Statistical methods for linguistic research: Advanced Tools

Shravan Vasishth

Department of Linguistics
University of Potsdam, Germany

August 9, 2015

Goals for this lecture

In this lecture I plan to discuss

1. Markov Chain Monte Carlo (MCMC) Sampling
2. Model evaluation and checking model assumptions:
    1. Using lmer.
    2. Using Stan or JAGS.
    3. The Box-Cox procedure for stabilizing variance.

Monte Carlo integration

It sounds fancy, but basically this amounts to sampling from a distribution, and computing summaries like the mean.
Formally, we calculate E(f(X)) by drawing samples $\{X_1, \ldots, X_n\}$ and then approximating:

$$E(f(X)) \approx \frac{1}{n} \sum f(X) \tag{1}$$

For example:

```
x<-rnorm(1000,mean=0,sd=1)
mean(x)

## [1] 0.02455744
```

Monte Carlo integration

We can increase sample size to as large as we want.
We can also compute quantities like $P(X < 1.96)$ by sampling:

```
mean(x<1.96)

## [1] 0.974
```

MCMC sampling

1. So, we can compute summary statistics using simulation if we know the distribution we are sampling from.
2. However, if we only know up to proportionality the form of the distribution to sample from, how do we get these samples to summarize from? Recall:
   Posterior $\propto$ Lik Prior
3. Markov Chain Monte Carlo (MCMC) methods provide that capability: they allow you to sample from distributions you only know up to proportionality.

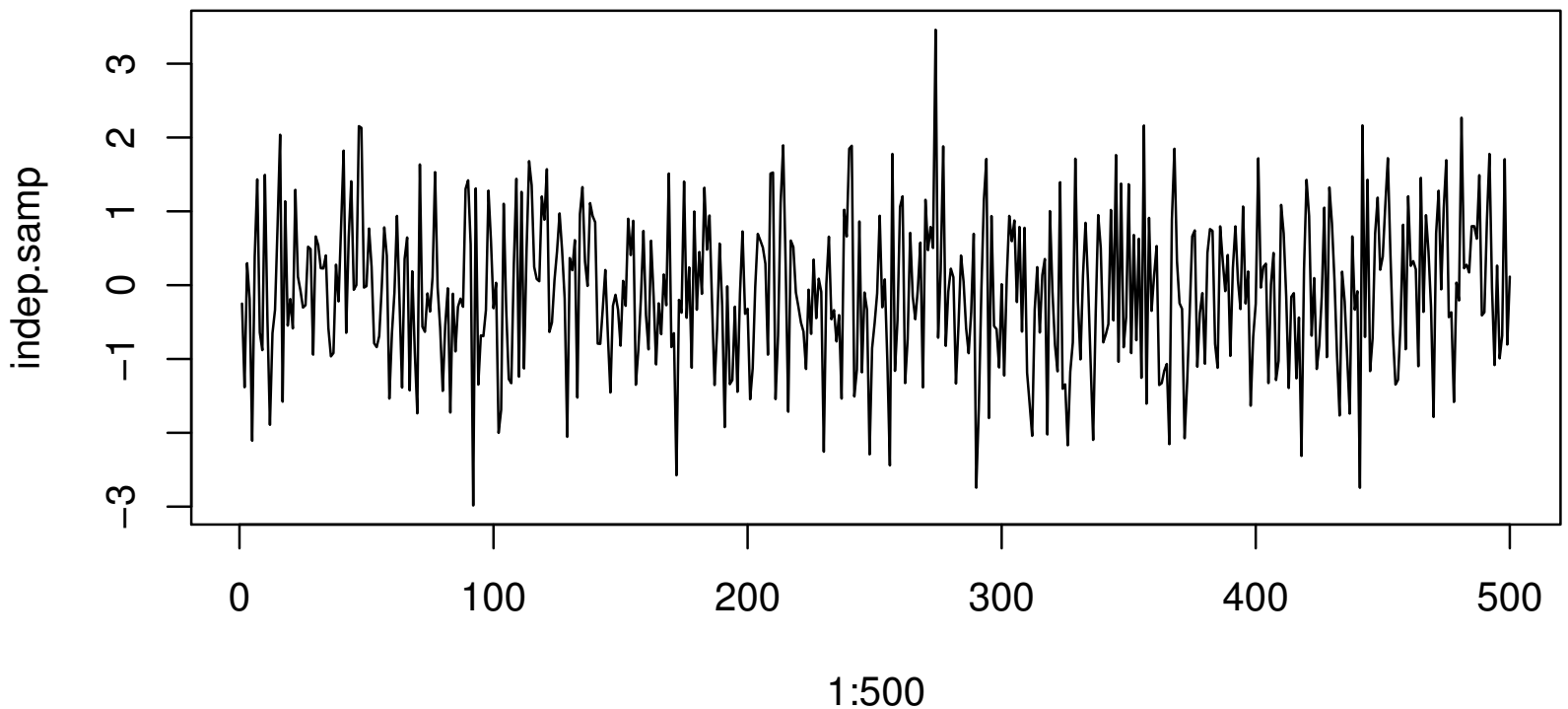
Markov chain sampling

We have been doing non-Markov chain sampling:

```
indep.samp<-rnorm(500,mean=0,sd=1)
head(indep.samp)

## [1] -0.2497943 -1.3832264  0.2974847 -0.1879254 -2.1080
```

Markov chain sampling

The vector of values sampled here are statistically independent.

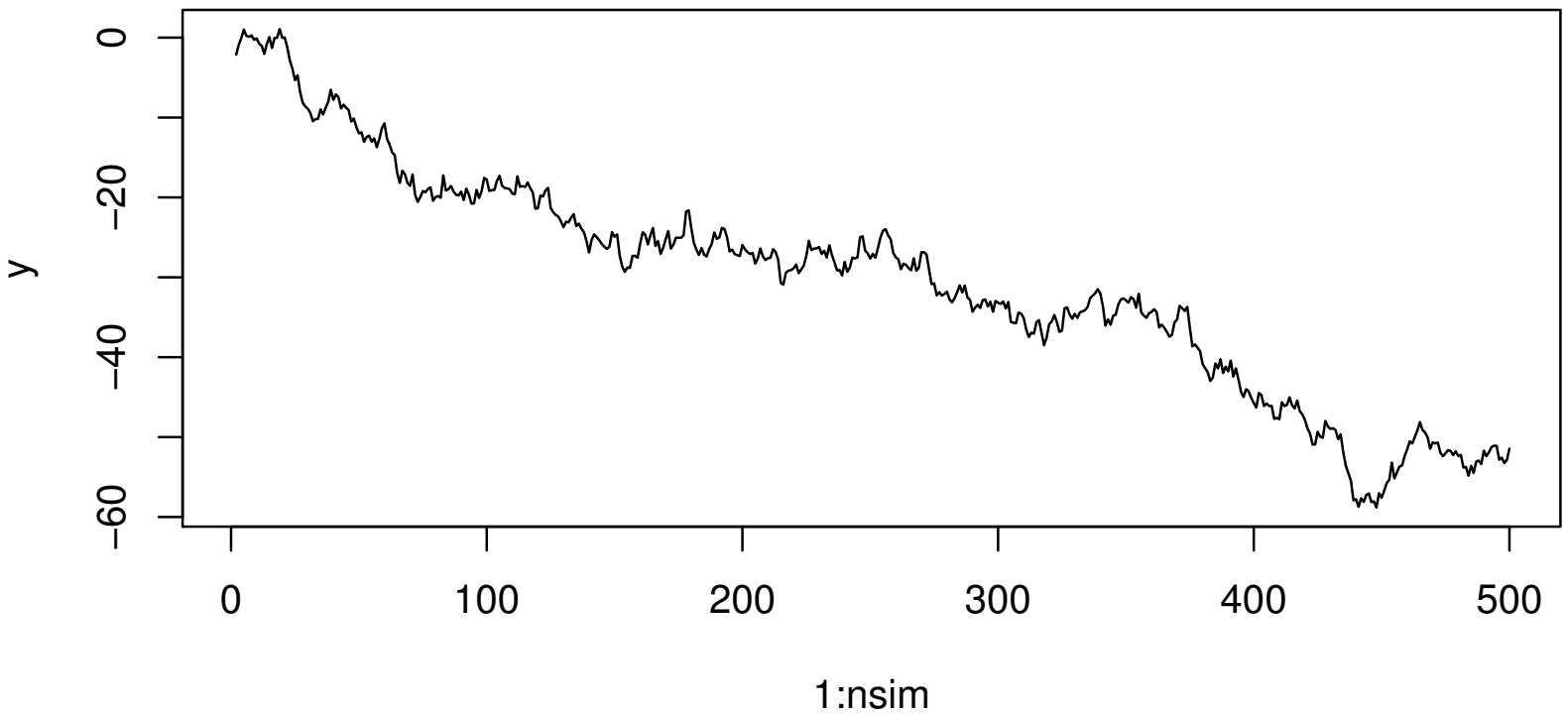


Markov chain

If the current value influences the next one, we have a Markov chain. Here is a simple Markov chain: the i-th draw is dependent on the $i-1$-th draw:

```
nsim<-500
x<-rep(NA,nsim)
y<-rep(NA,nsim)
x[1]<-rnorm(1) ## initialize x
for(i in 2:nsim){
## draw i-th value based on i-1-th value:
y[i]<-rnorm(1,mean=x[i-1],sd=1)
x[i]<-y[i]
}
```

Markov chain



MCMC sampling

The basic idea of MCMC is that we sample from an approximate distribution, and then correct or adjust the values.

Two sampling methods

Next, I will discuss two sampling methods, in order to give a flavor of how we can sample from a distribution.

1. Inverse sampling
2. Rejection sampling

The problem we have to solve is: given a distribution, say $f(x) = \frac{1}{40}(2x+3)$
how can we sample from it?
Recall that with the normal distribution it's easy: use `rnorm`.

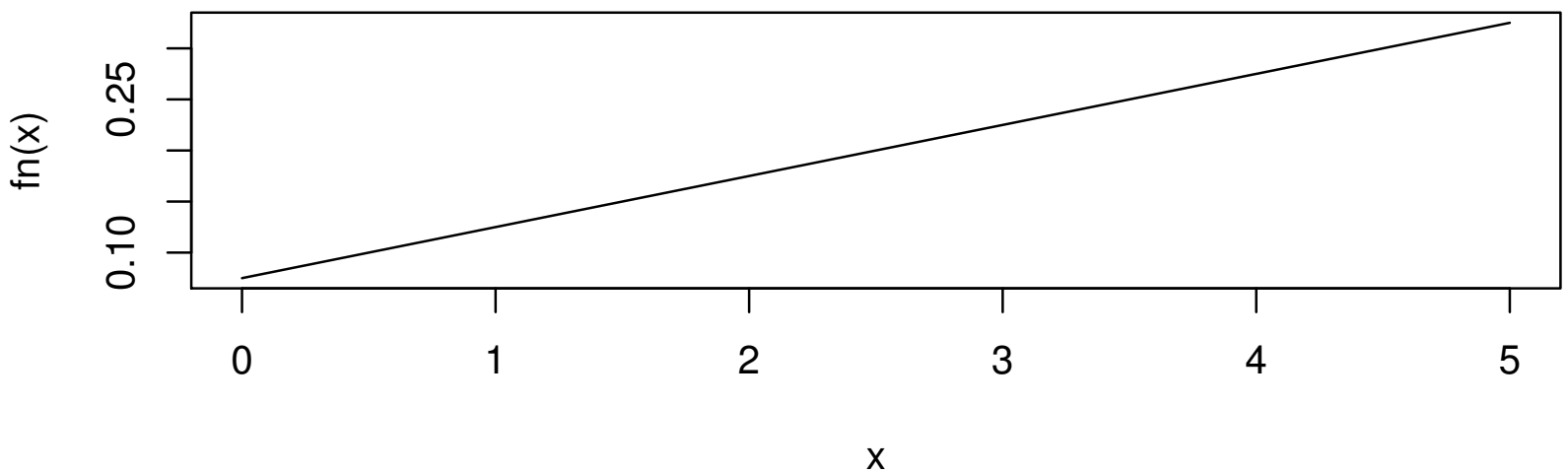
Inverse sampling

This method works when we can know the closed form of the pdf we want to simulate from and can derive the inverse of that function.
Steps:

1. Sample one number u from $Unif(0,1)$. Let $u = F(z) = \int_L^z f(x)\,dx$ (here, L is the lower bound of the pdf f).
2. Then $z = F^{-1}(u)$ is a draw from $f(x)$.

Inverse sampling

Example: Let $f(x) = \frac{1}{40}(2x+3)$, with $0 < x < 5$.



Inverse sampling

1. Let $f(x) = \frac{1}{40}(2x+3)$, with $0 < x < 5$.
2. We have to draw a number from the uniform distribution and then solve for z, which amounts to finding the inverse function:

$$u = \int_0^z \frac{1}{40}(2x+3)\,dx \tag{2}$$

3. Solving the integral:

$$\begin{aligned}\int_0^z \frac{1}{40}(2x+3) &= \frac{1}{40}\int_0^z (2x+3)\,dx \\ &= \frac{1}{40}\left[\frac{2x^2}{2}+3x\right]_0^z = \frac{1}{40}[z^2+3z]\end{aligned} \tag{3}$$

Inverse sampling

So, if $u = \frac{1}{40}[z^2 + 3z]$, then, $40u = z^2 + 3z$.
Completing the square, we can express z in terms of u:
Notice that

$$(z + \frac{3}{2})^2 = z^2 + 3z + \frac{9}{4} \tag{4}$$

Therefore

$$40u + \frac{9}{4} = (z + \frac{3}{2})^2 \tag{5}$$

Solving for z we get:

$$z = \frac{1}{2}(\sqrt{160u + 9} - 3) \tag{6}$$

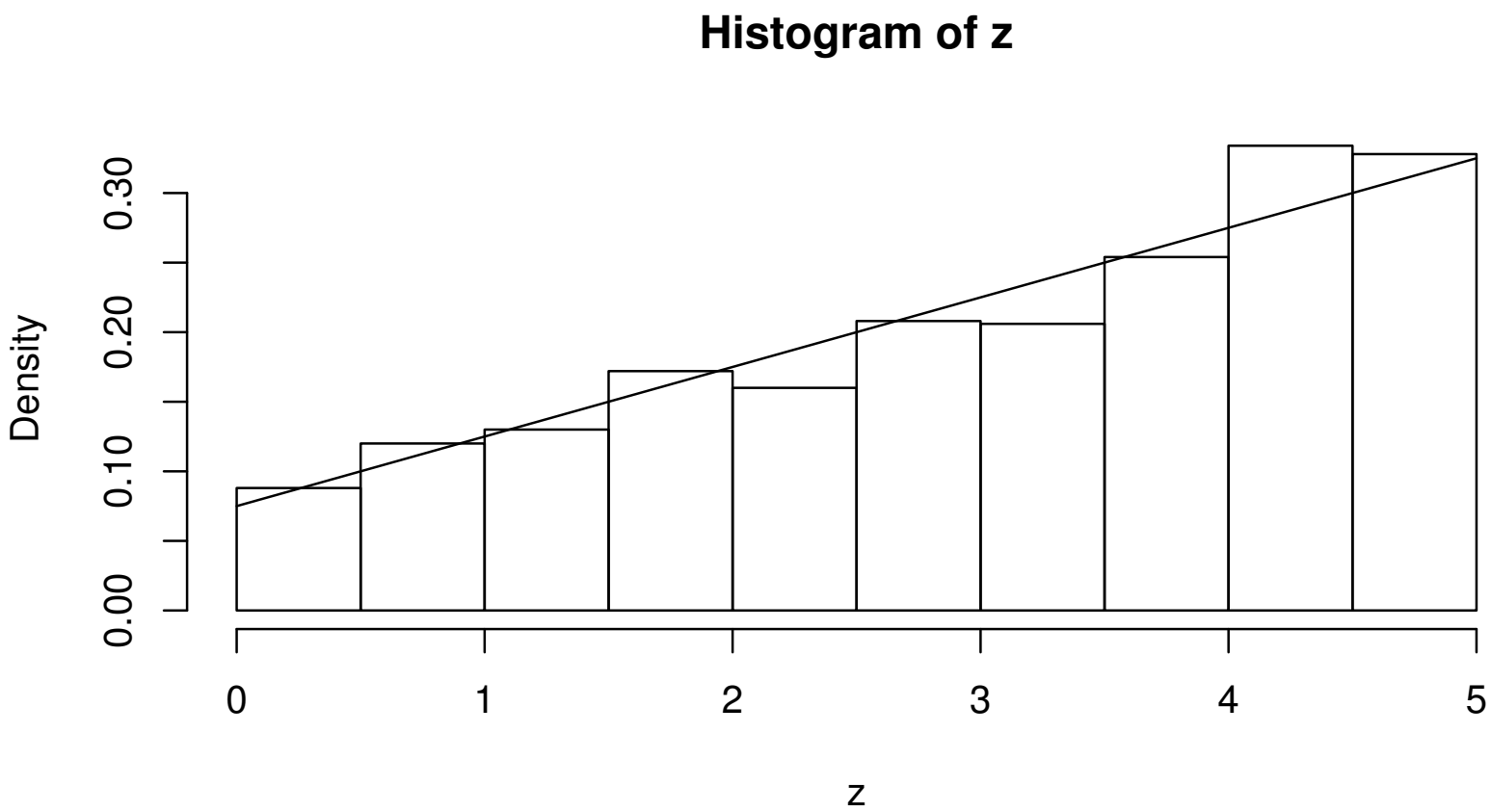
Note: you will never need to do this yourself, this is just intended to give you a sense of how sampling works.

Inverse sampling

```
u<-runif(1000,min=0,max=1)

z<-(1/2) * (sqrt(160*u +9)-3)
```

This method can't be used if we can't find the inverse, and it can't be used with multivariate distributions.

Inverse sampling



Rejection sampling

If $F^{-1}(u)$ can't be computed, we sample from $f(x)$ as follows:

1. Sample a value z from a distribution $g(z)$ from which sampling is easy, and for which

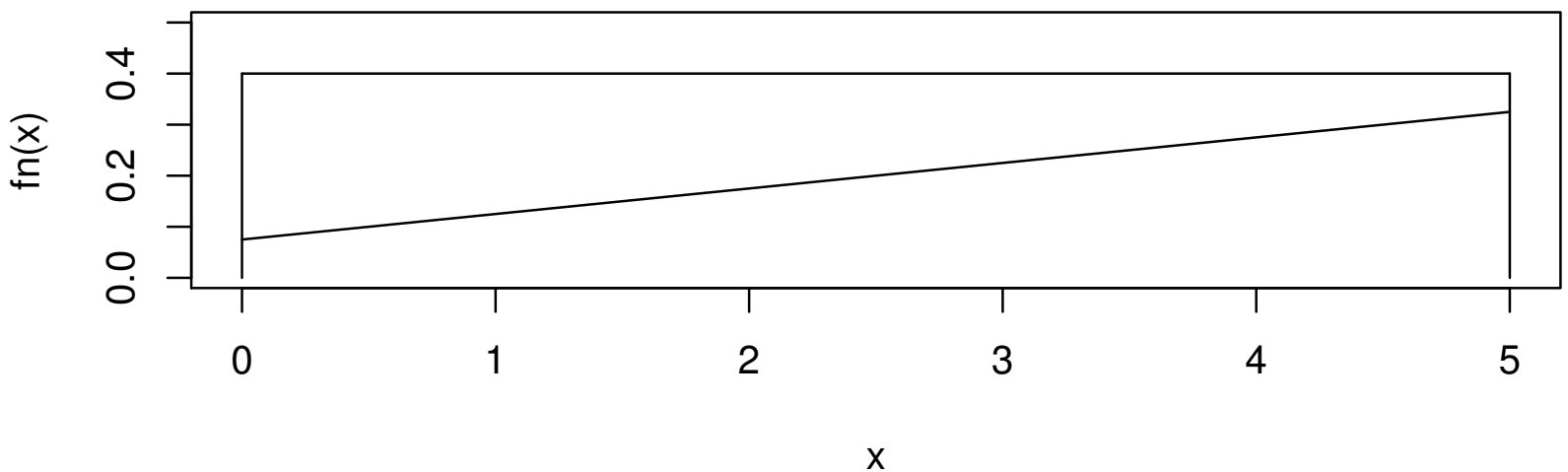
$$mg(z) > f(z) \quad m \text{ a constant} \tag{7}$$

   $mg(z)$ is called an envelope function because it envelops $f(z)$.
2. Sample $u \sim Unif(0,1)$. Compute mg(x)*u: this is the maximum value possible of the function mg(x).
3. If $mg(x)u < f(x)$, then z is treated as a draw from $f(x)$. Otherwise return to step 1.
   Note that another way to say this is:
   If $u < \frac{f(x)}{mg(x)}$ then accept.

Rejection sampling

1. For example, consider f(x) as above: $f(x) = \frac{1}{40}(2x+3)$, with $0 < x < 5$. The maximum height of $f(x)$ is 0.325.
2. So we need an envelope function that exceeds 0.325. The uniform density $Unif(0,5)$ has maximum height 0.2, so if we multiply it by 2 we have maximum height 0.4, which is greater than 0.325.

Rejection sampling



Rejection sampling

In the first step, we sample a number x from a uniform distribution Unif(0,5). This serves to locate a point on the x-axis between 0 and 5 (the domain of x).

```
x<-runif(1,0,5)
```

Keep in mind that even though you can't sample from the function f(x), you can evaluate it for any x:

```
fn(x)

## [1] 0.2222737
```

Rejection sampling

The next step involves locating a point in the y direction once the x coordinate is fixed. If we draw a number u from Unif(0,1):

```
u<-runif(1,0,1)
```

then $mg(x)u = 2 * 0.2u$ is a number between 0 and $2 \times 0.2 = 0.4$:

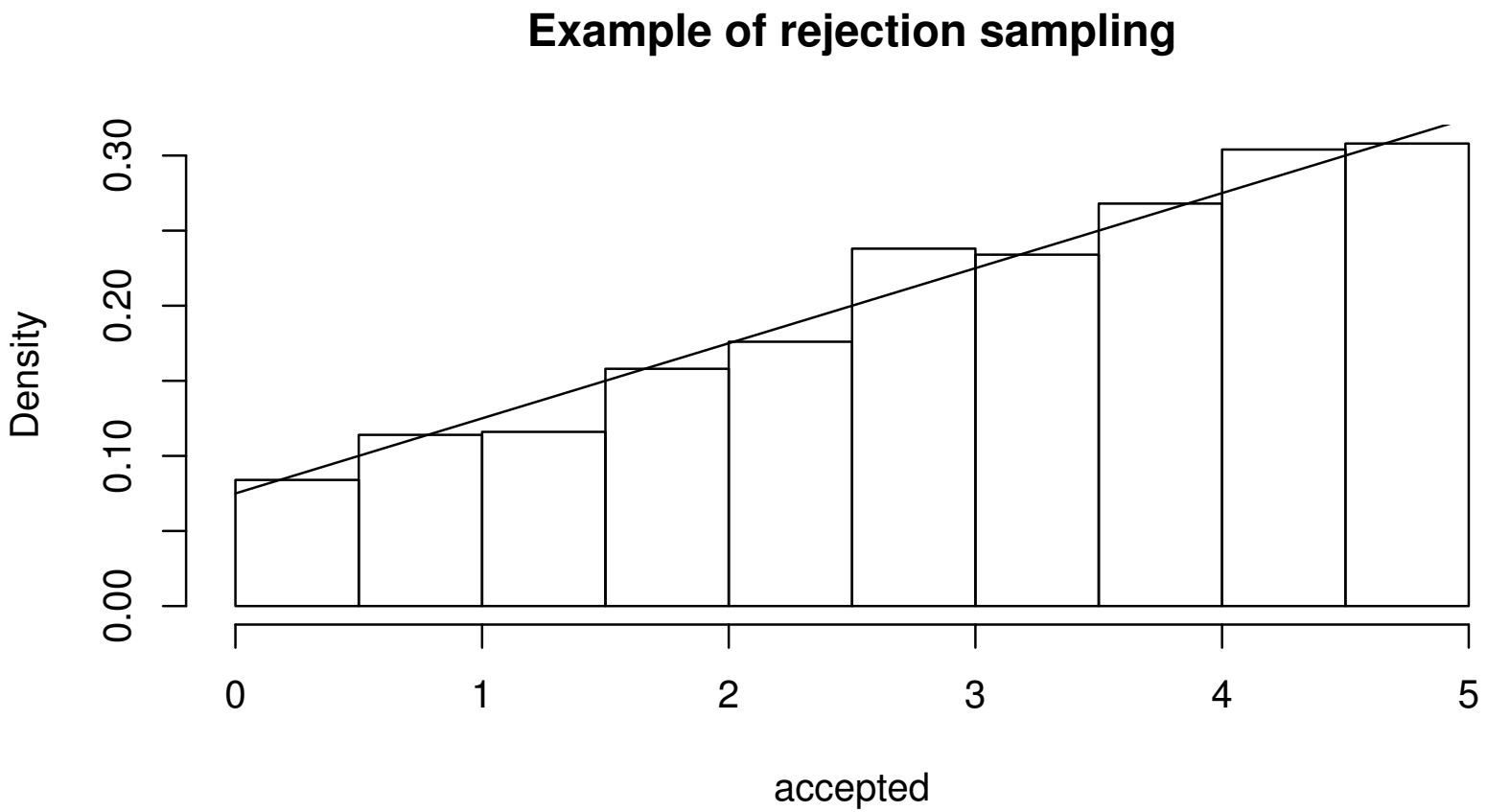
```
0.4*u

## [1] 0.2871135

0.4*u < fn(x)

## [1] FALSE
```

If this number is less than f(x), that means that the y value falls within f(x), so we accept it, else reject.

Rejection sampling

```
count<-0
k<-1
accepted<-rep(NA,1000)
rejected<-rep(NA,1000)
while(k<1001)
{
z<-runif(1,min=0,max=5)
r<-((1/40)*(2*z+3))/(2*.2)
if(r>runif(1,min=0,max=1)) {
  accepted[k]<-z
  k<-k+1} else {
    rejected[k]<-z
  }
count<-count+1
}
```

Rejection sampling



Rejection sampling can be used with multivariate distributions.

Some limitations of rejection sampling

1. Finding an envelope function may be difficult.
2. The acceptance rate would be low if the constant m is set too high and/or if the envelope function is too high relative to f(x), making the algorithm inefficient.

Summary so far

1. We now know what a Markov Chain is.
2. We know that there are several methods to sample from a distribution (we saw two).

Next, I will talk about a sampling method using in JAGS (Just Another **Gibbs Sampler**).

Gibbs Sampling

Goal: sample from a posterior distribution of some parameter(s). Let Θ be the vector of parameter values, say $\Theta =< \theta_1, \theta_2 >$. Let j index the j-th iteration.

Algorithm:

1. Assign starting values to $\Theta =< \theta_1, \theta_2 >$: $\Theta^{j=0} \leftarrow S$
2. Set $j \leftarrow j + 1$
3. 1. Sample $\theta_1^j \mid \theta_2^{j-1}$ (e.g., using inversion sampling).
   2. Sample $\theta_2^j \mid \theta_1^j$ (e.g., using inversion sampling).
4. Return to step 1.

Simplifying a bit, eventually, we will start sampling the parameter values from the correct distribution—we will get a sample from the posterior distribution.

Of course, I didn't explain here *why* this works. See Lynch book for details.

Gibbs sampling: Example

Sample two random variables from a bivariate normal, where $X \sim N(0,1)$ and $Y \sim N(0,1)$.

Gibbs sampling: Example

See MC walk graphic.

Gibbs sampling: Example

We can plot the "trace plot" of each density, as well as the marginal density: see traceplot graphic.

Gibbs sampling: Example

The trace plots show the points that were sampled. The initial values were not realistic, and it could be that the first few iterations do not really yield representative samples. We discard these, and the initial period is called “burn-in” (JAGS) or “warm-up” (Stan). The two dimensional trace plot traces the Markov chain walk (see Markov chain walk figure)

Gibbs sampling: Example

We can also summarize the marginal distributions. One way is to compute the 95% credible interval, and the mean or median. That's what we have been doing in the last few examples we saw.

Key point regarding MCMC sampling

1. As long as we know the posterior distribution up to proportionality, we can sample from the posterior: Posterior $\propto$ Lik Prior
2. There are other sampling methods, such as Metropolis-Hastings, and Hamiltonian MC (Stan uses this; see Stan manual).
3. As end-users, we do not need to be able to program these ourselves (except for fun).

Recall the JAGS commands

```
> headnoun.mod <- jags.model(
   file="gwmaximal.jag",
   data = headnoun.dat,
   n.chains = 4,
   n.adapt =2000 , quiet=T)
```

1. `n.chains`: Number of independent chains to run (always have more than 1, I use 4 usually)
2. `a.adapt`: the initial period (to be ignored).

Recall the JAGS commands

```
headnoun.res <- coda.samples(headnoun.mod,
                             var = track.variables,
                             n.iter = 10000,
                             thin = 1)
```

1. `n.iter`: Number of iterations (walks) in an MCMC chain.
2. `thin`: Reduce number of MCMC sequences from posterior (helps in plotting less dense plots). thin=1 means keep all of them. thin=2 means take every alternative one, etc.

Evaluating the model

The first thing we want to do is find out whether our model has converged: whether the MCMC sampler is sampling from the posterior distribution.
The function gelman.diag will give you this information, but the traceplots are also informative:

```
gelman.diag()
```

Evaluating model fit

Recall Gibson and Wu data.
We can examine quality of fit by generating **predicted** reading times, and comparing them with

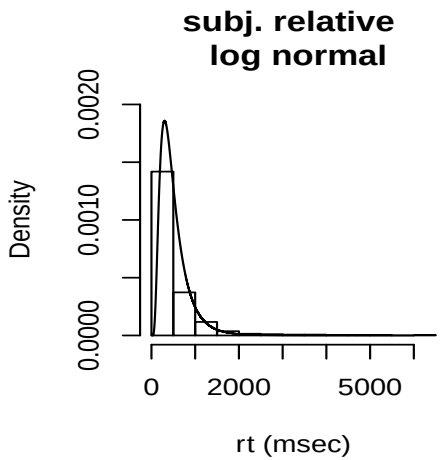
1. the actual data
2. new data (much better)

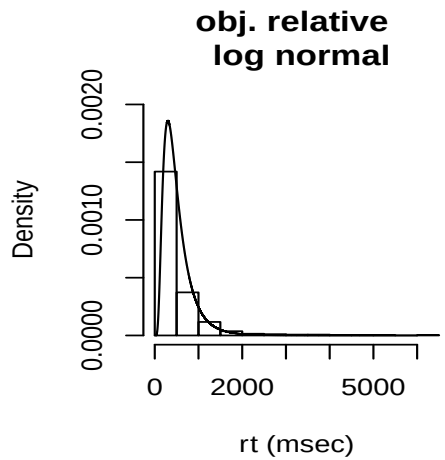
An example of posterior predictive checks (using Stan) is given in: http://www.ling.uni-potsdam.de/∼vasishth/statistics/BayesLMMs.html

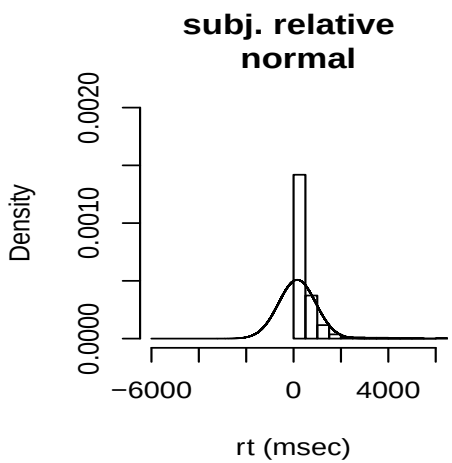
Evaluating model fit

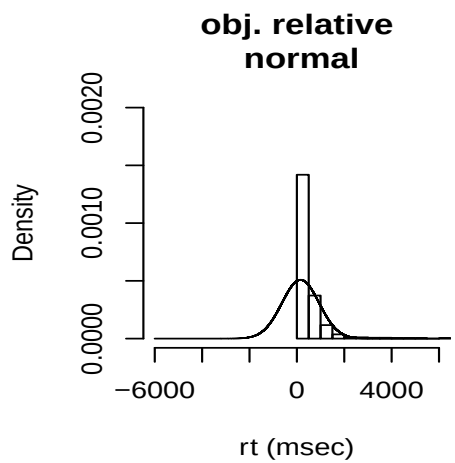
Using the log normal to model the Gibson and Wu data

The data clearly seem to be log-normally distributed:









Evaluating model fit

Using the log normal to model the Gibson and Wu data

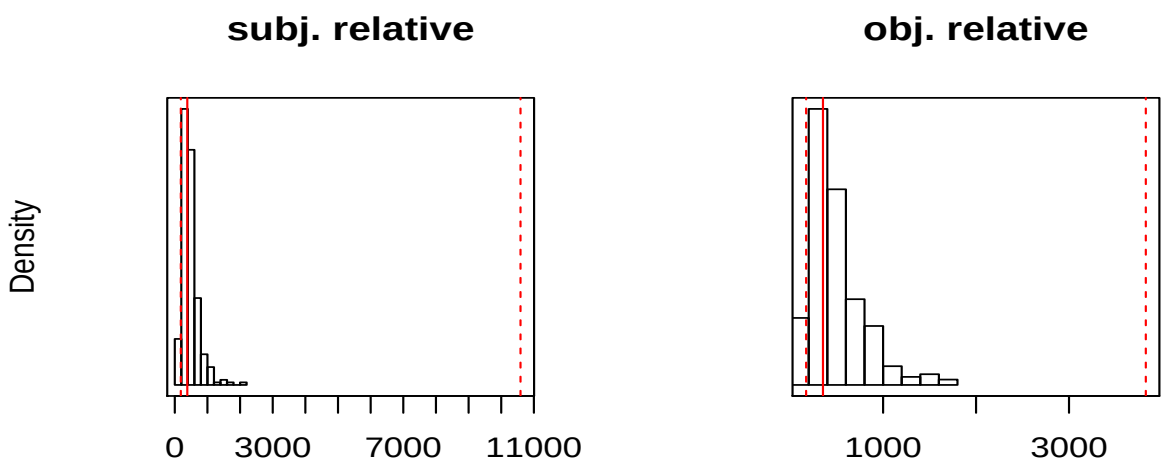
In Stan we simply need to write:

```
rt ~ lognormal(mu,sigma)
```

Evaluating model fit

Using new data from Vasishth et al 2013

These extreme values turn up again in the subject relative condition. Either we could fit a mixture of models to the data, or we transform the data to stabilize variance.



Checking model assumptions

```
## Loading required package:  Matrix
```

The best model in the Gibson and Wu data-set is:

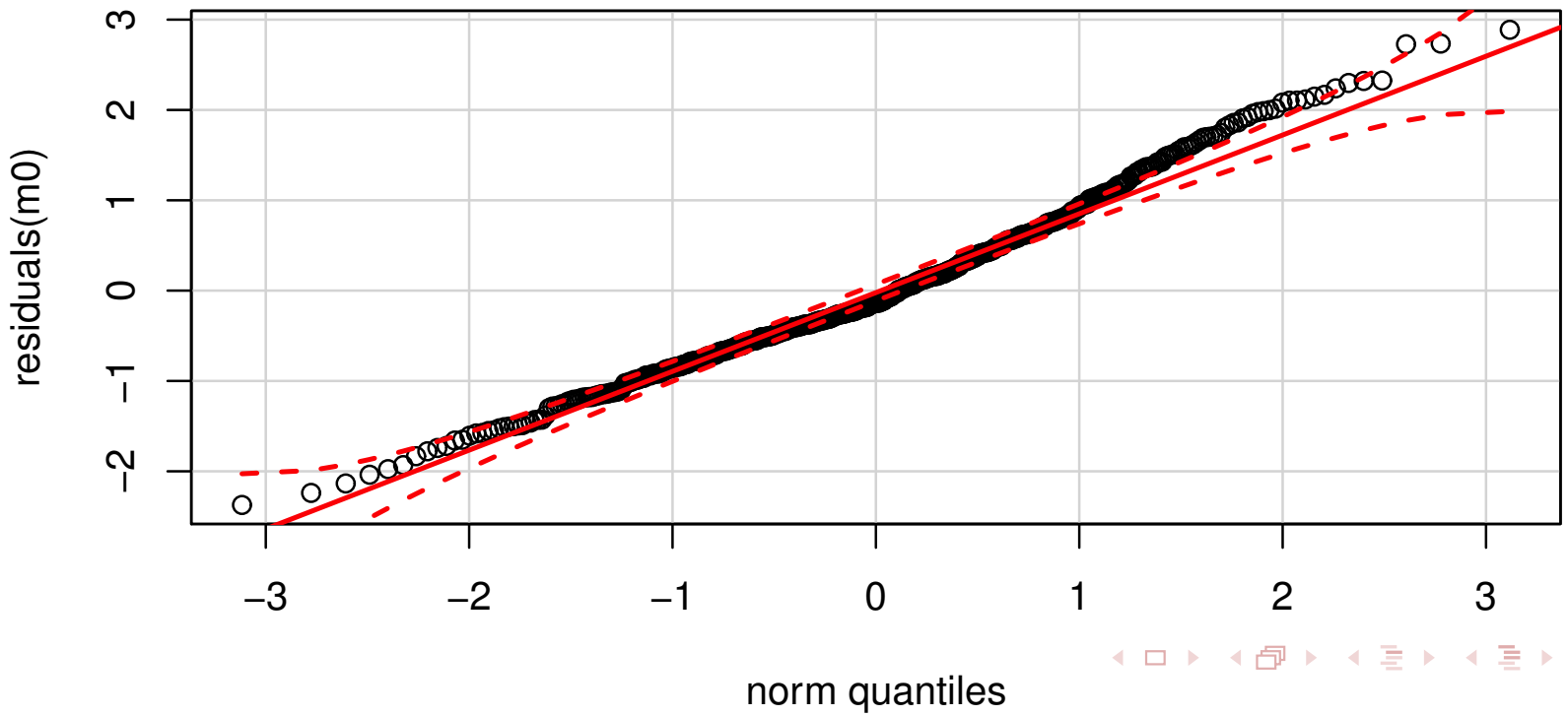
```
m0<-lmer(rrt~x+(1|subj)+(1|item),headnoun)
### varying slopes for subj
m1<-lmer(rrt~x+(1+x|subj)+(1|item),headnoun)
##
m1a<-lmer(rrt~x+(x||subj)+(1|item),headnoun)
### varying intercepts and slopes for subj and item
m2<-lmer(rrt~x+(1+x|subj)+(1+x|item),headnoun)
m2a<-lmer(rrt~x+(x||subj)+(x||item),headnoun)
```

Checking model assumptions

```
anova(m0,m1,m1a,m2,m2a)


## refitting model(s) with ML (instead of REML)

## Data: headnoun
## Models:
## m0: rrt ~ x + (1 | subj) + (1 | item)
## m1a: rrt ~ x + ((1 | subj) + (0 + x | subj)) + (1 | item)
## m1: rrt ~ x + (1 + x | subj) + (1 | item)
## m2a: rrt ~ x + ((1 | subj) + (0 + x | subj)) + ((1 | item) + (0 +
## m2a:     x | item))
## m2: rrt ~ x + (1 + x | subj) + (1 + x | item)
##     Df    AIC    BIC  logLik deviance  Chisq Chi Df Pr(>Chisq)
## m0   5 1603.5 1625.0 -796.76   1593.5
## m1a  6 1605.0 1630.8 -796.51   1593.0 0.5005      1     0.4793
## m1   7 1605.6 1635.7 -795.78   1591.6 1.4504      1     0.2285
## m2a  7 1607.0 1637.2 -796.51   1593.0 0.0000      0     1.0000
## m2   9 1608.5 1647.3 -795.27   1590.5 2.4737      2     0.2903
```

Checking model assumptions

```
library(car)
qqPlot(residuals(m0))
```



A bad consequence of ignoring model assumptions

The published result in Gibson and Wu is driven by 8 out of 547 extreme values $> 3000ms$:

```
Fixed effects:
             Estimate Std. Error t value
(Intercept)   548.43       51.56  10.638
x            -120.39       48.01  -2.508
```

Normalizing the residuals leads to no effect at all:

```
             Estimate Std. Error t value
(Intercept) -2.67217    0.13758 -19.423
x           -0.07494    0.08172  -0.917
```

Removing the 8/547 extreme values in raw RT:

```
             Estimate Std. Error t value
(Intercept)   494.59       34.09  14.509
x             -12.66       29.94  -0.423
```

Gibson and Wu: the published result

```
head.means<-aggregate(rt~subj+type,
                      mean,data=headnoun)
```

Now, after aggregation, we can run a paired t-test:

```
t_test_result<-t.test(subset(head.means,type=="subj-ext")$r
       subset(head.means,type=="obj-ext")$rt,paired=T)
t_test_result$statistic;t_test_result$p.value

##        t
## 2.63007
## [1] 0.01248104
```

The difference was significant. (They did an ANOVA, but it's the same thing, as $t^2 = F$.)

The lesson in this example

The lesson here is to

1. first examine your data graphically
2. check whether model assumptions are satisfied

Fitting models is not primarily about checking for statistical significance, we are also

1. defining how we think the data were generated
2. generating predictions about what future data will look like.

Non-normality of residuals can lead to loss of power

I showed an example where extreme values (non-normality) lead to an over-enthusiastic p-value.

Here is another consequence:

```
## number of simulations:
nsim<-100
## sample size:
n<-100
## predictor:
pred<-rep(c(0,1),each=n/2)
## matrices for storing results:
store<-matrix(0,nsim,4)
storenorm<-matrix(0,nsim,4)
## true effect:
beta.1<-0.5
```

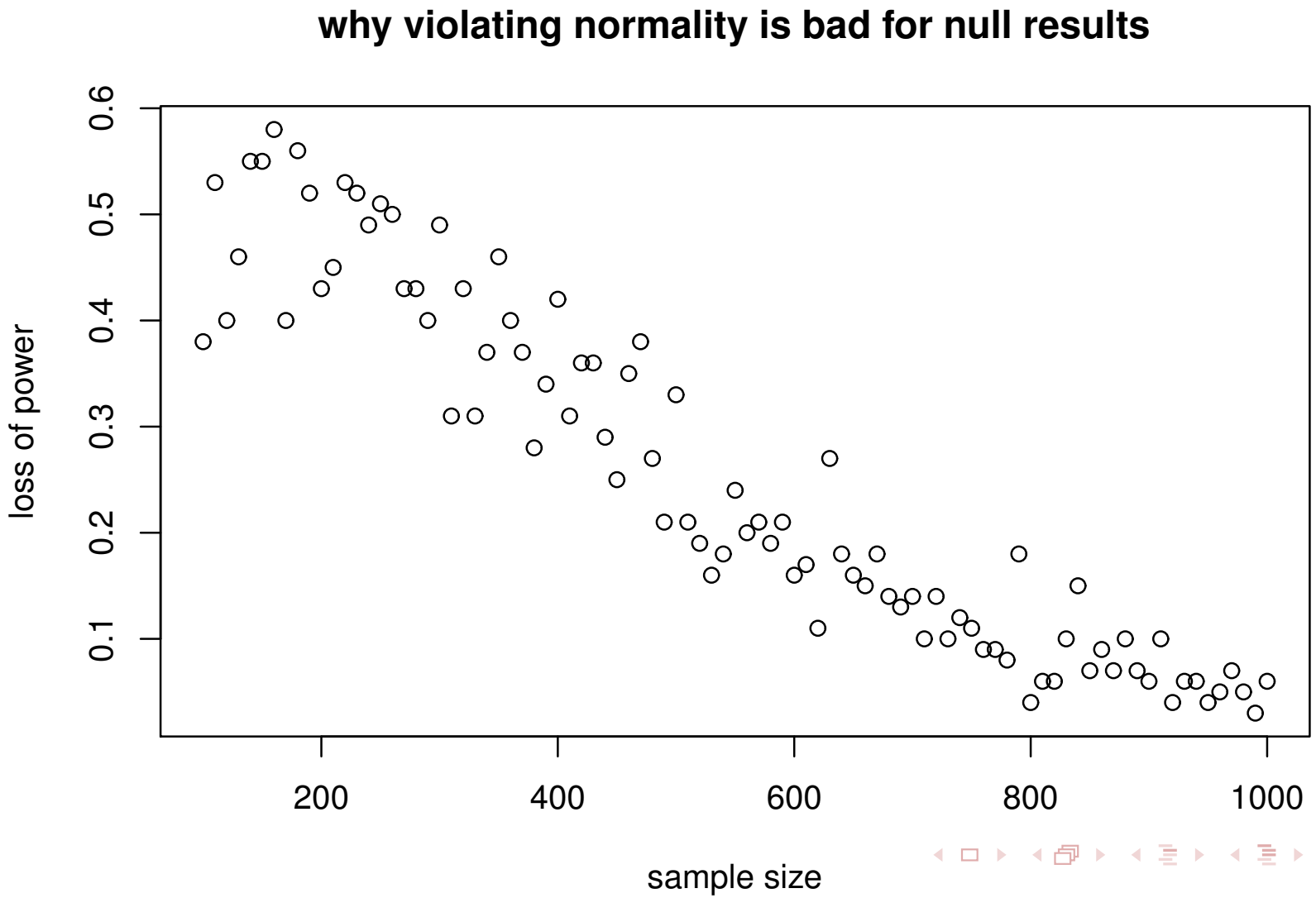
Non-normality of residuals can lead to loss of power

```
for(i in 1:nsim){
errors<-rlnorm(n)
errorsnorm<-rnorm(n)
y<-5 + beta.1*pred + errors ## generate data:
ynorm<-5 + beta.1*pred + errorsnorm
fm<-lm(y~pred)
fmnorm<-lm(ynorm~pred)
## store coef., SE, t-value, p-value:
store[i,1:4] <- summary(fm)$coef[2,1:4]
storenorm[i,1:4] <- summary(fmnorm)$coef[2,1:4]
}
```

Non-normality of residuals can lead to loss of power

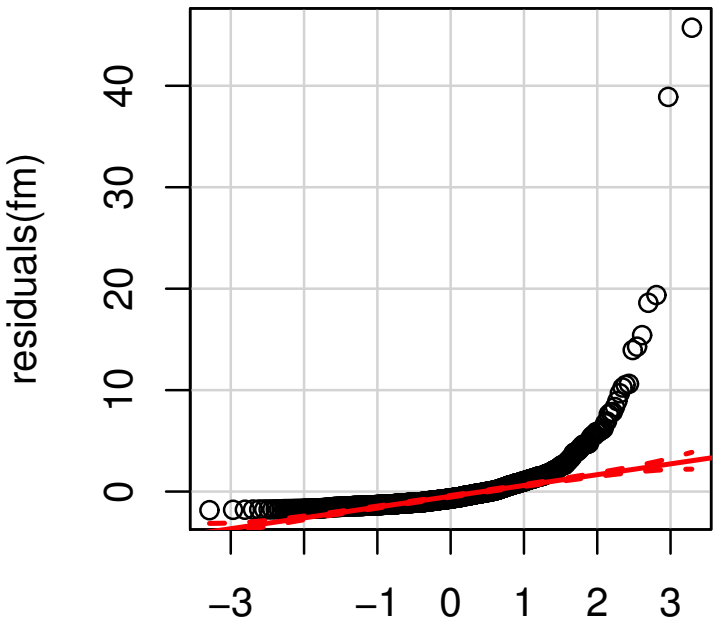
```
## ``observed'' power for raw scores:
mean(store[,4]<0.05)

## [1] 0.34

mean(storenorm[,4]<0.05)

## [1] 0.67
```

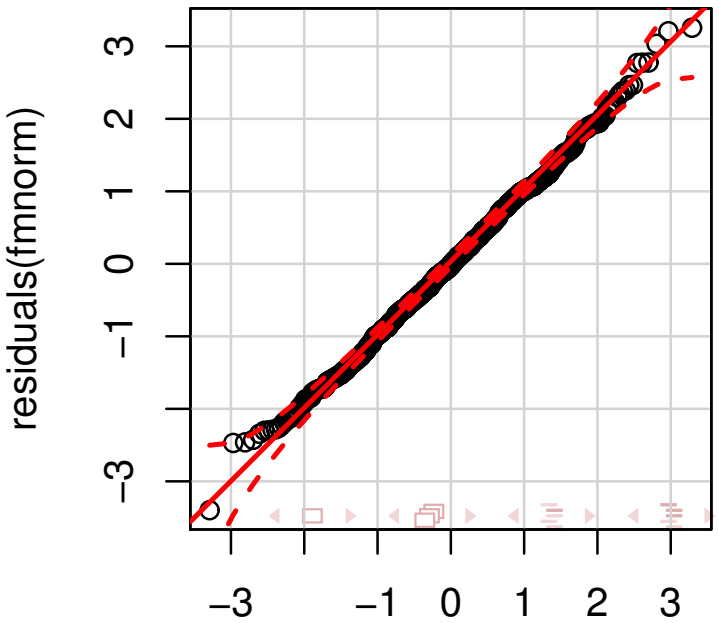
The cost of non-normality: loss of power



Non-normality of residuals can lead to loss of power

```
library(car)
op<-par(mfrow=c(1,2),pty="s")
qqPlot(residuals(fm))
qqPlot(residuals(fmnorm))
```

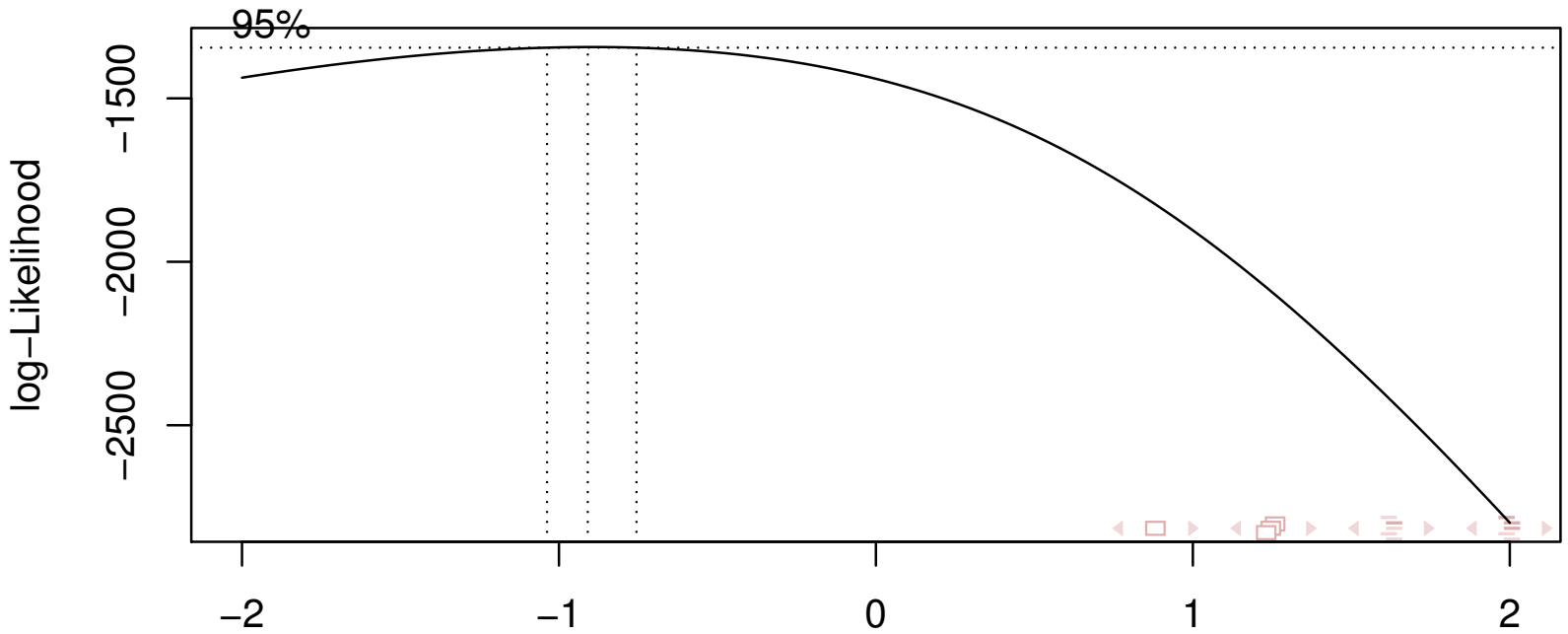




What transform to use?

Box-Cox method to find the appropriate variance stabilizing transform:

```
library(MASS)
boxcox(rt~type*subj,data=headnoun)
```



What transform to use?

1. A λ of -1 implies a reciprocal transform.
2. A λ of 0 implies a log transform.

We generally use the reciprocal or log transform for reading time data.
See Venables and Ripley 2002 for details (or Box and Cox, 1964), or read the Linear Modeling lecture notes.

Concluding remarks

1. We now have some idea about how we can sample from a posterior distribution.
2. We also have some idea of how to check for convergence, evaluate model assumptions using posterior predictive checks.

In the next lecture, we will fit some more complex linear mixed models.