0.0
0.2
0.4
0.6
0.8
1.0
0.5
1.0
1.5
2.0

About quadrature formulas

- Works well in 1 dimension, tolerable in a few more
- Unworkable in many dimensions (a common application scenario)

Computing integrals -- (3) simple Monte Carlo

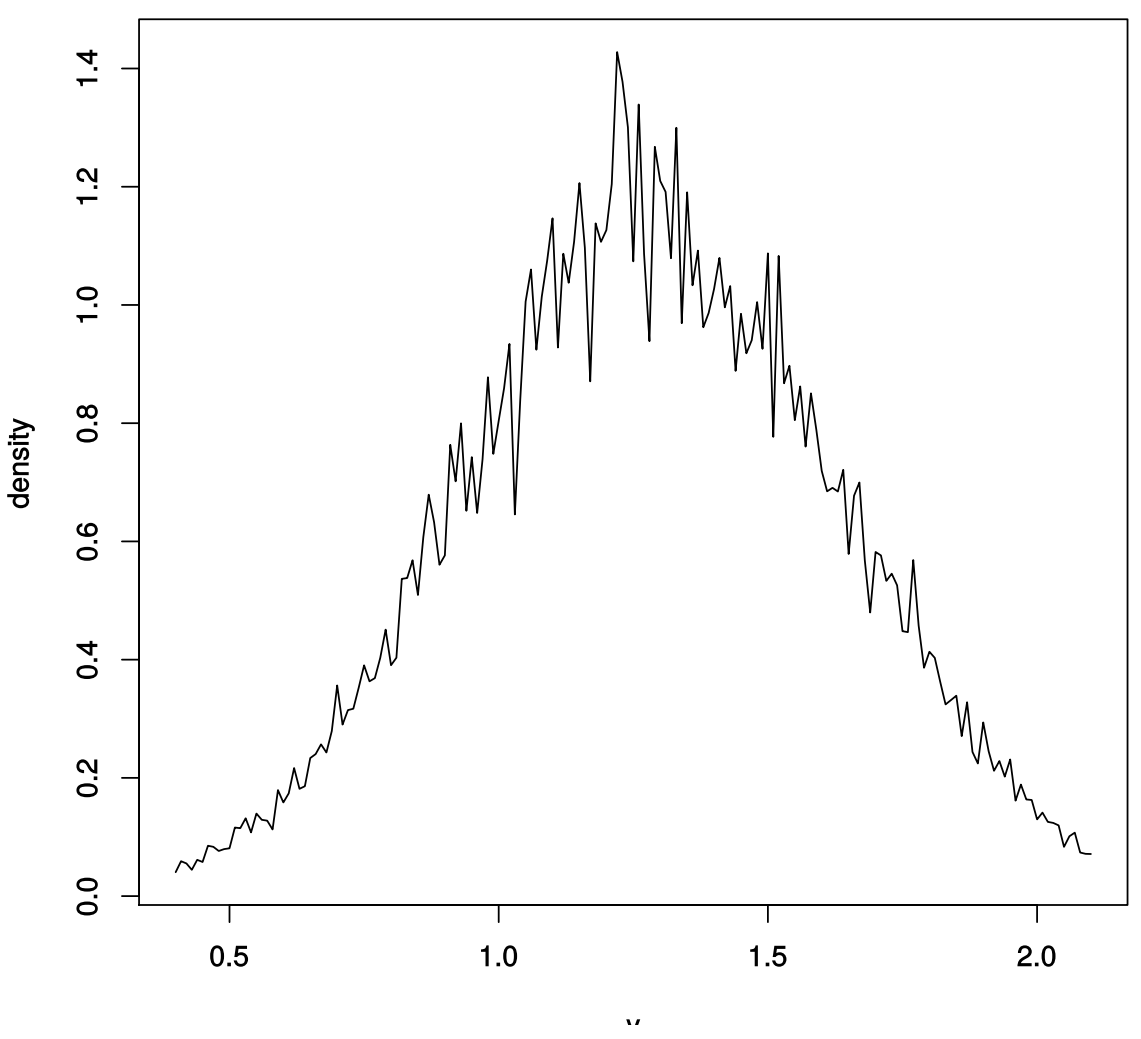
- runif, rnorm, rgamma, etc generate pseudorandom numbers
- Approximate integral as average over samples:
  $\int f(x)p(x)dx \approx \frac{1}{n}\sum_i f(x_i)$

Linear regression example using simple Monte Carlo

```
py.given.x.via.monte.carlo <-
  function (y, x) {
    mean (mapply (function (a.0, a.1) {
                     py.given.x.a0.a1.sigma.e (y, x, a.0, a.1, 0.25) *
                     L(a.0, a.1, 0.25) *
                     psigma.e(0.25)},
                   rnorm (1000, mean=0, sd=1),
                   rgamma (1000, shape=2, scale=1)))}
```

Calculating a posterior distribution via simple Monte Carlo

```
pp <- sapply (yy, function (y) {py.given.x.via.monte.carlo (y, 0.75)})
I <- sum(pp)*0.01
pp <- pp/I
```


Density of y|x via simple Monte Carlo
density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0.5
1.0
1.5
2.0

About simple Monte Carlo

- Easy to apply, may be inefficient
- Limited applicability (difficulty of constructing sampling distribution)
    - … this is the motivation for Markov chain Monte Carlo

Computing integrals -- (4) Markov chain Monte Carlo

- More general than ordinary Monte Carlo
- If one can't explicitly construct sampling distribution, construct Markov chain s.t. stationary distribution is desired distribution
- For this Markov chain, averages over samples are equal to averages over the desired distribution
- From current sample, jump somewhere else; maybe accept the new sample. Repeat many times.

Markov chain Monte Carlo example

We'll work with a function proportional to the distribution we want to sample;

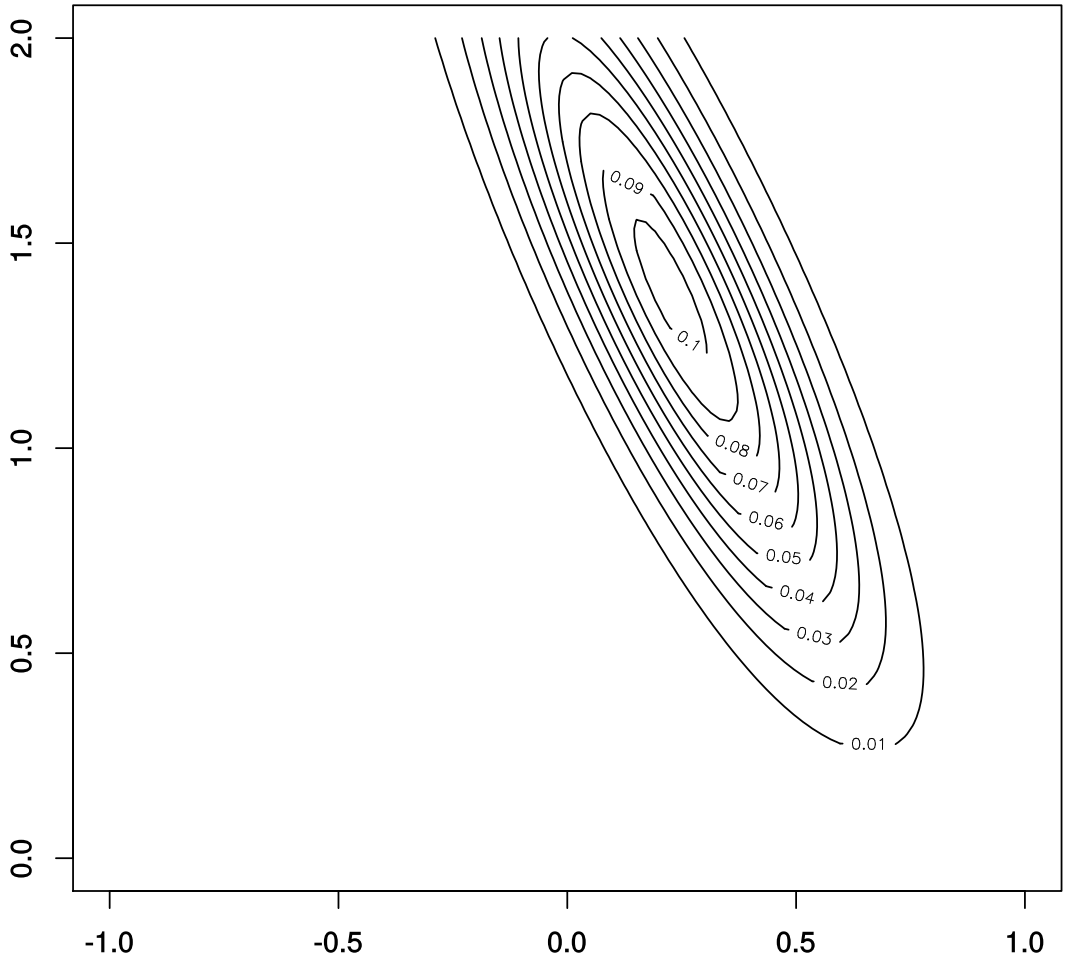
namely the product of the likelihood function and the prior for the parameters.

Here is the likelihood function for the linear regression example:

```
L <- function (a.0, a.1, sigma.e) {
          py.given.x.a0.a1.sigma.e (data.y[1], data.x[1], a.0, a.1, sigma.e) *
          py.given.x.a0.a1.sigma.e (data.y[2], data.x[2], a.0, a.1, sigma.e) *
          py.given.x.a0.a1.sigma.e (data.y[3], data.x[3], a.0, a.1, sigma.e)}
```

and here is the prior:

```
pa.0 <- function (a.0) {exp(-0.5*a.0^2)/sqrt(2*pi)}
pa.1 <- function (a.1) {a.1*exp(-a.1)}
```


2.0
1.5
1.0
0.5
0.0
-1.0
-0.5
0.0
0.5
1.0
0.09
0.1
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01

Proposal distribution which is just a Gaussian bump:

```
Q <- function (a.0, a.1) {rnorm (2, mean=c(a.0, a.1), sd=0.05)}
```

Generate a new proposed point:

```
p1 <- Q (p0[1], p0[2])
```

Compute acceptance ratio:

```
r <- L (p1[1], p1[2]) / L (p0[1], p0[2])
```

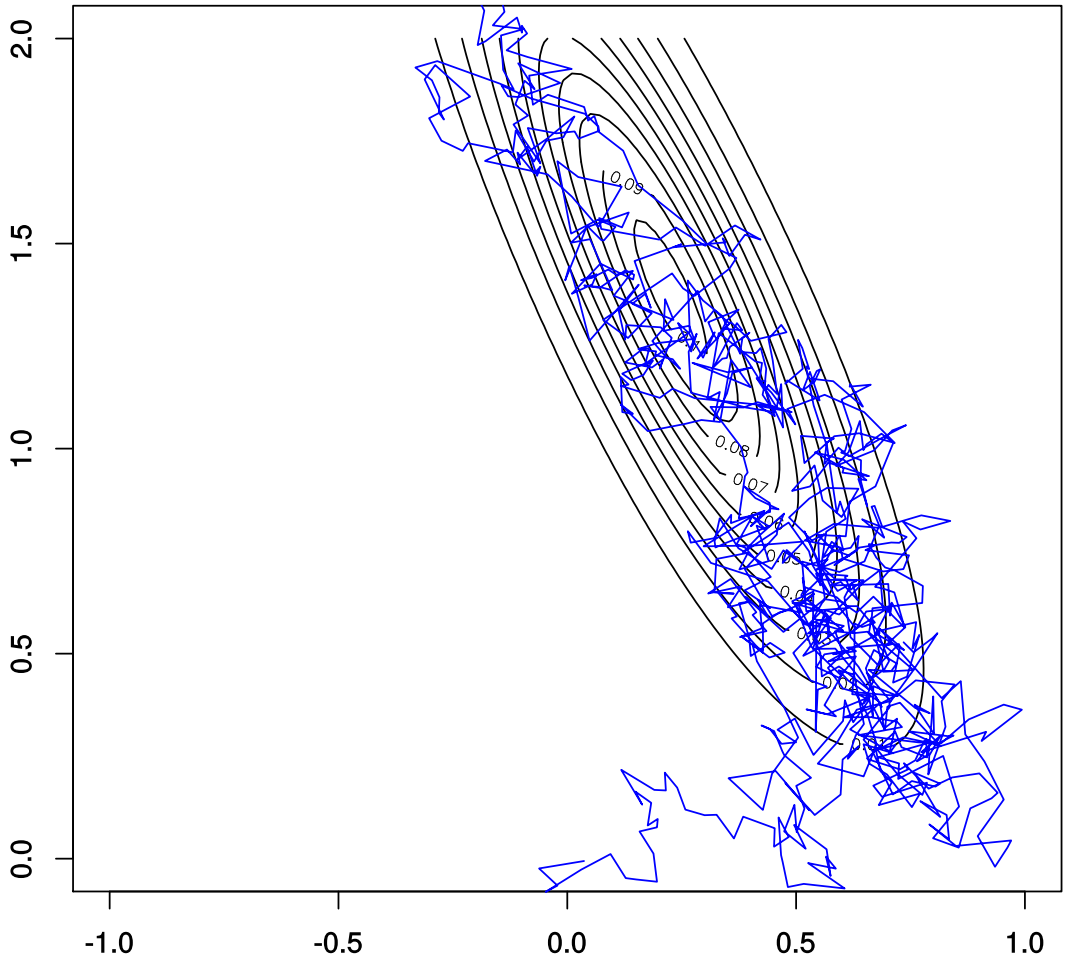
Accept or reject proposal:

```
if (r > 1 || r > runif(1)) {p0 <- p1}
```

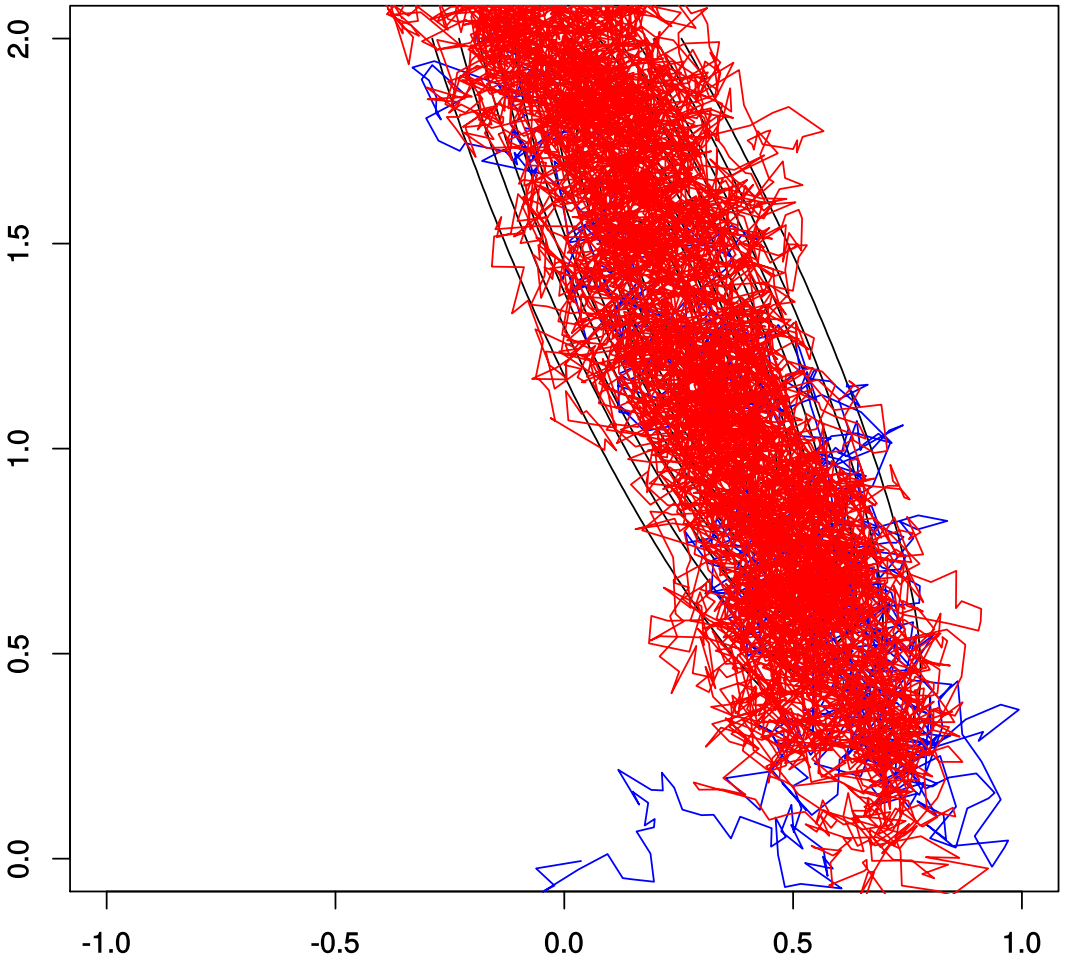
Generate a sequence of samples given a starting point `xy0`:

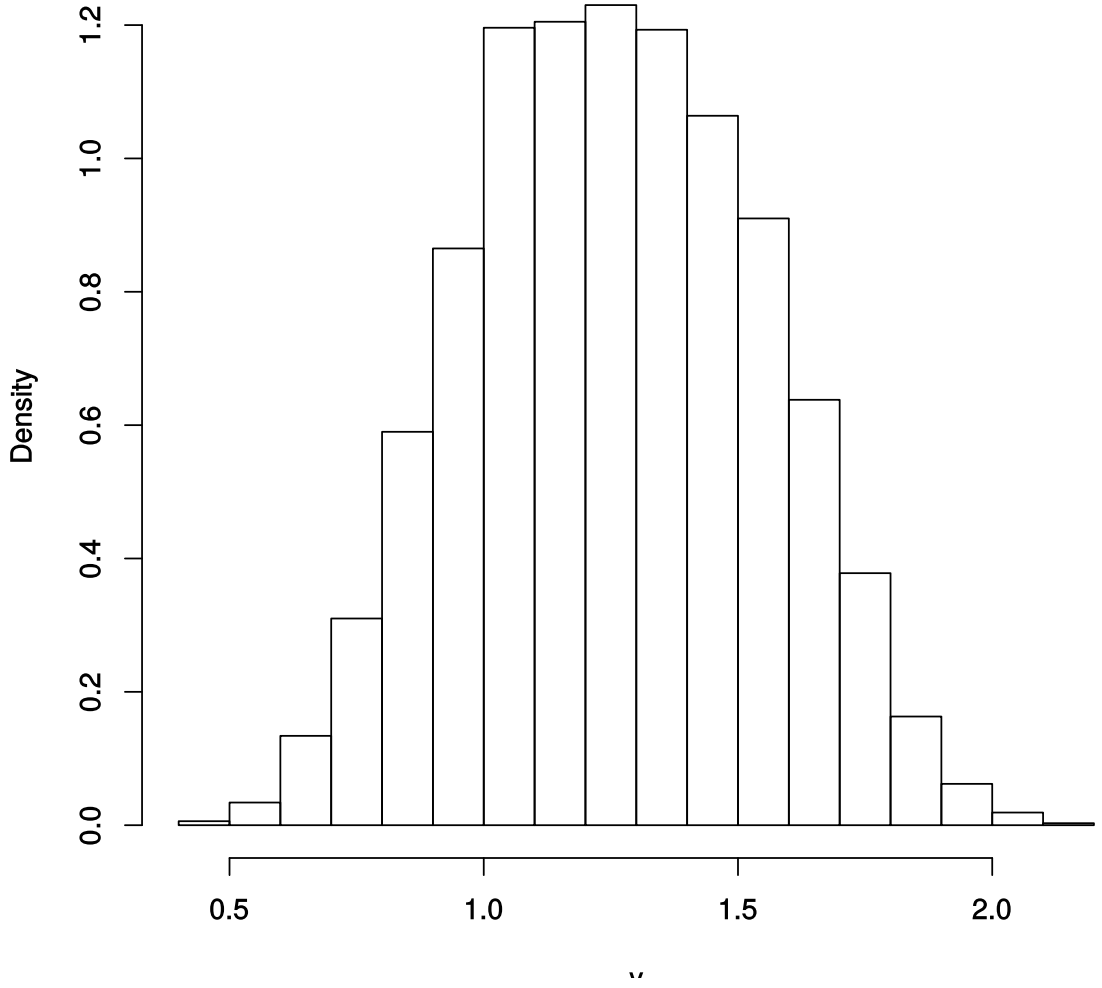
```
mcmc.sequence <- function (n, p0) {
  a.0 <- vector (length=n)
  a.1 <- vector (length=n)
  a.0[1] <- p0[1]
  a.1[1] <- p0[2]
  for (i in 2:n) {
    p1 <- Q (p0[1], p0[2])
    r <- L (p1[1], p1[2]) / L (p0[1], p0[2])
    if (r > 1 || r > runif(1)) {p0 <- p1}
    a.0[i] <- p0[1]
    a.1[i] <- p0[2]
  }
  list (x=a.0, y=a.1)
}
```

"Burn-in" sequence -- initial samples not representative of distribution



Running the chain longer yields more representative samples




Density of y|x via Markov chain Monte Carlo
Density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0.5
1.0
1.5
2.0

About Markov chain Monte Carlo

- MCMC slower than ordinary MC due to autocorrelation, but more general
- Writing a sampler is a distraction that obscures the model

Stan, a package for Bayesian inference

- Stan has its own modeling language, primarily declarative
  - User states how variables are related, Stan figures out calculations
- Inference via "no U-turn sampling" MCMC
- Also implements variational inference and optimization functions

Linear regression model in Stan

```
model {
  real mu;
  for (i in 1:3) {
    mu = alpha[1] + alpha[2] * x[i];
    y[i] ~ normal(mu, sigma_e);
  }
}

generated quantities {
  real y_pred;
  y_pred = normal_rng(alpha[1] + alpha[2] * new_x, sigma_e);
}
```

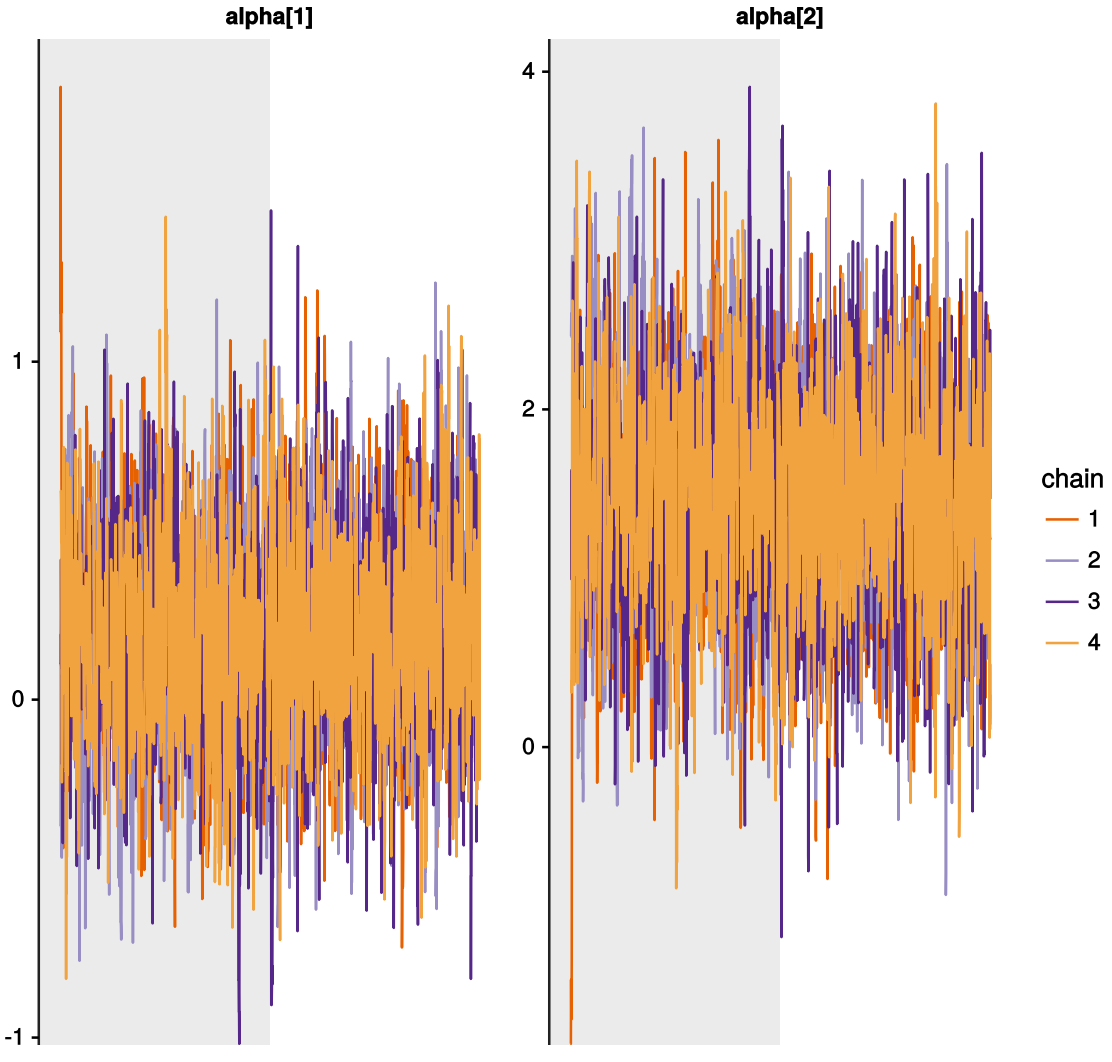
Making inferences via RStan

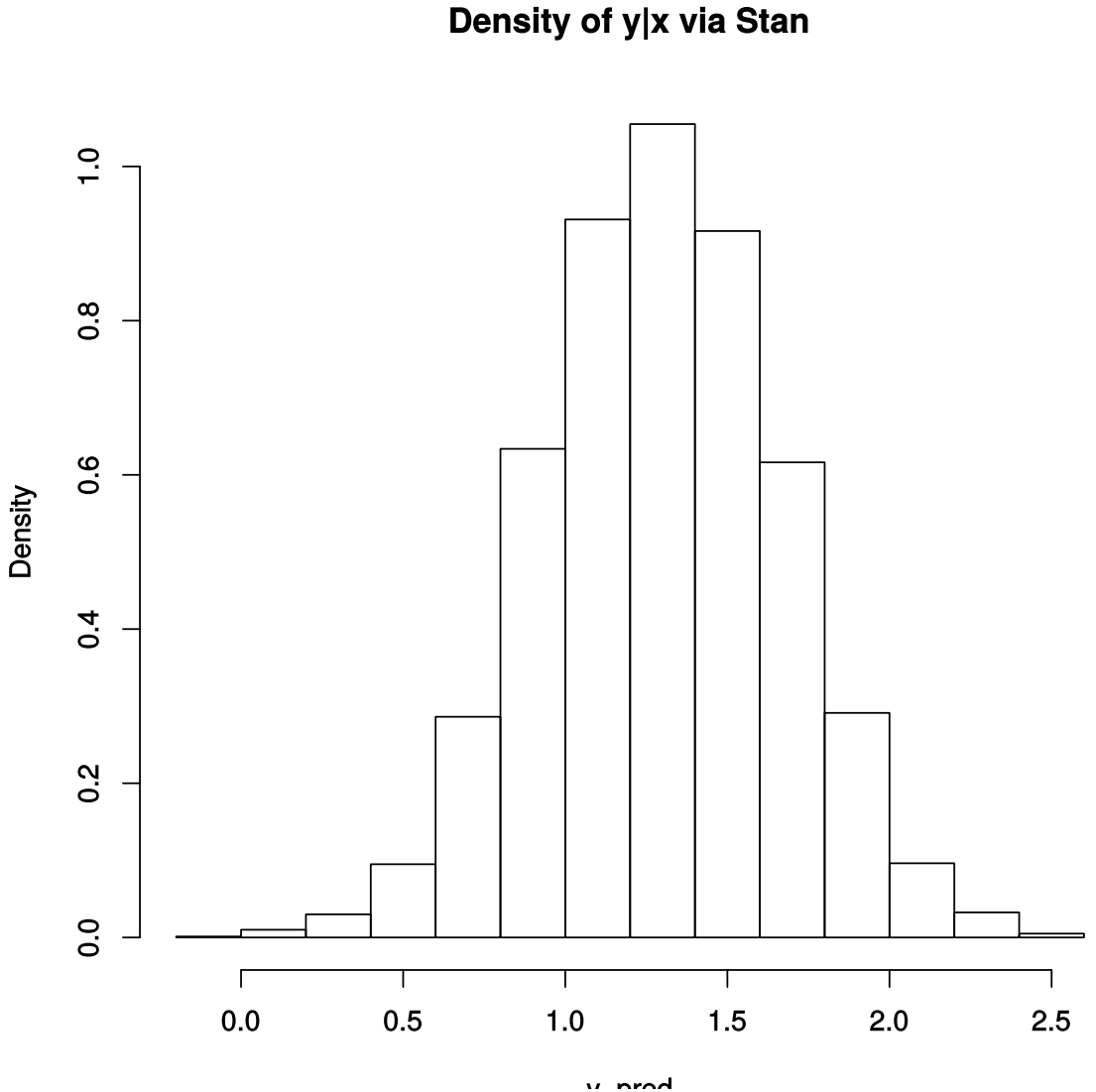
```
library (rstan)
stan.data <- list (x=c(0.1, 0.5, 0.6), y=c(0.35, 0.9, 1.1), sigma_e=0.25, new_x=0.
my.stan.model <- stan (file="model.stan", data=stan.data, iter=2000, chains=4)

traceplot (my.stan.model, pars = c("alpha"), inc_warmup = T)

y_pred_struct <- extract (my.stan.model, 'y_pred')
y_pred <- unlist (y_pred_struct, use.names=F)
hist (y_pred, freq=F, main="Density of y|x via Stan")
```

Samples from the Markov chain for the parameters




Density of y|x via Stan
Density
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
2.0
2.5

What use is a probability?

- Probability calculations are part of a bigger picture of decision analysis
- Rational approach is to make decision on basis of expected utility
  - Utility = quantify value or preference
  - Probability = quantify belief
  - Expected utility = utility integrated wrt probability distribution
- Instead of using a probability by itself, consider expected gain/loss
- e.g. P(change service | data) = 0.2; what should service provider do?
  - answer has to depend on what service provider stands to gain or lose