Solutions to Homework 2

Spring 2008

1. Write the entropy as $-\sum_{i=1}^{m} p_i \log_2 p_i$, remembering (Figure 1) that $0 \log 0 = 0$.

   (a) $0 \leq p_i \leq 1$ for all i, so $\log_2 p_i \leq 0$, and $-p_i \log_2 p_i \geq 0$. Hence, $H \geq 0$, being a sum of non-negative terms.

   (b) There are several ways to do this one. Using the hint, write

   $$p_i = \frac{1}{m} + \delta_i \tag{1}$$

   Since $\sum_i p_i = 1$, we must also have $\sum_i \delta_i = 0$. What we'd like to show is that H is maximized when all the $\delta_i = 0$. At this point, we have a situation like that in note about maximizing likelihood for Markov chain. We can either solve for one of the δ_i in terms of the others, and maximize with respect to them, or we can use Lagrange multipliers. Let's do that: the Lagrangian is

   $$\mathcal{L} = -\sum_{i=1}^{m} \left(\frac{1}{m} + \delta_i\right) \log_2 \left(\frac{1}{m} + \delta_i\right) + \lambda \sum_i \delta_i \tag{2}$$

   Taking the derivative of $\mathcal{L}$ with respect to δ_i,

   $$\begin{aligned} 0 &= -\left[\log_2 \frac{1}{m} + \delta_i + \frac{1}{\ln 2} \frac{\frac{1}{m} + \delta_i}{\frac{1}{m} + \delta_i}\right] + \lambda & (3)\\ &= -\log_2 \left(\frac{1}{m} + \delta_i\right) - \frac{1}{\ln 2} + \lambda & (4)\\ \log_2 \left(\frac{1}{m} + \delta_i\right) &= -\frac{1}{\ln 2} + \lambda & (5)\\ \frac{1}{m} + \delta_i &= 2^{-\frac{1}{\ln 2} + \lambda} & (6)\\ \delta_i &= -\frac{1}{m} + 2^{-\frac{1}{\ln 2} + \lambda} & (7) \end{aligned}$$

   On the other hand, taking the derivative of $\mathcal{L}$ with respect to λ, we recover the constraint:

   $$\sum_{i=1}^{n} \delta_i = 0 \tag{8}$$

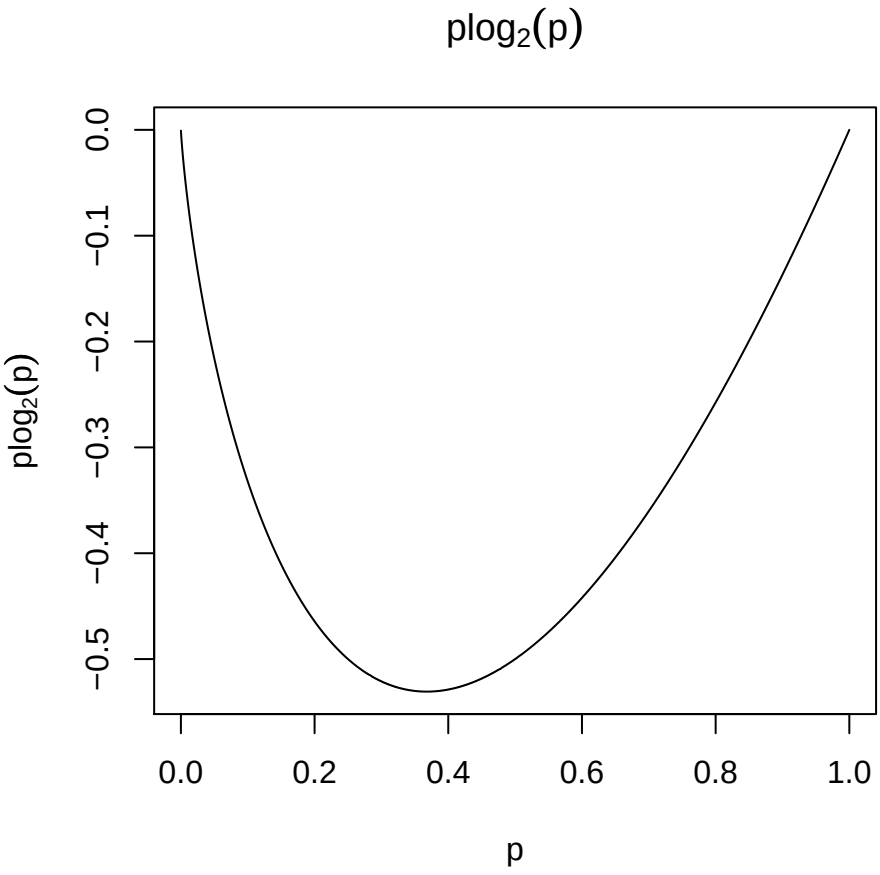

Figure 1: Plot of $p \log_2 p$ versus p. Note that $0 \log 0 = 0$.

Plugging in Eq. 7, and noticing that it is the *same* for all i,

$$
\begin{aligned}
0 &= \sum_{i=1}^{m} -\frac{1}{m} + 2^{-\frac{1}{\ln 2}+\lambda} && (9)\\
&= m\left[-\frac{1}{m} + 2^{-\frac{1}{\ln 2}+\lambda}\right] && (10)\\
&= -\frac{1}{m} + 2^{-\frac{1}{\ln 2}+\lambda} && (11)
\end{aligned}
$$

so (going back to Eq. 7 again) $\delta_i = 0$, as desired.

2. We need to do three things here. First, we need to generate symbol sequences from the logistic map. Second, we need to count the number of times each word of length L appears, and how often it is immediately followed by each other word of length L. Finally, we need to somehow use thouse counts to test independence. (Notice, by the way, that the second part, about gathering count statistics of words, is something we're going to have to use a lot in the other problems.) I'll do this by writing a bunch of small functions, each of which solves a small piece of the problem; this makes it easier to test that they're doing what they are supposed to, and to re-use them later, if need be.

   The first part uses some code for the logistic map (taken from `03.R`, plus the discretization function.

```
logistic.map <- function(x,r) {
  return(4*r*x*(1-x))
}

logistic.map.ts <- function(timelength,r,initial.cond=NULL) {
  x <-vector(mode="numeric",length=timelength)
  if(is.null(initial.cond)) {
    x[1] <-runif(1)
  } else {
    x[1] <-initial.cond
  }
  for (t in 2:timelength) {
    x[t] = logistic.map(x[t-1],r)
  }
  return(x)
}

logistic.genpart <- function(x) {
  # Apply the generating partition of the logistic map to a vector of
  # real values
  # So return "L" at each position where x < 0.5 and "R" elsewhere
  ifelse(x<0.5,'L','R')
```

```
}

logistic.symbseq <- function(timelength,r) {
  # Get a real-valued trajectoy from the logistic map
  x <- logistic.map.ts(timelength,r)
  # Apply the generating partition and return that
  s <- logistic.genpart(x)
  return(s)
}
```

The second part needs us to take a window of length L (not to be confused with the symbol "L"!), slide it along the symbol sequence, and record all the patterns we see. We also need to keep track of which pattern followed that one, in the next L block. Start with a function to find all the different blocks, in order.

```
symbseq.to.blocks <- function(s,L) {
  # Take a symbol sequence and return a list of the (overlapping) blocks
  # of length L it contains
  n <- length(s)
  # A length L block can't start at any position whose index is greater
  # than n-L+1 (though it could start there).
  # Should really check that this is < n, and return an error if not!
  max.index <- n -L+1
  blocks <- NULL # Null list
  for (i in 1:max.index) {
    blocks <- c(blocks, paste(s[i:(i+L-1)],collapse=""))
  }
  return(blocks)
}
```

Now we need to get not just each block of length L, but also the following block of length L. We'll do this by taking the complete list of L-blocks and splitting it into two parts (which in general will overlap).

```
symbseq.to.successive.blocks <- function(s,L) {
  n <- length(s)
  # Produce the complete list of length-L blocks
  all.blocks <- symbseq.to.blocks(s,L)
  # The "leaders" begin at positions 1, 2, ... n-2L+1 (because there
  # needs to be another, following block of length L after each of them
  max.index.leaders <- n-2*L+1
  # The "followers" begin at positions L+1, L+2, ... n-L+1 (because there
  # needs to be a "leader" block of lenght L before each of them
  min.index.followers <- L+1
  max.index.followers <- n-L+1
```

```
  leaders <- all.blocks[1:max.index.leaders]
  followers <- all.blocks[min.index.followers:max.index.followers]
  return(list(leaders=leaders,followers=followers))
}
```

At this point it's a good idea to check that everything is working right with a small example.

```
> ss <- c("L","R","L","R","L","R","L")
> symbseq.to.blocks(ss,2)
[1] "LR" "RL" "LR" "RL" "LR" "RL"
> symbseq.to.successive.blocks(ss,2)
$leaders
[1] "LR" "RL" "LR" "RL"

$followers
[1] "LR" "RL" "LR" "RL"
```

You can check by hand that the code works on this example, and on this one:

```
> rr <- c("L","L","L","R","L","R","R")
> symbseq.to.blocks(rr,2)
[1] "LL" "LL" "LR" "RL" "LR" "RR"
> symbseq.to.successive.blocks(rr,2)
$leaders
[1] "LL" "LL" "LR" "RL"

$followers
[1] "LR" "RL" "LR" "RR"
```

Ideally at this point I'd check an $L = 3$ case, but I'm just the teacher here.

Finally, we need to test whether the follower blocks are statitically independent of the leader blocks. The standard way to test whether two discrete random variables are independent is to use the χ^2 ("chi-squared") test. If you need a refresher on the theory of this test, I'd suggest either Wikipedia, or (better yet) Larry Wasserman's *All of Statistics*. Fortunately, this is built in to `R`, in the imaginatively-named function `chisq.test`. It needs to be given a contigency table, but there is a function to build that, called `table`. Here's how `table` works:

```
> table(symbseq.to.successive.blocks(ss,2))
       followers
leaders LR RL
     LR  2  0
     RL  0  2
```

and here's how the testing function works:

```
> chisq.test(table(symbseq.to.successive.blocks(ss,2)))

  Pearson's Chi-squared test with Yates' continuity correction

data:  table(symbseq.to.successive.blocks(ss, 2))
X-squared = 1, df = 1, p-value = 0.3173

Warning message:
In chisq.test(table(symbseq.to.successive.blocks(ss, 2))) :
  Chi-squared approximation may be incorrect
```

`chisq.test` is giving us a warning here, because the χ^2 approximation to the distribution of the test statistic is only valid if there are a fairly large number of counts for each cell in the table. The usual rule of thumb is that the expected number of counts must be at least 5; let's say 10 to be safe. Each cell in the table corresponds to a word of length $2L$, and we expect (for IID coin-tossing) that each such word is equally likely, so we want $10 = n/2^{2L}$, or $n = 10 \times 2^{2L}$.

Putting everything together, then, we can write the following program.

```
logistic.map.independence.test <- function(L,n=min(1e4,10*(2^(2*L))),r=1) {
  s <- logistic.symbseq(n,r)
  successive.blocks <- symbseq.to.successive.blocks(s,L)
  my.tab <- table(successive.blocks)
  my.test <- chisq.test(my.tab)
  return(list(p.value=my.test$p.value,test=my.test,count.table=my.tab))
}
```

The default value for n is set so that the program won't take forever to run if you should accidentally input a large L — but that's only a default so it can be over-ridden. Returning the full test results and the count table as well as the p-value is not strictly necessary but doesn't hurt. Working for different values of r is also a bonus (but just as easy as not including it).

How do we know if this is working? If the "leader" and "follower" blocks *are* independent, then the p value of the test should be uniformly distributed on $[0, 1]$, and their CDF should be a straight diagonal line. Let's check that by re-running the test a bunch of times and plotting the empirical CDF (Figure 2).

```
> plot(ecdf(replicate(1000,logistic.map.independence.test(2)$p.value)),
  xlab="Nominal p-value",ylab="True p-value",main="Distribution of p-values")
> abline(a=0,b=1,col="blue",lty=2)
```

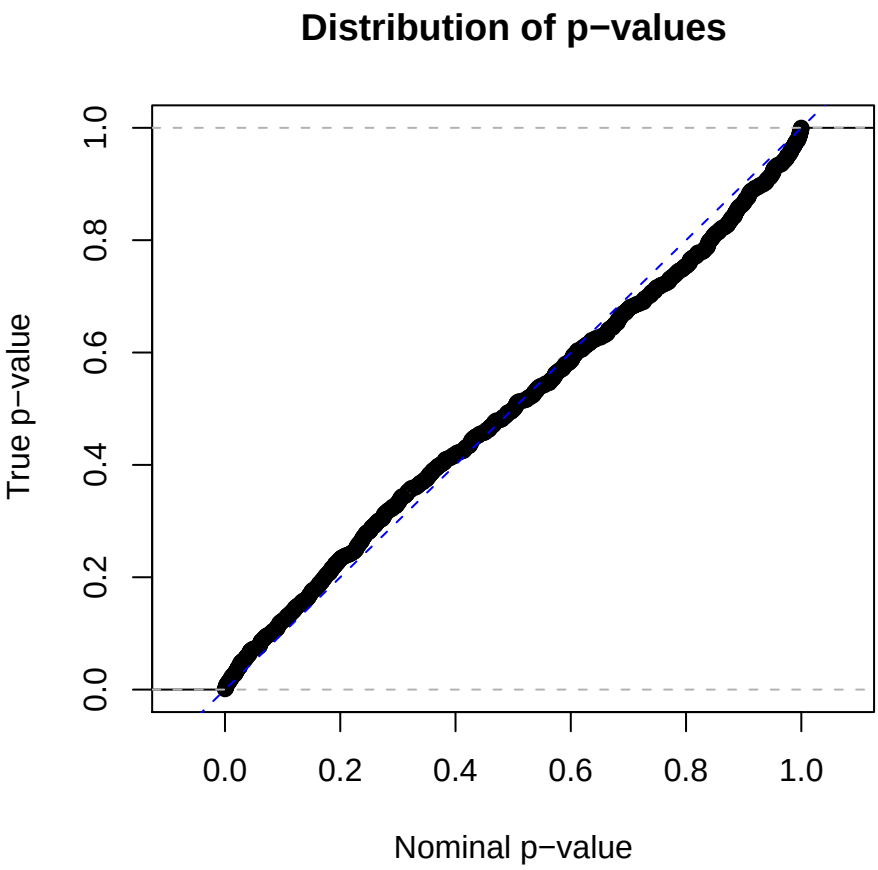

Figure 2: Distribution of p-values obtained from question 2 (black circles), with theoretical uniform distribution (dashed blue line).

3. Recall that the topological entropy rate is defined as

$$h_0 \equiv \lim_{L\to\infty} \frac{1}{L} \log W_L \tag{12}$$

where W_L is the number of allowed words of length L.

(a) There are at least three ways to do this. The simplest one is to just count the number of distinct *observed* words of length L, $\widehat{W}_L$, and estimate by division:

$$\widehat{h}_0^{division} \equiv \frac{1}{L} \log \widehat{W}_L \tag{13}$$

for some large L, as our estimate of h_0.

The second way is to notice that if the limit exists, then for large L we must have

$$\log W_L \approx C + h_0 L \tag{14}$$

where C is some constant we don't care about. So if we regress $\widehat{W}_L$ on L, the slope will be an estimate of h_0. Call this the *regression* estimate.

The third way is to use the fact, mentioned in the on-line notes about the topological entropy rate, that

$$h_0 = \lim_{L\to\infty} \log W_L - \log W_{L-1} \tag{15}$$

(Notice that this also follows from the linear expression I gave above.) So yet another estimate of h_0 is to take

$$\log \widehat{W}_L - \log \widehat{W}_{L-1} \tag{16}$$

for some large L. Call this the *difference* estimate.

All three estimates will ultimately converge on the same value, if you feed them enough data. In principle, all of them work best when the value of L is large. In practice, if L is too large relative to n, we see only a very small sample of the allowed words, i.e., $\widehat{W}_L$ becomes much smaller than W_L, introducing systematic errors into our estimate. At the very least, when $\widehat{W}_L > \widehat{W}_{L+1}$, we do not have enough data to say what is happening with the longer words.

For the logistic map, we use a binary alphabet (symbols set), so there are at most 2^L words of each length. To give us some chance of seeing each of them, we should use a symbol sequence which is at least a few times longer than the number of words we might run into, say 10×2^L.

Here's how to do the division estimate.

```
logistic.TER.division <- function(r,L,n=10*(2^L)) {
  s <- logistic.symbseq(n,r)
  blocks <- symbseq.to.blocks(s,L)
  word.table <- table(blocks)
  W.L <- dim(word.table) # Counts number of distinct allowed words
  return(log(W.L)/L)
}
```

Notice the trick with using the `table` function to identify all the *distinct* words. Let's re-cycle that for the regression estimate.

```
logistic.TER.regression <- function(r,L,n=10*(2^L)) {
  s <- logistic.symbseq(n,r)
  W <- vector(mode="numeric",length=L)
  for (i in (1:L)) {
    blocks <- symbseq.to.blocks(s,i)
    W[i] <- dim(table(blocks))
  }
  my.regression <- lm(logcounts ~ lengths,
                      data.frame(logcounts=log(W),lengths=(1:L)))
  return(as.vector(my.regression$coefficients[2]))
}
```

And here's the difference estimate:

```
logistic.TER.difference <- function(r,L,n=10*(2^L)) {
  s <- logistic.symbseq(n,r)
  lastW <- dim(table(symbseq.to.blocks(s,L)))
  nextotlastW <- dim(table(symbseq.to.blocks(s,L-1)))
  return(log(lastW) - log(nextotlastW))
}
```

To double-check these, notice that when $r = 1$, we have IID coin-tossing, and every sequence of length L is allowed, so $W_L = 2^L$. This means that h_0 should be $\log 2 = 0.6931472$.

```
> logistic.TER.division(1,3)
[1] 0.6931472
> logistic.TER.regression(1,3)
[1] 0.6931472
> logistic.TER.difference(1,3)
[1] 0.6931472
```

which checks out.

Finally, let's plot these estimates as functions of r to see if we're getting something reasonable. We know that h_0 should be zero whenever the logistic map goes to a limit cycle (see the online notes for details). I use $L = 6$ simply for reasons of speed. The plot is Figure 3.

```
> r.values <- seq(from=0,to=1,length.out=200)
> difference.values <- sapply(r.values,logistic.TER.difference,L=6)
> plot(r.values,difference.values,type="l",xlab="r",ylab="Estimated h0",
      main="topological entropy rate estimates")
> division.values <- sapply(r.values,logistic.TER.division,L=6)
> lines(r.values,division.values,lty=2)
> regression.values <- sapply(r.values,logistic.TER.regression,L=6)
> lines(r.values,regression.values,lty=3)
```

(b) The easiest way to get a value for the standard error here is simply to re-run the estimator multiple times and take the standard deviation. This only captures the error associated with the fluctuations from one run of the simulation to another, rather than the systematic errors which come from biases in the estimator, etc.

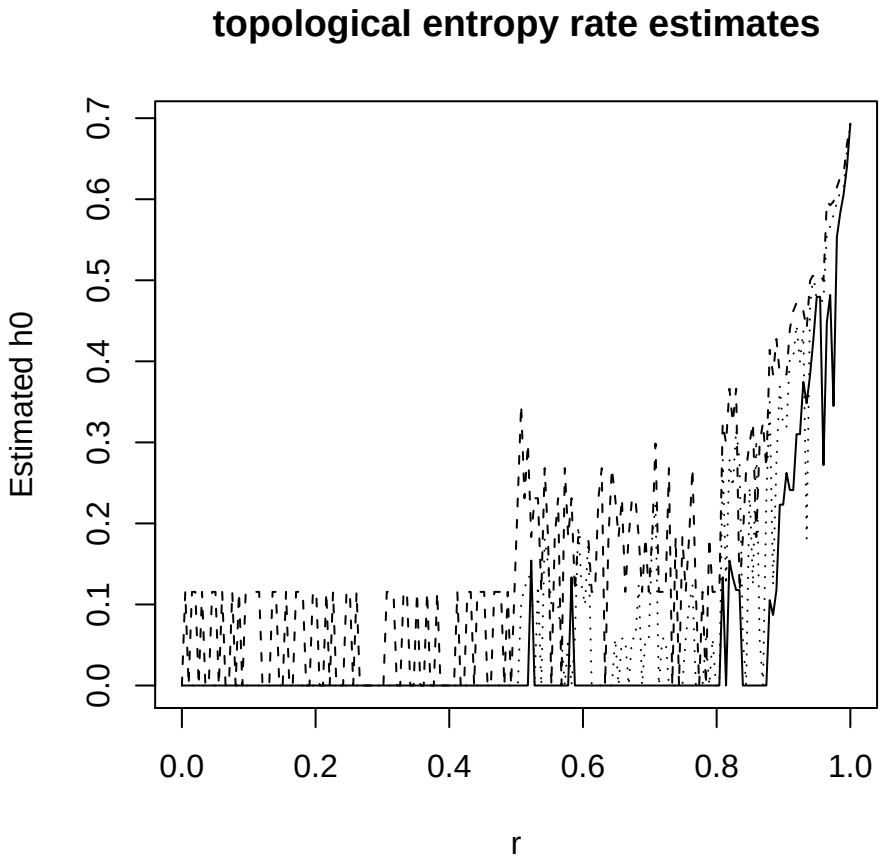

Figure 3: Three estimates of the topological entropy rate of the logistic map. Solid line, difference estimate. Dashed line, division estimate. Dotted line, regression estimate. The true value of h_0 is 0 whenever the map goes to a limit cycle, i.e., whenever $r < 0.866$ or so, suggesting that the division and regression estimates may have a larger upward bias than the difference estimate.