PM520: Lecture 6

Metropolis-Hastings Markov chain Monte Carlo [MCMC]

Lab Task #4 - The Polya Urn

- Code up this version of the Urn model. Then try the following:
- Start with one 'red' ball and the black ball (but code the colors as numbers, for simplicity's sake, so Ball1<-1, Ball2<-2, say).
- Repeat until you have 51 non-black balls, but then throw away the last ball you added (we will discuss why we do this later); replicate this process multiple times.

1. What is the distribution of the number of (non-black) colors at the end?
2. What is the distribution of the number of balls of the commonest color at the end?
3. How many replicates do you need to do to answer these questions? How did you decide upon this number?
4. Repeat 1. and 2., but start with two balls of a same color+black ball each time. Compare your results to those from 1. and 2. Can you explain what you see?

Lab Task 5 - The Generalized Polya Urn (note: you will need this version later in the course, so it better work!)

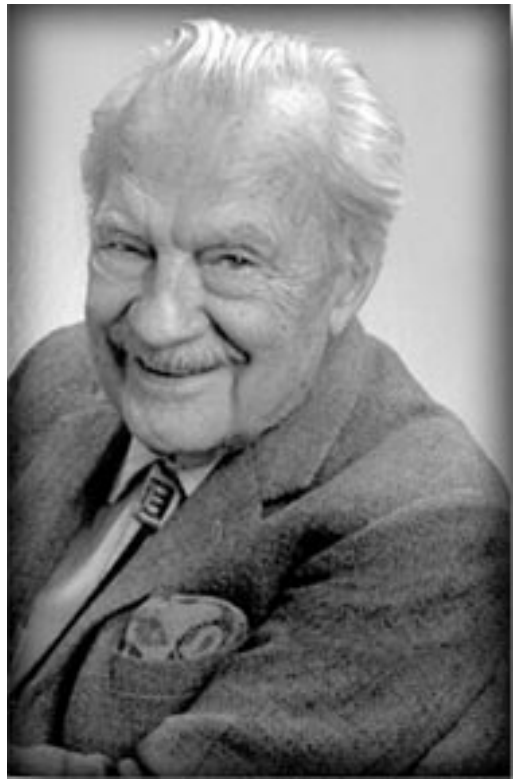
- Code up a further generalized version of the Polya Urn model
  - Imagine that each color has a 'weight'
  - When we draw a ball, a given ball is chosen with probability equal to (Weight of that ball)/(total weight of all balls in the Urn).
  - [So, our previous version had implicit weights of 1 for each color]
- Start with two 'red' balls and the black ball (but, again, code the colors as numbers, for simplicity's sake).
- Repeat until you have 50 non-black balls; replicate this process multiple times.
- Assume all, non-black colors have weight 1
  1. What is the expected number of different (non-black) colors in the Urn at the end as a function of the weight of the black ball ?
  2. What is the distribution of the number of (non-black) colors at the end, as a function of the weight of the black ball? (Try weight=1, or 2, or, ... , or 10).

Results

- Expected # colors as a function of weight of black ball:

Weight= 1 Mean number of colors in urn at end: 3.997
Weight= 2 Mean number of colors in urn at end: 6.4264
Weight= 3 Mean number of colors in urn at end: 8.4478
Weight= 4 Mean number of colors in urn at end: 10.2922
Weight= 5 Mean number of colors in urn at end: 11.8176

Metropolis algorithm (Metropolis, 1953)



Born: June 11, 1915
Chicago

- Define an 'energy' for a configuration of a system. Assume we are **minimizing** energy (denoted by f(x)).
- Propose moves from an old state x, to a new state x' according to a **symmetric** proposal kernel q() [So q(x->x')=q(x'->x) for all x, x'].
- Let ΔE denote the change in energy between the new and old state. i.e., we define ΔE(x,x')=f(x')-f(x).
- Move from x to x' with prob. min{1,exp(-ΔE(x,x')/T)}, where T is a 'temperature' [FIXED]. *(This is for a minimizing algorithm, if you are maximizing use h=min{1,exp(ΔE(x,x')/T)}.)*
-

If you are minimizing:
when energy decreases, you move with prob. 1
when energy increases, you move with prob. <1.

Simulated annealing

- Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. "Optimization by Simulated Annealing." Science 220, 671-680, 1983.
- Similar in spirit to the Metropolis algorithm.
- Again, uses a rule q for proposing moves to a new state x' based on the current state x. Assume we are trying to find the **minimum** of f.
- Define $\Delta E(x,x')=f(x')-f(x)$.
- Move to new state with prob. $h=\min\{1,\exp(-\Delta E(x,x')/T)\}$, where T is a 'temperature'. *(This is for a minimizing algorithm, if you are maximizing use $h=\min\{1,\exp(\Delta E(x,x')/T)\}$)*
- e.g. T=1:
  - $f(x')-f(x)=1 \rightarrow h=1/e < 1$
  - $f(x')-f(x)=-1 \rightarrow h=e > 1$
- **T is always positive, but decreases over time** (What starting value? At what rate should it decrease?) so moves in the wrong direction become less likely.

SA or Metropolis task #1

- Code up either the SA or Metropolis algorithm. Test it on a simple function, such as $f(x,y)= x^2+y^2$
- Then test it on the Rosenbrock function (see next slide)

SA or Metropolis task #2: regression

Regression fit aims to minimize sum of squared residuals:

$\mathbf{x}=\{x_1,x_2,x_3,\ldots,x_n\}$

$\mathbf{y}=\{y_1,y_2,y_3,\ldots,y_n\}$

a=intercept; b=slope; so y=ax+b

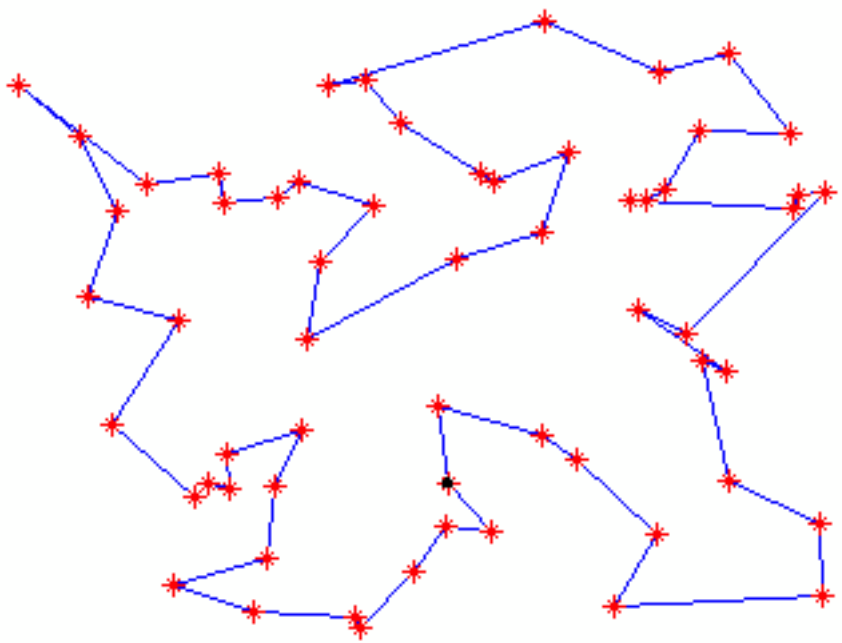
$SS(a,b)=\sum_i$ (Observed y_i - Fitted y)^2

So, search through space of values for (a,b) (i.e., Intercept, slope), to find minimal (a,b) that give minimal SS(a,b).

Work in small groups if you wish to.

Traveling salesman problem

- Given a set of cities, with known distances between them, what is the shortest round-trip route that visits all cities?



Sudoku

	7		1		5			
5						3		
		6	7	9		1	2	5
1	5	2	8		6		9	
				3			5	
3				5			8	
8	6			7		5	1	
				6				
		9			2		4	

Metropolis-Hastings Markov chain Monte Carlo [MCMC]

Markov Chains

- Z(t) is the state of the system at time t [t is discrete]
- The state-space, S, is the set of values that Z(t) can take.
- The transition function, $P(z_1,z_2)$, [or $P(z_2 \mid z_1)$] gives the probability that the next value of Z is z_2, given that Z is currently in state z_1.
- The **Markov condition**: $P(Z(t)=z(t) \mid Z(t-1)=z(t-1),Z(t-2)=z(t-2), \ldots,Z(0)=z(0)) = P(Z(t)=z(t) \mid Z(t-1)=z(t-1))$.
  - "Where you go depends upon where you are, but not how you got there."
  - "The future is independent of the past, except through the present state".
- Example 1:
  - X(t) = roll a six-sided die
  - Define Z(t) = [X(t)+X(t-1)+…+X(1)]
  - So $P(z_1,z_2)=1/6$ if $1 \le z_2-z_1 \le 6$ (for integers z_1,z_2)

Markov Chains

- Z(t) is the state of the system at time t [t is discrete]
- The state-space, S, is the set of values that Z(t) can take.
- The transition function, $P(z_1,z_2)$, [or $P(z_2 \mid z_1)$] gives the probability that the next value of Z is z_2, given that Z is currently in state z_1.
- The **Markov condition**: $P(Z(t)=z(t) \mid Z(t-1)=z(t-1),Z(t-2)=z(t-2), \ldots,Z(0)=z(0)) = P(Z(t)=z(t) \mid Z(t-1)=z(t-1))$.
  - "Where you go depends upon where you are, but not how you got there."
  - "The future is independent of the past, except through the present state".
- Example 2:
  - X(t) = -1 with prob 1/2; +1 otherwise.
  - Define Z(0) = 0; $Z(t)=Z(t-1)+X(t)$ $[=X(t)+X(t-1)+\ldots+X(0)]$
  - So $P(z_1,z_2)=1/2$ if $|z_2-z_1| =1$ (for integers z_1,z_2)
  - This is a "symmetric random walk".

Stationary distributions

The values it can take

The rule that describes how it moves

- A Markov chain, Z(), with *state-space, S,* and *transition function,* P(), has stationary distribution $\pi()$ if:

$$\sum_{x \in S} \pi(x) P(x \rightarrow y) = \pi(y), \quad \text{for all } y \in S$$

- Intuition: if the state of Z at iteration t is distributed according to $\pi()$, then so is the state of Z at iteration t+1 (and t+2 and t+3 and.....).
- It is the long-term distribution of Z().
- Chain must also be **irreducible** (i.e., it can get to every state from every other state [eventually]).
- The chain must be **aperiodic**. (i.e. no period. Why?)
- Fact: All irreducible, aperiodic Markov chains on finite state-spaces have a stationary distribution.

Detailed balance

- A sufficient, but not necessary condition for $\pi()$ to be the stationary distribution is the *detailed balance* requirement:

$$\pi(x)P(x \to y) = \pi(y)P(y \to x), \quad \text{for all } x, y \in S$$

- Recall previous requirement:

$$\sum_{x \in S} \pi(x)P(x \to y) = \pi(y), \quad \text{for all } y \in S$$

How to sample from a random variable

- Want to sample from a density f(x).
- Suppose we can calculate the cumulative density function F(x).

1. Generate u~Unif[0,1].
2. Set u=F(x).
3. Solve for x.
4. x has density f.

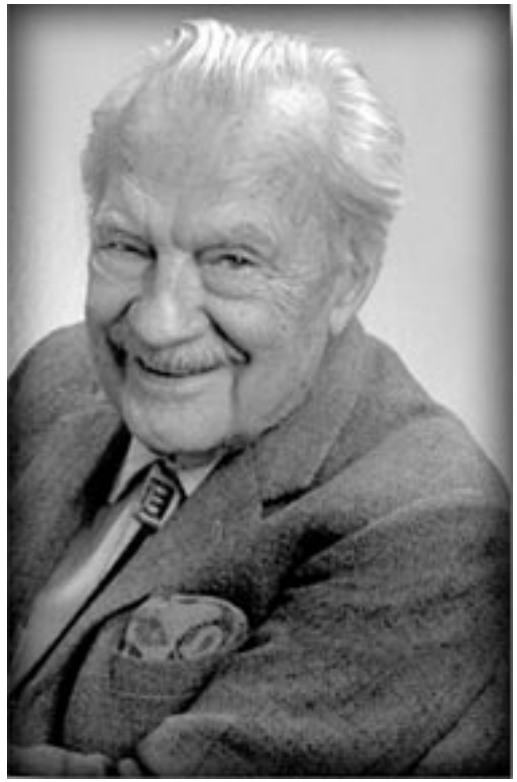
e.g. exponential distribution. $F(x)=1-e^{\lambda x}$, so $u= 1-e^{\lambda x}$, so $x= \log(1-u)/\lambda$.

What if we cannot calculate the CDF F? (e.g. urns, population genetics models,…)

Methods for approximating a probability density f

- Monte Carlo methods:
  - find a way to sample from f,
  - do this many times and use the outcomes as an empirical approximation to f.
  - e.g.,
    - Number of different colored balls in urn model.
- Simple, but not always very efficient.

Recall: Metropolis algorithm for 'optimization'



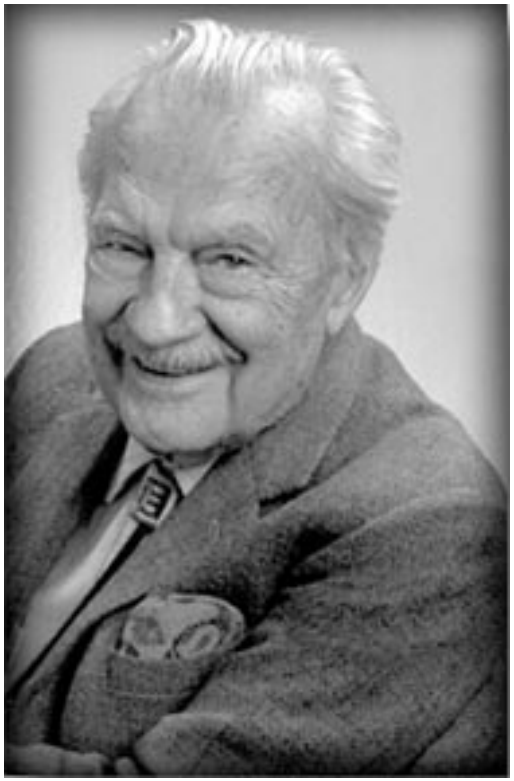
Born: June 11, 1915
Chicago

- Define an 'energy' for a configuration of a system. Assume we are minimizing energy
- Propose moves from an old state x, to a new state x' according to a symmetric proposal kernel q()
  [So q(x->x')=q(x'->x) for all x, x']
- Let $\Delta E=f(x')-f(x)$ denote the change in energy between the new and old state
- Move to new state with prob. $\min\{1,\exp(-\Delta E/T)\}$, where T is a 'temperature' [FIXED]

- In our context, the "energy" is f(x).

Metropolis algorithm for **sampling** from a probability density function f...

- Assume we can calculate density f(x) (but cannot easily draw samples from it).
- Let x be the current state of the system.
- Iterate the following:
  - Propose move from old state x, to new state x' according to a symmetric proposal kernel q() [e.g. symmetric random walk]
  - Calculate h= min{1,f(x')/f(x)}
  - With prob. h, set x=x'; otherwise leave x unchanged.



Born: June 11, 1915
Chicago

- Let X(i) denote the state of this process at iteration i.
- X() is a Markov Chain (sequence of random variables in which the next state depends only upon the last state, not any earlier states).

The **stationary distribution** of X() is f.

X has stationary distribution f

- Showing the following three things will show that the stationary distribution of this version of Metropolis algorithm is f:

1. Detailed balance equation:
    - f(x)P(x->x')=f(x')P(x'->x)
    - So, for us, f(x)q(x->x')P(accept move to x')=f(x')q(x'->x)P(accept move to x).
2. Ergodicity: the chain has period 1. (e.g., it does not alternate between two sets of states).
3. Irreducible: all states can be reached.

- Ergodicity and irreducibility will be a function of how you choose to propose changes to x, (i.e., how you define q). So don't pick something that has a period ≠ 1, or that can't visit all possible values of x.
- But what about detailed balance?

Detailed balance

- Need to show that $f(x)P(x\to x')=f(x')P(x'\to x)$

  i.e., $f(x)q(x\to x')P(\text{accept move } x\to x')=f(x')q(x'\to x)P(\text{accept move } x'\to x)$
- If q is symmetric (e.g. symmetric random walk), then, by definition we will have $q(x\to x')=q(x'\to x)$, so we just need to show that...
- $f(x)P(\text{accept move } x\to x')=f(x')P(\text{accept move to } x'\to x)$, i.e.
- $f(x')/f(x) = P(\text{accept move } x\to x')/P(\text{accept move to } x'\to x)$.
- But
  - $P(\text{accept move } x\to x')= \min\{1, f(x')/f(x)\}$
  - $P(\text{accept move } x'\to x)= \min\{1,f(x)/f(x')\}$
- At least one of $f(x')/f(x)$ and $f(x)/f(x')$ will be ≥1 by construction. Without loss of generality, suppose $f(x)/f(x')\geq 1$. So...
- $f(x')/f(x) = P(\text{accept move } x\to x')/P(\text{accept move to } x'\to x)$.

  $= \min\{1, f(x')/f(x)\}/\min\{1,f(x)/f(x')\}= f(x')/f(x)\}$. □

This is why the probability of accepting a move x->x' is defined in the way it is. If it were defined any other way, we would not get f as the stationary distribution.

Example: Simulating values from Normal dist^{n}.

Pseudocode:

```
MHNormal<-function(Mean,SD,EndPt,NumberOfIts){ # Mean and SD are the Mean and Std. dev. of the Normal,
        #We start somewhere in the range [-EndPt, +EndPt] and run NumberOfIts iterations.
        NormalXs<-mat.or.vec(1,NumberOfIts)  # declare a vector to store the x-values we generate during the course of the
         algorithm
        # you need to start somewhere: sample a random number, x, between +EndPt and -EndPt. Then do the following loop:
        for (i in 1:NumberOfIts){    # each iteration performs one step of the Metropolis process
            # propose a new x-value (xprime) by adding a Unif(-0.5,0.5) (say) random variable to the current value
            # decide whether to move to the new value -
            # Calculate Metropolis term (the ratio of the likelihoods of the two points, i.e., H<-f(x')/f(x))
            H<-min(1,dnorm(xprime,Mean,SD)/dnorm(x,Mean,SD))
            p<-runif(1,0,1) # generate an random number
            if (p<H){
                  x<-x'      # move to x'
            } # we don't need an else, since we just keep the existing x in that case
            # save the current value of x to the vector NormalXs

            # every now and again, plot a histogram of the accepted values
            if (i%%100==0){ # this will pot one every 100 iterations (i%%100 means the remainder when i is divided by 100)
                  hist(NormalXs[0:i],breaks=HistBars)
            }
        }
        return (NormalXs)  # returns the vector of x-values
}
set.seed(4970)   # any number will do here
# Call this function as follows:
MyMH<-MHNormal(0,1,5,50000)
```

On Github in repo:
MetropolisForNormals

(Non-examinable) exercise 1 - part 1

- Code up the Metropolis algorithm for sampling from a Normal distribution with mean 0 and std. dev. 1.
- Use a symmetric proposal kernel that adds a number y to x each time, so that x' = x+y, where y~Unif[-c,+c]
- Run the algorithm:
  - Start it from some value (how are you going to choose that value?)
  - Note that consecutive samples are not independent.
  - You can construct (almost) independent samples by sub-sampling your iterations, sampling every n^{th} iteration.
  - Explore how the choice of n you need to use to get independent sub-samples varies as a function of c above.

Non-examinable exercise 1 - part 2

- Use the code you wrote for the Metropolis algorithm for sampling from a Normal distribution with mean 0 and std. dev. 1.
- Use an **asymmetric** proposal kernel that at each iteration:
  - with prob. 3/4 adds a number y to x e, so that x' = x+y, where y~Unif[0,c]
  - with prob. 1/4 subtracts a number y from x, so x' = x-y, where y~Unif[0,c].
- Run the algorithm for c=0.5 (say),
  - what happens?

For general (non-symmetric) proposal kernels - Detailed balance

- Recall, we need f(x)P(x->x')=f(x')P(x'->x)

- Rewriting, we need:
- f(x')/f(x) = P(move x->x')/P(move to x'->x).

  = P(propose move x->x')P(accept x->x') / P(propose move x'->x)P(accept x'->x))

  = q(x->x') P(accept x->x')/q(x'->x) P(accept x'->x)

  Rearranging, we need:

  [P(accept x->x') / P(accept x'->x)] = [q(x'->x)/q(x->x')][f(x')/f(x)]

Metropolis-Hastings

- We need: [P(accept x->x') / P(accept x'->x)] = [q(x'->x)/q(x->x')][f(x')/f(x)]
- Old acceptance prob: h= min{1,f(x')/f(x)}
- New acceptance probability: use h=min{1, [f(x')q(x'->x)] / [f(x)q(x->x')]} the **Hastings Ratio.**
- Detailed balance is satisfied, so the stationary distribution is f.

Hastings at the U. Toronto chess club

MCMC: Metropolis-Hastings

1. If at *x*, propose move to *x′* according to transition kernel q(*x* -> *x′*).
2. Calculate

$$h = \min\left\{1, \frac{f(x')q(x' \to x)}{f(x)q(x \to x')}\right\}$$

No need to be symmetric - although it often is.

3. Move to *x'* with prob. h, else remain at *x*.
4. Return to 1.

(Metropolis et al. 1953, Hastings 1970)

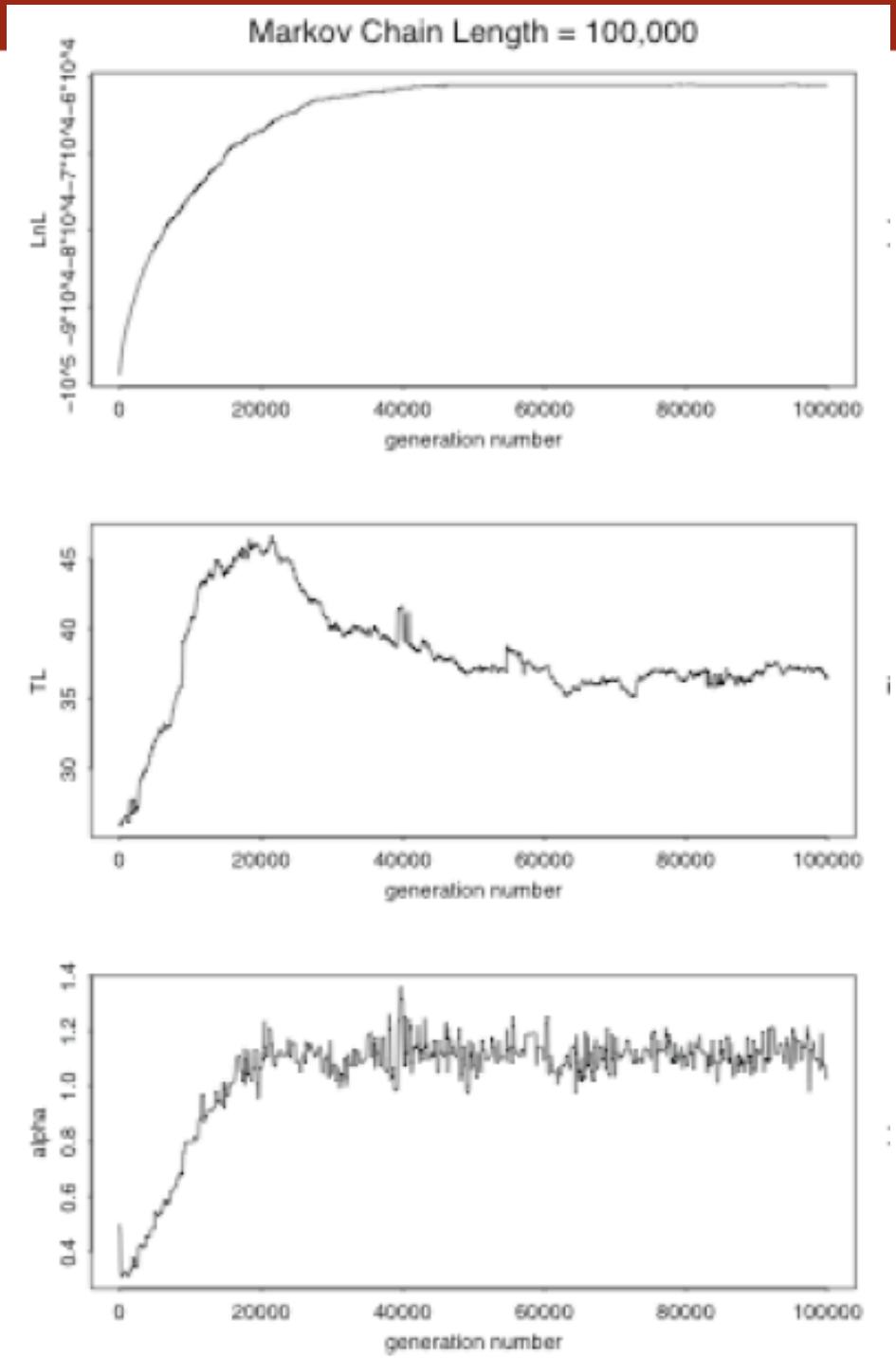
The resulting Markov chain has stationary distribution f.

Non-examinable exercise 1 - part 3

- Use the code you wrote for the Metropolis algorithm for sampling from a Normal distribution with mean 2 and std. dev. 1.
- Use an **asymmetric** proposal kernel that at each iteration:
  - with prob. 3/4 adds a number y to x e, so that x' = x+y, where y~Unif[0,c]
  - with prob. 1/4 subtracts a number y from x, so x' = x-y, where y~Unif[0,c].
- Correct it by using the Hastings Ratio [i.e. adding the term q(x'->x)/ q(x->x')]
- Run the algorithm and show that it works now.

- You are now doing Metropolis-Hastings Markov chain Monte Carlo [MH-MCMC]. This is perhaps the most common form of MCMC.

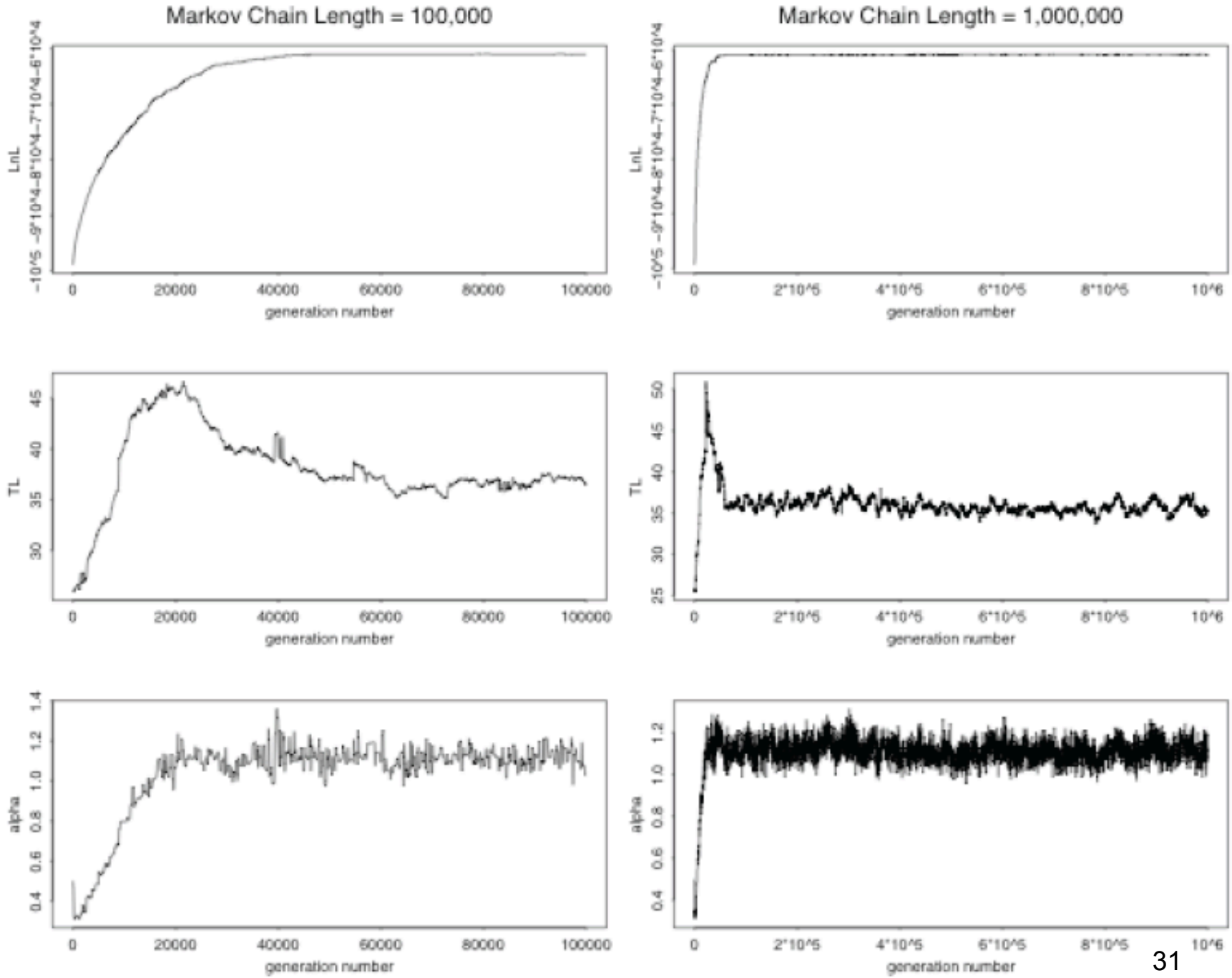
- *Once stationarity is reached* each iteration represents a sample from f.
- Store *every* iteration after 'burn-in' period.
- Consecutive outputs are correlated, so subsample to produce 'independent' samples.
- The art is in choosing a good transition kernel q.
- Aim for an acceptance rate (the proportion of proposals that are accepted) in the region of 20%. [If it is too low, your step size is probably too big and the algorithm won't mix very well; if it is too high, your step-size is probably too small and again the algorithm won't mix well] *But this is just a rule of thumb!*

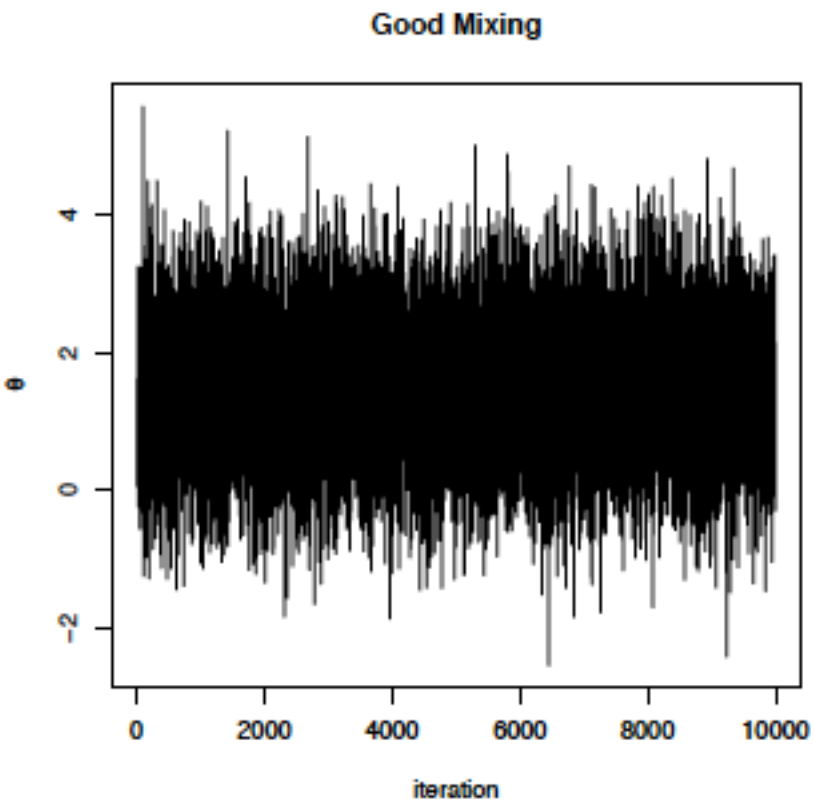


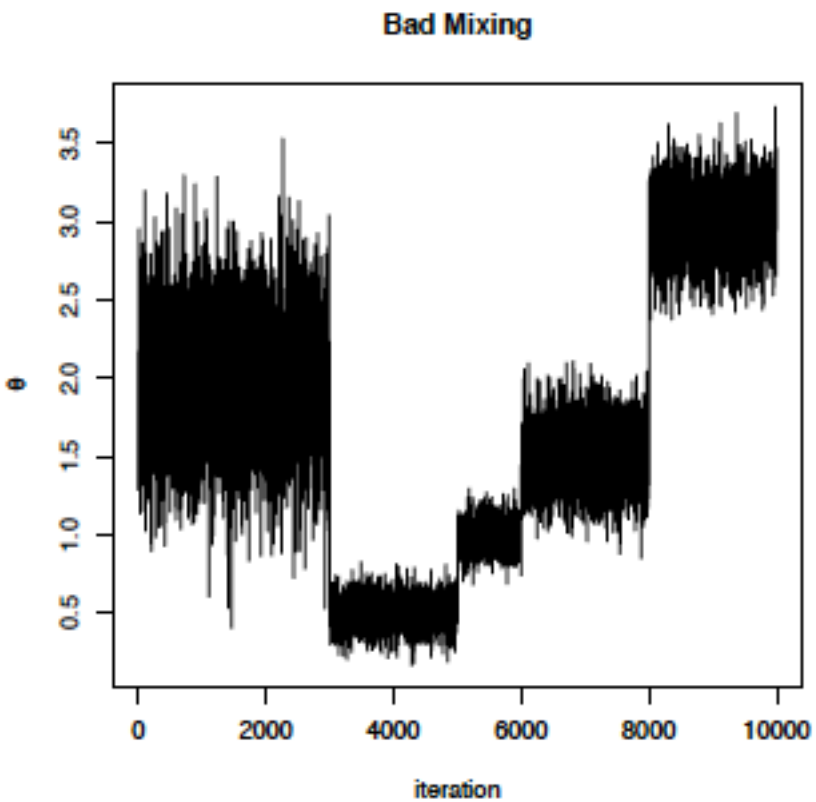
Convergence: "once stationarity has been reached"

- to assess by eye, plot output against iteration number

[MCMC package in R. CODA convergence diagnostics (SPLUS/R).]

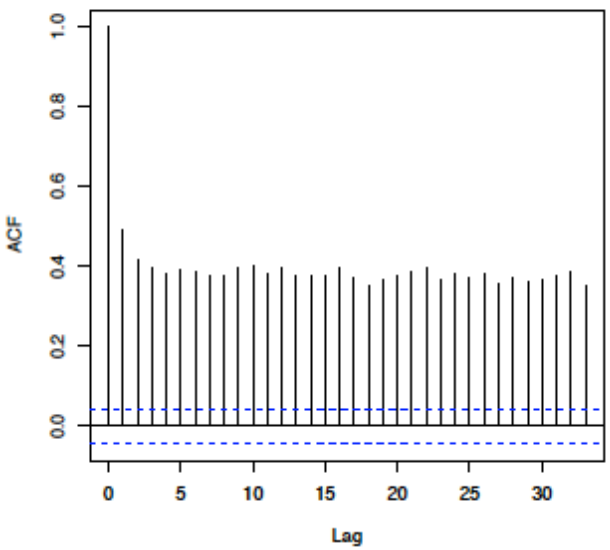

Markov Chain Length = 100,000
Markov Chain Length = 1,000,000
LnL
−10^5 −9*10^4 −8*10^4 −7*10^4 −6*10^4
0
20000
40000
60000
80000
100000
2*10^5
4*10^5
6*10^5
8*10^5
10^6
generation number
TL
30
35
40
45
50
25
alpha
0.4
0.6
0.8
1.0
1.2
1.4


Good Mixing
θ
4
2
0
−2
0
2000
4000
6000
8000
10000
iteration

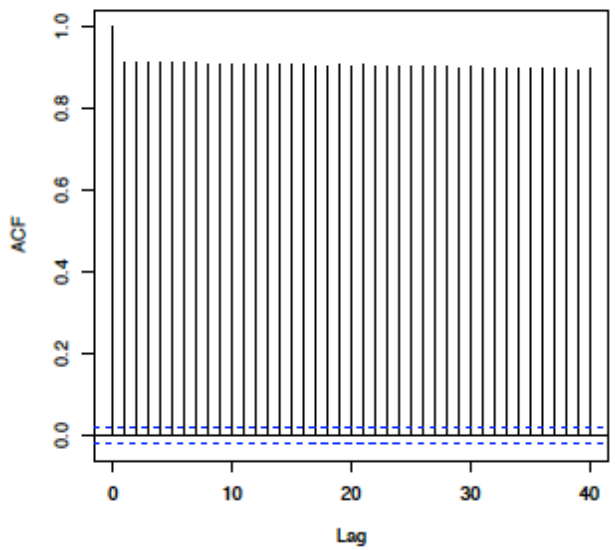

Bad Mixing
θ
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
2000
4000
6000
8000
10000
iteration

- Autocorrelation plots (i^{th} bar shows correlation between X(t) and X(t+i))

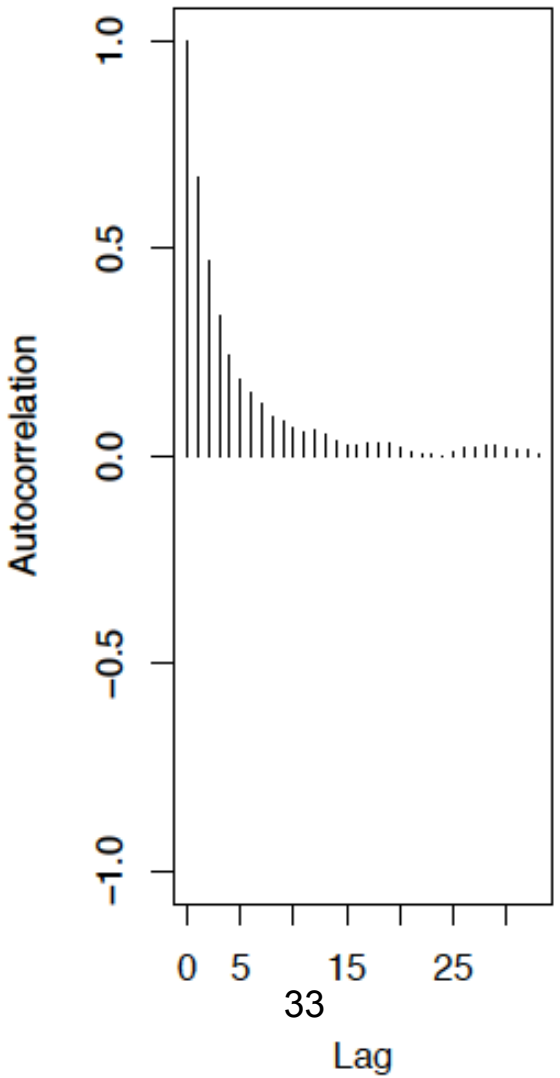
Bad mixing



Really bad mixing



Good mixing



Metropolis-Hastings: Features

- Pros:
  - The algorithm ‘learns’.
  - Comparison is ‘local’.
  - MCMC package in R [next week!], Gelman/CODA convergence diagnostics, WinBUGS, OpenBUGS, JAGS, many others…

- Cons:
  - Need to be able to calculate f(x) (may be a problem in some cases).
  - Need to sample from stationary distribution.
  - Consecutive outputs are correlated.
  - May get stuck in local minima

Gelman stationarity test

- Run two (or more) replicate analyses and compare the answers.
- Compare the variance of the parameter distribution in the two (or more) chains.
- If both chains have reached stationarity, the variance within each replicate analysis will be the same as that across (i.e. after combining) the two replicates.
- So the Gelman test compares “across run” variance to “within run” variance (dividing the former by the latter). Stationarity has been reach if the statistic is close to 1 (“less than 1.1” is a common “rule of thumb”).

Non examinable exercise 2

- Code up the Metropolis-Hastings MCMC algorithm for sized-biased sampling from an arbitrary distribution over the range [0,10].
  - Test your algorithm by using it to construct sized-biased samples from an exp(λ) distribution. Plot your results and compare them to the curve $x\lambda e^{-\lambda x}$.
  - Recall: Sized-biased samples from a density f(x) have density xf(x), rather than f(x)
  - Note that your algorithm will work for any density function f.

Pseudocode on Github in SizeBiasedMCMC

Gelman stationarity test

```
# Gelman code
library("coda")
```

Example on GitHub in the SizeBiasedMCMC repo as SizeBiasedMCMCWorking.R

```
# for example
SB<-SizedBiasedMCMC(Exponential,1,100000,1,0,10)
SB2<-SizedBiasedMCMC(Exponential,1,100000,1,0,10)
# convert to mcmc objects, with a burn-in - the Gelman routine needs them as column vectors (each variable in a column)
# so we need to transpose them
MCMC1<-mcmc(t(SB),start=1000)
MCMC2<-mcmc(t(SB2),start=1000)
# combine different mcmc chain objects to a mcmc list.
Combined<-mcmc.list(list(MCMC1,MCMC2))
# gelman functions are
gelman.plot(Combined) # for plots
gelman.diag(Combined) # for diagnostic values
```

Examinable assignment 3 - part 1

- Use the IndeptGamma.R code to use MCMC to produce samples from a Gamma(2.3,2.7) random variable
- Show how the performance deteriorates when you run the algorithm to produce samples from a Gamma(0.1,0.01).
  - Use the Gelman plot to illustrate the deterioration in performance
- Find a proposal kernel (i.e. q(x->x'), the way of producing new candidate values) that performs more efficiently.
  - Again use the Gelman plots to show how performance has changed.
  - Discuss how the performance has improved

- Due On March 30th, at 1pm

Code on GitHub in Assignment3 as 'IndeptGamma.R'

Examinable assignment 3 - part 1

```
# metropolis-hastings independence sampler for a
# gamma based on normal candidates with the same mean and variance
set.seed(371)
gamm<-function (n, a, b)
{
  mu <- a/b   # the mean of the gamma distribution
  sig <- sqrt(a/(b * b))   # the stadard deviation of the gamma distn
  vec <- vector("numeric", n)   # this is where we are going to put the random variables we generate
  x <- a/b
  vec[1] <- x # We arbitrarily start the MCMC process at the mean
  for (i in 2:n) {
    can <- rnorm(1, mu, sig)
    hprob <- min(1, (dgamma(can, a, b)/dgamma(x,a,b))/(dnorm(can, mu, sig)/dnorm(x, mu, sig)))   # where is the q term here?
    u <- runif(1)
    if (u < hprob)
      x <- can
    vec[i] <- x
  }
  vec
}
```

Code on GitHub in Assignment3 repo as ‘IndeptGamma.R’

Sampling from Posterior Distributions

- Data, D; model parameter(s) θ. Density f(D|θ).

- Bayes theorem:

$$f(\theta|D) = f(D|\theta)\pi(\theta)/f(D)$$

Likelihood

Prior distribution

Normalizing constant

MCMC: Metropolis-Hastings

1. If at θ, propose move to θ' according to transition kernel q(θ -> θ').
2. Calculate

$$h = \min\left\{1, \frac{[P(\theta'|D)q(\theta' \to \theta)}{[P(\theta|D)q(\theta \to \theta')}\right\}$$

3. Move to θ' with prob. h, else remain at θ.
4. Return to 1.

The stationary distribution of this chain will be P(θ | D)

MCMC: Metropolis-Hastings

1. If at θ, propose move to θ' according to transition kernel $q(\theta \rightarrow \theta')$.
2. Calculate

$$h = \min\left\{1, \frac{[P(D|\theta')\pi(\theta')/P(D)]q(\theta' \rightarrow \theta)}{[P(D|\theta)\pi(\theta)/P(D)]q(\theta \rightarrow \theta')}\right\}$$

3. Move to θ' with prob. h, else remain at θ.
4. Return to 1.

The stationary distribution of this chain will be $P(\theta \mid D)$

MCMC: Metropolis-Hastings

1. If at θ, propose move to θ' according to transition kernel q(θ -> θ').
2. Calculate

$$h = \min\left\{1, \frac{P(D|\theta')\pi(\theta')q(\theta' \to \theta)}{P(D|\theta)\pi(\theta)q(\theta \to \theta')}\right\}$$

3. Move to θ' with prob. h, else remain at θ.
4. Return to 1.

The stationary distribution of this chain will be P(θ | D).

P(D) disappears, **so this can be used even when *P(D)* cannot be calculated!**

Examinable Assignment 3 part 2: Code-breaking (due March 30th, 1pm)

- Gzo uclfg gcpo C qhcs okof te Gollk Qoeetb zo rhf slvem ce h LtqqfLtkio Fcqxol Rlhcgz tvgfcso gzo gollhio tu Gzo Sheiolf. Gzo ahlmced qtg hggoesheg zhs yltvdzg gzo ihl tvg hes zo rhf fgcqq ztqsced gzo sttl taoe yoihvfo Gollk Qoeetbf qoug uttg rhf fgcqq shedqced tvgfcso, hf cu zo zhs utldtggoe zo zhs teo. Zo zhs h ktvedqttmced uhio yvg zcf zhcl rhf yteo rzcgo. Ktv itvqs goqq yk zcf okof gzhg zo rhf aqhfgolos gt gzo zhclqceo, yvg tgzolrcfo zo qttmos qcmo hek tgzol ecio ktveddvk ce h sceeol jhimog rzt zhs yooe faoesced gtt pviz pteok ce h jtceg gzhg obcfgf utl gzhg avlatfo hes utl et tgzol. Gzolo rhf h dclq yofcso zcp. Zol zhcl rhf h qtxoqk fzhso tu shlm los hes fzo zhs h scfgheg fpcqo te zol qcaf hes txol zol fztvqsolf fzo zhs h yqvo pcem gzhg hqptfg phso gzo LtqqfLtkio qttm qcmo jvfg hetgzol hvgtptycqo. Cg scseg wvcgo. Etgzced ihe. Gzo hggoesheg rhf gzo vfvhq zhqugtvdz izhlhigol ce h rzcgo ithg rcgz gzo ehpo tu gzo lofghvlheg fgcgizos hiltff gzo ulteg tu cg ce los. Zo rhf doggced uos va. "Qttm, pcfgol," zo fhcs rcgz he osdo gt zcf xtcio, "rtvqs ktv pces h rztqo qtg avqqced ktvl qod cegt gzo ihl ft C ihe mces tu fzvg gzo sttl Tl fztvqs C taoe cg hqq gzo rhk ft ktv ihe uhqq tvg" Gzo dclq dhxo zcp h qttm rzciz tvdzg gt zhxo fgvim hg qohfg utvl ceizof tvg tu zcf yhim. Cg scseg ytgzol zcp oetvdz gt dcxo zcp gzo fzhmof. Hg Gzo Sheiolf gzok dog gzo ftlg tu aotaqo gzhg scfcqqvfcte ktv hytvg rzhg h qtg tu dtquced pteok ihe st utl gzo aolftehqcgk.

[CodedMessage_Short.txt in Assignment3 repo on Github]

Your clues:

- It is a simple substitution code
  - e.g. “a/A” might stand for “m/M”; “b/B” might stand for “f/F”
- The substitution is the same for the entire message.
- I have prepared a table of pairwise letter frequencies…
- We are doing this just after we learned about optimization and MCMC.

Pairwise letter frequencies

- 1 20 33 52 0.1 12 18 5 39 1 12 57 26 181 1 20 1 75 95 104 9 20 13 1 26 1
- 11 1 0.1 0.1 47 0.1 0.1 0.1 6 1 0.1 17 0.1 0.1 19 0.1 0.1 11 2 1 21 0.1 0.1 0.1 11 0.1
- 31 0.1 4 0.1 38 0.1 0.1 38 10.1 0.1 18 9 0.1 0.1 45 0.1 1 11 1 15 7 0.1 0.1 0.1 1 0.1
- 48 20 9 13 57 11 7 25 50.1 3 1 11 14 16 41 6 0.1 14 35 56 10.1 2 19 0.1 10.1 0.1
- 110 23 45 126 48 30 15 33 41 3 5 55 47 111 33 28 2 169 115 83 6 24 50 9 26 0.1
- 25 2 3 2 20 11 1 8 23 1 0.1 8 5 1 40 2 0.1 16 5 37 8 0.1 3 0.1 2 0.1
- 24 3 2 2 28 3 4 35 18 1 0.1 7 3 4 23 1 0.1 12 9 16 7 0.1 5 0.1 1 0.1
- 114 2 2 1 302 2 1 6 97 0.1 0.1 2 3 1 49 1 0.1 8 5 32 8 0.1 4 0.1 4 0.1
- 10 5 32 33 23 17 25 6 1 1 8 37 37 179 24 6 0.1 27 86 93 1 14 7 2 0.1 2
- 2 0.1 0.1 0.1 2 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 8 0.1 0.1 0.1 0.1 0.1
- 6 1 1 1 29 1 0.1 2 14 0.1 0.1 2 1 9 4 0.1 0.1 0.1 5 4 1 0.1 2 0.1 2 0.1
- 40 3 2 36 64 10 1 4 47 0.1 3 56 4 2 41 3 0.1 2 11 15 8 3 5 0.1 31 0.1
- 44 7 1 1 68 2 1 3 25 0.1 0.1 1 5 2 29 11 0.1 3 10.1 9 8 0.1 4 0.1 18 0.1
- 40 7 25 146 66 8 92 16 33 2 8 9 7 8 60 4 1 3 33 106 6 2 12 0.1 11 0.1
- 16 12 13 18 5 80 7 11 12 1 13 26 48 106 36 15 0.1 84 28 57 115 12 46 0.1 5 1
- 23 1 0.1 0.1 30 1 0.1 3 12 0.1 0.1 15 1 0.1 21 10.1 0.1 18 5 11 6 0.1 1 0.1 1 0.1
- 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 9 0.1 0.1 0.1 0.1 0.1
- 50 7 10.1 20 133 8 10 12 50 1 8 10 14 16 55 6 0.1 14 37 42 12 4 11 0.1 21 0.1
- 67 11 17 7 74 11 4 50 49 2 6 13 12 10 57 20 2 4 43 109 20 2 24 0.1 4 0.1
- 59 10 11 7 75 9 3 330 76 1 2 17 11 7 115 4 0.1 28 34 56 17 1 31 0.1 16 0.1
- 7 5 12 7 7 2 14 2 8 0.1 1 34 8 36 1 16 0.1 44 35 48 0.1 0.1 2 0.1 1 0.1
- 5 0.1 0.1 0.1 65 0.1 0.1 0.1 11 0.1 0.1 0.1 0.1 0.1 4 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 1 0.1
- 66 1 1 2 39 1 0.1 44 39 0.1 0.1 2 1 12 29 0.1 0.1 3 4 4 1 0.1 2 0.1 1 0.1
- 1 0.1 2 0.1 1 0.1 0.1 0.1 2 0.1 0.1 0.1 0.1 0.1 0.1 3 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 0.1
- 18 7 6 6 14 7 3 10.1 11 1 1 4 6 3 36 4 0.1 3 19 20.1 1 1 12 0.1 2 0.1
- 1 0.1 0.1 0.1 3 0.1 0.1 0.1 1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1

[LetterPairFreqFrom7Novels.txt in GitHub Assignment 3 repo

Pairwise letter frequencies

- 1 20 33 52 0.1 12 18 5 39 1 12 57 26 181 1 20 1 75 95 104 9 20 13 1 26 1
- 11 1 0.1 0.1 47 0.1 0 1 0.1 6 1 0.1 17 0.1 0.1 19 0.1 0.1 11 2 1 21 0.1 0.1 0.1 11 0.1
- 31 0.1 4 0.1 38 0.1 0.1 38 10.1 0.1 18 9 0.1 0.1 45 0.1 1 11 1 15 7 0.1 0.1 0.1 1 0.1
- 48 20 9 13 57 11 7 25 50.1 3 1 11 14 16 41 6 0.1 14 35 56 10.1 2 19 0.1 10.1 0.1
- 110 23 45 126 48 30 15 33 41 3 5 55 47 111 33 28 2 169 115 83 6 24 50 9 26 0.1
- 25 2 3 2 20 11 1 8 23 1 0.1 8 5 1 40 2 0.1 16 5 37 8 0.1 3 0.1 2 0.1
- 24 3 2 2 28 3 4 35 18 1 0.1 7 3 4 23 1 0.1 12 9 16 7 0.1 5 0.1 1 0.1
- 114 2 2 1 302 2 1 6 97 0.1 0.1 2 3 1 49 1 0.1 8 5 32 8 0.1 4 0.1 4 0.1
- 10 5 32 33 23 17 25 6 1 1 8 37 37 179 24 6 0.1 27 86 93 1 14 7 2 0.1 2
- 2 0.1 0.1 0.1 2 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 8 0.1 0.1 0.1 0.1 0.1
- 6 1 1 1 29 1 0.1 2 14 0.1 0.1 2 1 9 4 0.1 0.1 0.1 5 4 1 0.1 2 0.1 2 0.1
- 40 3 2 36 64 10 1 4 47 0.1 3 56 4 2 41 3 0.1 2 11 15 8 3 5 0.1 31 0.1
- 44 7 1 1 68 2 1 3 25 0.1 0.1 1 5 2 29 11 0.1 3 10.1 9 8 0.1 4 0.1 18 0.1
- 40 7 25 146 66 8 92 16 33 2 8 9 7 8 60 4 1 3 33 106 6 2 12 0.1 11 0.1
- 16 12 13 18 5 80 7 11 12 1 13 26 48 106 36 15 0.1 84 28 57 115 12 46 0.1 5 1
- 23 1 0.1 0.1 30 1 0.1 3 12 0.1 0.1 15 1 0.1 21 10.1 0.1 18 5 11 6 0.1 1 0.1 1 0.1
- 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 9 0.1 0.1 0.1 0.1 0.1
- 50 7 10.1 20 133 8 10 12 50 1 8 10 14 16 55 6 0.1 14 37 42 12 4 11 0.1 21 0.1
- 67 11 17 7 74 11 4 50 49 2 6 13 12 10 57 20 2 4 43 109 20 2 24 0.1 4 0.1
- 59 10 11 7 75 9 3 330 76 1 2 17 11 7 115 4 0.1 28 34 56 17 1 31 0.1 16 0.1
- 7 5 12 7 7 2 14 2 8 0.1 1 34 8 36 1 16 0.1 44 35 48 0.1 0.1 2 0.1 1 0.1
- 5 0.1 0.1 0.1 65 0.1 0.1 0.1 11 0.1 0.1 0.1 0.1 0.1 4 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 1 0.1
- 66 1 1 2 39 1 0.1 44 39 0.1 0.1 2 1 12 29 0.1 0.1 3 4 4 1 0.1 2 0.1 1 0.1
- 1 0.1 2 0.1 1 0.1 0.1 0.1 2 0.1 0.1 0.1 0.1 0.1 0.1 3 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 0.1
- 18 7 6 6 14 7 3 10.1 11 1 1 4 6 3 36 4 0.1 3 19 20.1 1 1 12 0.1 2 0.1
- 1 0.1 0.1 0.1 3 0.1 0.1 0.1 1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1

How often “a” was followed by “a”

[LetterPairFreqFrom7Novels.txt in GitHub Assignment 3 repo

Pairwise letter frequencies

- 1 20 33 52 0.1 12 18 5 39 1 12 57 26 181 1 20 1 75 95 104 9 20 13 1 26 1
- 11 1 0.1 0.1 47 0.1 0.1 0.1 6 1 0.1 17 0.1 0.1 19 0.1 0.1 11 2 1 21 0.1 0.1 0.1 11 0.1
- 31 0.1 4 0.1 38 0.1 0.1 38 10.1 0.1 18 9 0.1 0.1 45 0.1 1 11 1 15 7 0.1 0.1 0.1 1 0.1
- 48 20 9 13 57 11 7 25 50.1 3 1 11 14 16 41 6 0.1 14 35 56 10.1 2 19 0.1 10.1 0.1
- 110 23 45 126 48 30 15 33 41 3 5 55 47 111 33 28 2 169 115 83 6 24 50 9 26 0.1
- 25 2 3 2 20 11 1 8 23 1 0.1 8 5 1 40 2 0.1 16 5 37 8 0.1 3 0.1 2 0.1
- 24 3 2 2 28 3 4 35 18 1 0.1 7 3 4 23 1 0.1 12 9 16 7 0.1 5 0.1 1 0.1
- 114 2 2 1 302 2 1 6 97 0.1 0.1 2 3 1 49 1 0.1 8 5 32 8 0.1 4 0.1 4 0.1
- 10 5 32 33 23 17 25 6 1 1 8 37 37 179 24 6 0.1 27 86 93 1 14 7 2 0.1 2
- 2 0.1 0.1 0.1 2 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 8 0.1 0.1 0.1 0.1 0.1
- 6 1 1 1 29 1 0.1 2 14 0.1 0.1 2 1 9 4 0.1 0.1 0.1 5 4 1 0.1 2 0.1 2 0.1
- 40 3 2 36 64 10 1 4 47 0.1 3 56 4 2 41 3 0.1 2 11 15 8 3 5 0.1 31 0.1
- 44 7 1 1 68 2 1 3 25 0.1 0.1 1 5 2 29 11 0.1 3 10.1 9 8 0.1 4 0.1 18 0.1
- 40 7 25 146 66 8 92 16 33 2 8 9 7 8 60 4 1 3 33 106 6 2 12 0.1 11 0.1
- 16 12 13 18 5 80 7 11 12 1 13 26 48 106 36 15 0.1 84 28 57 115 12 46 0.1 5 1
- 23 1 0.1 0.1 30 1 0.1 3 12 0.1 0.1 15 1 0.1 21 10.1 0.1 18 5 11 6 0.1 1 0.1 1 0.1
- 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 9 0.1 0.1 0.1 0.1 0.1
- 50 7 10.1 20 133 8 10 12 50 1 8 10 14 16 55 6 0.1 14 37 42 12 4 11 0.1 21 0.1
- 67 11 17 7 74 11 4 50 49 2 6 13 12 10 57 20 2 4 43 109 20 2 24 0.1 4 0.1
- 59 10 11 7 75 9 3 330 76 1 2 17 11 7 115 4 0.1 28 34 56 17 1 31 0.1 16 0.1
- 7 5 12 7 7 2 14 2 8 0.1 1 34 8 36 1 16 0.1 44 35 48 0.1 0.1 2 0.1 1 0.1
- 5 0.1 0.1 0.1 65 0.1 0.1 0.1 11 0.1 0.1 0.1 0.1 0.1 4 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 1 0.1
- 66 1 1 2 39 1 0.1 44 39 0.1 0.1 2 1 12 29 0.1 0.1 3 4 4 1 0.1 2 0.1 1 0.1
- 1 0.1 2 0.1 1 0.1 0.1 0.1 2 0.1 0.1 0.1 0.1 0.1 0.1 3 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 0.1
- 18 7 6 6 14 7 3 10.1 11 1 1 4 6 3 36 4 0.1 3 19 20.1 1 1 12 0.1 2 0.1
- 1 0.1 0.1 0.1 3 0.1 0.1 0.1 1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1

How often “a” was followed by “b”

[LetterPairFreqFrom7Novels.txt in GitHub Assignment 3 repo

Pairwise letter frequencies

- 1 20 33 52 0.1 12 18 5 39 1 12 57 26 181 1 20 1 75 95 104 9 20 13 1 26 1
- 11 1 0.1 0.1 47 0.1 0.1 0.1 6 1 0.1 17 0.1 0.1 19 0.1 0.1 11 2 1 21 0.1 0.1 0.1 11 0.1
- 31 0.1 4 0.1 38 0.1 0.1 38 10.1 0.1 18 9 0.1 0.1 45 0.1 1 11 1 15 7 0.1 0.1 0.1 1 0.1
- 48 20 9 13 57 11 7 25 50.1 3 1 11 14 16 41 6 0.1 14 35 56 10.1 2 19 0.1 10.1 0.1
- 110 23 45 126 48 30 15 33 41 3 5 55 47 111 33 28 2 169 115 83 6 24 50 9 26 0.1
- 25 2 3 2 20 11 1 8 23 1 0.1 8 5 1 40 2 0.1 16 5 37 8 0.1 3 0.1 2 0.1
- 24 3 2 2 28 3 4 35 18 1 0.1 7 3 4 23 1 0.1 12 9 16 7 0.1 5 0.1 1 0.1
- 114 2 2 1 302 2 1 6 97 0.1 0.1 2 3 1 49 1 0.1 8 5 32 8 0.1 4 0.1 4 0.1
- 10 5 32 33 23 17 25 6 1 1 8 37 37 179 24 6 0.1 27 86 93 1 14 7 2 0.1 2
- 2 0.1 0.1 0.1 2 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 8 0.1 0.1 0.1 0.1 0.1
- 6 1 1 1 29 1 0.1 2 14 0.1 0.1 2 1 9 4 0.1 0.1 0.1 5 4 1 0.1 2 0.1 2 0.1
- 40 3 2 36 64 10 1 4 47 0.1 3 56 4 2 41 3 0.1 2 11 15 8 3 5 0.1 31 0.1
- 44 7 1 1 68 2 1 3 25 0.1 0.1 1 5 2 29 11 0.1 3 10.1 9 8 0.1 4 0.1 18 0.1
- 40 7 25 146 66 8 92 16 33 2 8 9 7 8 60 4 1 3 33 106 6 2 12 0.1 11 0.1
- 16 12 13 18 5 80 7 11 12 1 13 26 48 106 36 15 0.1 84 28 57 115 12 46 0.1 5 1
- 23 1 0.1 0.1 30 1 0.1 3 12 0.1 0.1 15 1 0.1 21 10.1 0.1 18 5 11 6 0.1 1 0.1 1 0.1
- 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 9 0.1 0.1 0.1 0.1 0.1
- 50 7 10.1 20 133 8 10 12 50 1 8 10 14 16 55 6 0.1 14 37 42 12 4 11 0.1 21 0.1
- 67 11 17 7 74 11 4 50 49 2 6 13 12 10 57 20 2 4 43 109 20 2 24 0.1 4 0.1
- 59 10 11 7 75 9 3 330 76 1 2 17 11 7 115 4 0.1 28 34 56 17 1 31 0.1 16 0.1
- 7 5 12 7 7 2 14 2 8 0.1 1 34 8 36 1 16 0.1 44 35 48 0.1 0.1 2 0.1 1 0.1
- 5 0.1 0.1 0.1 65 0.1 0.1 0.1 11 0.1 0.1 0.1 0.1 0.1 4 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 1 0.1
- 66 1 1 2 39 1 0.1 44 39 0.1 0.1 2 1 12 29 0.1 0.1 3 4 4 1 0.1 2 0.1 1 0.1
- 1 0.1 2 0.1 1 0.1 0.1 0.1 2 0.1 0.1 0.1 0.1 0.1 0.1 3 0.1 0.1 0.1 3 0.1 0.1 0.1 0.1 0.1 0.1
- 18 7 6 6 14 7 3 10.1 11 1 1 4 6 3 36 4 0.1 3 19 20.1 1 1 12 0.1 2 0.1
- 1 0.1 0.1 0.1 3 0.1 0.1 0.1 1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1

How often “a” was followed by “c”

[LetterPairFreqFrom7Novels.txt in GitHub Assignment 3 repo

R commands to read coded messages

```
# read coded message

input2<- readLines("CodedMessage_Short.txt", n=1000)

# read frequency table

PairsFreqs<-read.table(“LetterPairFreqFrom7Novels.txt")
```

Assignment 3

- Use one of the methods you have met in the course to try to break the code.
- I may not be easy to break the code completely without resorting to manual ‘tweaks’ at the end, but you should be able to get to the point at which you can work out what the text is saying.
- Write it up as an Rmarkdown file and include a description of the methods you are using (for both parts of the assigment) and a discussion of your results.
- Due March 30th, 1pm.

Suggestion

- Need something to maximize, so let's maximize the likelihood.
- Suppose we have a sequence of letters $l_1\ l_2\ l_3\ l_4\ l_5$. Then [Recall $P(A|B)=P(A\cap B)/P(B)$]:
  - $P(l_1\ l_2\ l_3\ l_4\ l_5)=P(l_1)\ P(l_2|l_1)\ P(l_3|l_2\ l_1)\ P(l_4|l_3\ l_2\ l_1)\ P(l_5|l_4\ l_3\ l_2\ l_1)$
- Suppose the sequence of letters is Markovian. Then:
  - $P(l_1\ l_2\ l_3\ l_4\ l_5)=P(l_1)\ P(l_2|l_1)\ P(l_3|l_2)\ P(l_4|l_3)\ P(l_5|l_4)$
  - $P(l_1\ l_2\ l_3\ l_4\ l_5)=P(l_1)\ [P(l_1l_2)/P(l_1)]\ [P(l_2l_3)/P(l_2)]\ [P(l_3l_4)/P(l_3)]\ [P(l_4l_5)/P(l_4)]$
- Let $f_{\alpha\beta}$ denote the frequency with which the letter pair $\alpha\beta$ is observed in text (from the file on Blackboard). Let f_α denote the frequency with which the letter α is observed in text. Then $f_\alpha = \sum_\beta f_{\alpha\beta}$.
- Use $f_{\alpha\beta}$ as an estimate of $P(\alpha\beta)$ and f_α as an estimate of $P(\alpha)$.
- Now treat it as a maximization (of likelihood) or MCMC problem (in which case you get a posterior distribution over all possible decodings).

END