MTH 511a: Statistical Simulation and Data Analysis (2021-2022 - I)

Instructor

Dootika Vats
607 Rajeev Motwani Building

E-mail: dootika@iitk.ac.in
Web: http://home.iitk.ac.in/ dootika/

Course Description

As the course title suggests, this course has two aspects: statistical simulation and data analysis. The course will be broken up broadly into three parts: (i) fundamental tools for statistical simulation, (ii) some selective topics in data analysis and optimization, and (iii) specialized simulation tools for certain data analyses methods.

Prerequisites

MSO201a. Instructor's approval if you have knowledge of common discrete and continuous distributions, conditional probability, law of large numbers, central limit theorem, confidence intervals.

Lectures and Tutorials

The course will be taught synchronously on the Zoom platform. Attendance is not compulsory, but there is much to be gained from the attending the lectures and asking questions. Recordings of each lecture will be uploaded on mooKIT.

Mon: 5pm - 6pm
Wed: 5pm - 6pm
Thu: 5pm - 6pm *(tutorial)*
Frid: 5pm - 6pm

Quizzes

There will be overall *at least* 10 quizzes throughout the semester. You will often require `R` to complete the quizzes, so before starting any quiz, make sure you read the instructions on the quiz carefully.

The lowest quiz grade of each student will be dropped.

Course Webpage

The course will be running through IIT Kanpur's mooKIT portal. Videos, notes, slides, and `R` code will all be shared on mooKIT.

This term we will be using Piazza for class discussion. The system is highly catered to getting you help fast and efficiently from classmates, the TA, and myself. Rather than emailing questions to the teaching staff, I encourage you to post your questions on Piazza. The class signup link will be shared after the add/drop period is over.

Special Emphasis

This course requires you being adept at computing and fundamental statistical concepts. You must get comfortable with R before the class since online quizzes, exams, and mini-project, will require the use of `R`.

Marks Distribution

Quizzes	30%
Mid-sem Exam	30%
Mini-project	10%
Final Exam	30%

Academic Honesty

Academic integrity is essential to a positive teaching and learning environment. All students enrolled in the course are expected to complete coursework responsibilities with fairness and honesty. Failure to do so by seeking unfair advantage over others or misrepresenting someone elses work as your own, can result in disciplinary action.

> *Scholastic dishonesty means plagiarizing; cheating on assignments or examinations; engaging in unauthorized collaboration on academic work; taking, acquiring, or using test materials without faculty permission; submitting false or incomplete records of academic grades, honors, awards, or professional endorsement; or altering, forging, or misusing a University academic record; or fabricating or falsifying of data, research procedures, or data analysis.*

References

There will be no one particular book. The following will be useful references.

- "Simulation" by Sheldon M. Ross (Academic Press, Fourth Edition), 2006, Chaps. 1-5.
- "Non-Uniform Random Variable Generation" by Luc Devroye. [Online book available]
- "Statistical Inference" by Casella and Berger.
- "Elements of Statistical Learning" by Hastie, Tibshirani, and Friedman
- "Convex Optimization" by Boyd and Vandenberghe
- "An Introduction to the Bootstrap" By Efron
- "Monte Carlo Statistical Methods" by Casella and Robert

Calendar

Week 1	Introduction to Monte Carlo
Week 1	Pseudo-ranom number generator
Week 2	Generating random variables - Discrete
Week 3	Generating random variables - Continuous
Week 4	Importance Sampling and Monte Carlo
Week 5	Maximum likelihood estimation with examples
Week 6	Gradient based optimization methods
Week 7	Gradient based optimization methods
Week 8	Least squares and optimization
Week 9	Bootstrap and crossvalidation
Week 10	MM algorithm and EM algorithm
Week 11	Stochastic optimization
Week 12	Simulated annealing
Week 13	Introduction to Bayesian methods
Week 14	Markov chain Monte Carlo

MTH511a: Pseudorandom Number Generation

Instructor: Dootika Vats

1 Pseudorandom Number Generation

The building block of computational simulation is the generation of uniform random numbers. If we can draw from $U(0,1)$, then we can draw from *most* other distributions. Thus the construction of sampling from $U(0,1)$ requires special attention.

Computers can generate numbers between $(0,1)$, which although are not exactly random (and in fact deterministic), but have the appearance of being $U(0,1)$ random variables. These draws from $U(0,1)$ are *pseudorandom* draws.

The goal in *pseudorandom* generation is to draw

$$X_1, \ldots, X_n \overset{\text{approx iid}}{\sim} U(0,1)\,.$$

so that they are as uniformly distributed as possible, and as independent as possible. We will learn about two different pseudorandom generators. These are very basic ones that are actually not really used in real life, but make our point well.

Note: After this lecture, we will always assume that all $U(0,1)$ draws are exactly iid and perfectly random. We will forget that they are infact, pseudorandom. For more on this, checkout CS744 at IITK.

1.1 Multiplicative congruential method

A common algorithm to generate a sequence $\{x_n\}$ is the *multiplicative congruential method*:

1. Set *seed* x_0, and positive integers a, m.
2. $x_t = a\, x_{t-1} \mod m$
3. Return sequence x_t/m for $t = 1, \ldots, n$.

x_t is one of $0, 1, \ldots m-1$, and so x_t/m is between $(0,1)$.

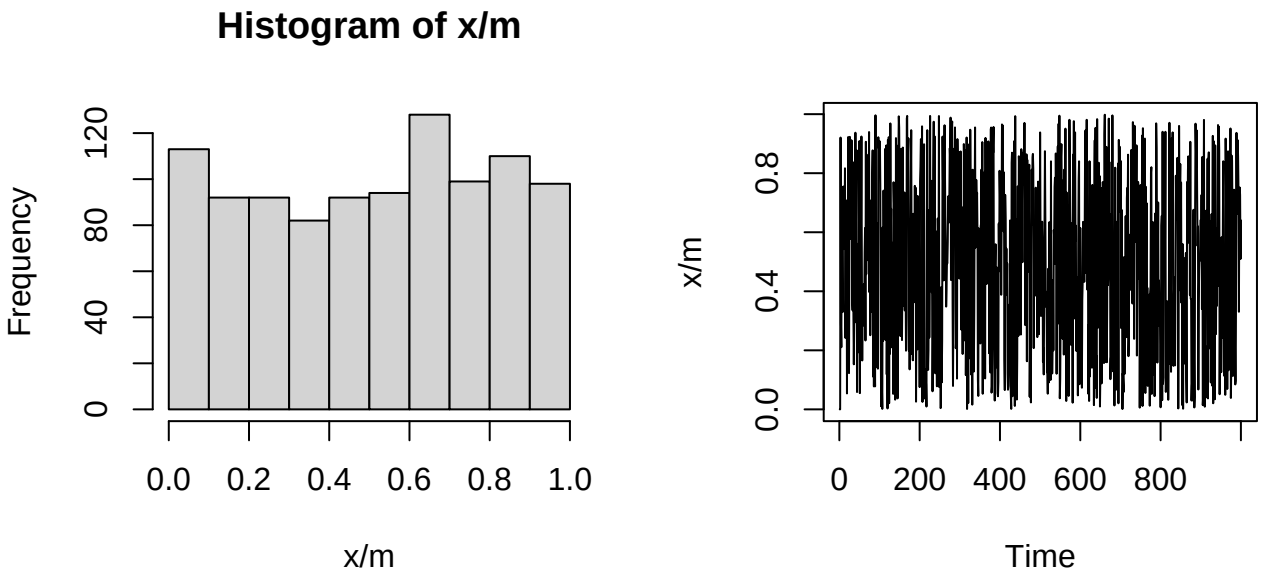
Also note that after some finite number of steps $< m$, the algorithm will repeat itself, since when a seed x_0 is set, a deterministic sequence of numbers follows.

Example 1 *Set* $a = 123$ *and* $m = 10$*, and let* $x_0 = 7$*. Then*
$x_1 = 123 * 7 \mod 10 = 1$
$x_2 = 123 * 1 \mod 10 = 3$
$x_3 = 123 * 3 \mod 10 = 9$
$x_4 = 123 * 9 \mod 10 = 7$
$x_5 = 123 * 7 \mod 10 = 1$
$\vdots$

Thus, we see that the above choices of a, m, x_0 repeats itself. Naturally, both a and m should be chosen to be large so as to avoid repetition.

Based on the bits of your machine, it is recommended to set $m = 2^{31} - 1$ and $a = 7^5$. Notice that both are large.

```
m <- 2^(31) - 1
a <- 7^5
x <- numeric(length = 1e3)
x[1] <- 7

for(i in 2:1e3)
{
  x[i] <- (a * x[i-1]) %% m
}
par(mfrow = c(1,2))
hist(x/m) # looks close to uniformly distributed
plot.ts(x/m) # look like it's jumping around too
```



Any pseudorandom generation method should satisfy:

1. for any initial seed, the resultant sequence has the "appearance" of being IID from Uniform$[0, 1]$.
2. for any initial seed, the number of values generated before repetition begins is large
3. the values can be computed efficiently.

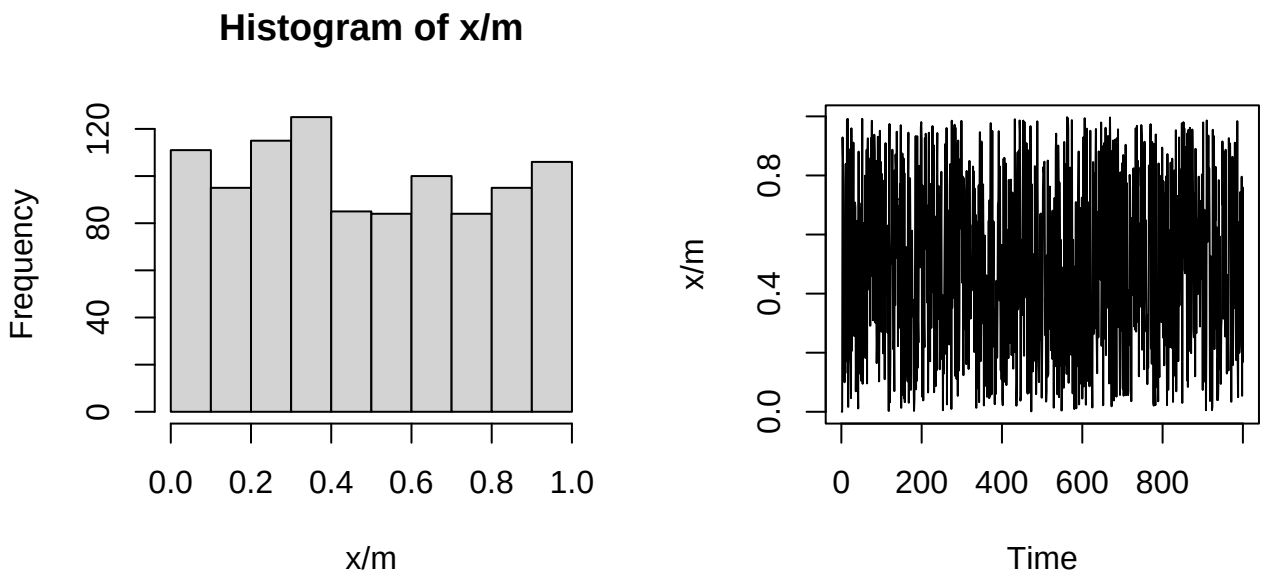
Typically m should be a large prime number

1.2 Mixed Congruential Generator

Another method is the *mixed congruential generator*:

1. Set seed x_0, and positive integers a, c, m.
2. $x_t = (a\, x_{t-1} + c) \mod m$
3. Return sequence x_t/m for $t = 1, \ldots, n$.

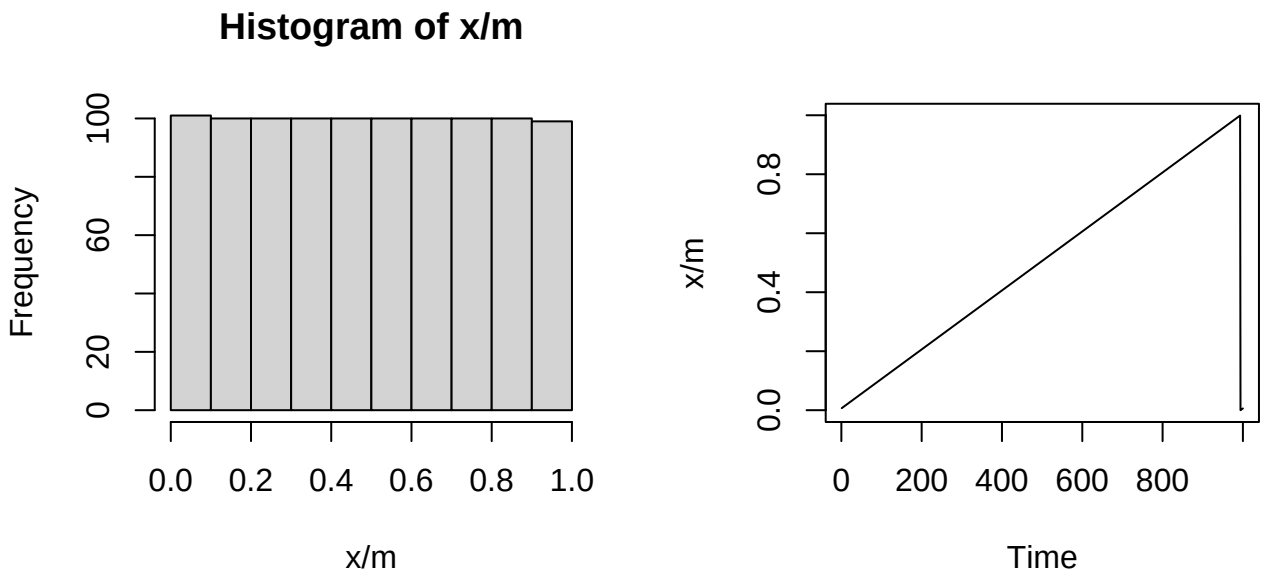
```
m <- 2^(31) - 1
a <- 7^5
c <- 2^(10) - 1
x <- numeric(length = 1e3)
x[1] <- 7

for(i in 2:1e3)
{
  x[i] <- (c + a * x[i-1]) %% m
}
par(mfrow = c(1,2))
hist(x/m) # looks close to uniformly distributed
plot.ts(x/m) # look like it's jumping around too
```



We must be cautious not to be happy with a just a histogram. A histogram shows that the empirical distribution of all samples is uniformly distributed. But we can still get a uniform looking histogram if we set $a = 1$, $m = 1e3$ and $c = 1$

```
m <- 1e3
a <- 1
```

```r
c <- 1
x <- numeric(length = 1e3)
x[1] <- 7

for(i in 2:1e3)
{
  x[i] <- (c + a * x[i-1]) %% m
}
par(mfrow = c(1,2))
hist(x/m) # looks uniformly distributed
plot.ts(x/m) # look like it's jumping around too
```



Although a histogram shows an almost perfect uniform distribution, the trace plot shows that the draws don't behave like they are independent.

We could also use

$$x_n = (a_1 x_{n-1} + a_2 x_{n-2} + \cdots + a_k x_{n-k} + c) \mod m\,,$$

but this requires more flops from the computer, and so is not as computationally viable.

We claim that these methods return "good" pseudosamples, in the sense of the three points stated above. There are statistical hypothesis tests, like the Kolmogorov-Smirnov test, one can do to test whether a sample is truly random: independent and identically distributed.

`runif()` in R uses the Mersenne-Twister generator by default (we will not go into this), but there are options to use other generators.

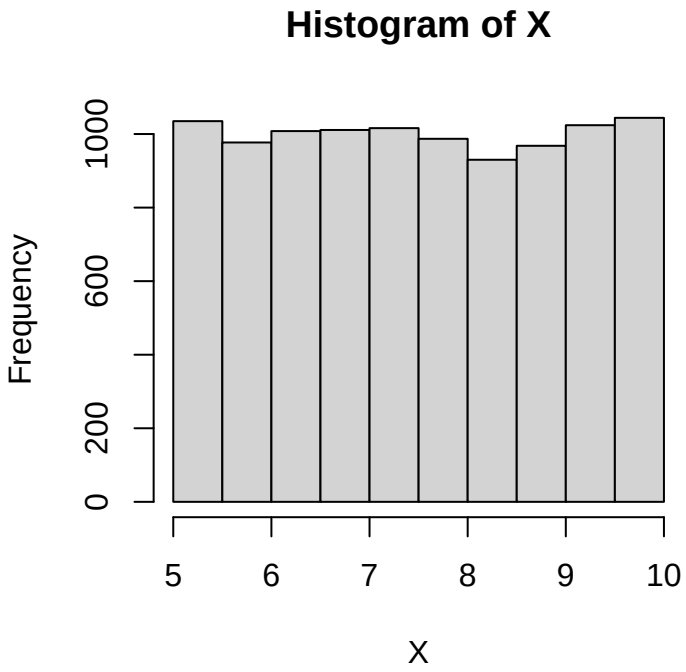
2 Generating $U(a, b)$

Suppose we can draw from $U(a, b)$ for any $a, b \in \mathbb{R}$. But we only know how to draw from $U(0, 1)$. Note that if $U \sim U(0, 1)$, then for any a, b,

$$(b-a)U + a \sim U(a,b) \quad .$$

That means, we can draw $U \sim U(0,1)$ and set $X = (b-a)U + a$. Then $X \sim U(a,b)$.

```
set.seed(1)
repeats <- 1e4
b <- 10
a <- 5
U <- runif(repeats, min = 0, max = 1)
X <- (b - a) * U + a #R is vectorized

hist(X)
```



2.1 Questions to think about

- Given a sample of pseudorandom draws from $U(0,1)$ and perfectly IID draws from $U(0,1)$, would you be able to tell the difference?

MTH 511a: L3 - Generating Discrete Random Variables

Instructor: Dootika Vats

1 Generating Discrete Random Variables

Suppose X is a discrete random variable having probability mass function

$$\Pr(X = x_j) = p_j \quad j = 0, 1, \ldots, \quad \text{so that} \sum p_j = 1\,.$$

We are interested in sampling from such reaizations of discrete random variables.

Examples of such random variables are: Bernoulli, Poisson, Geometric, Negative Binomial, Binomial, etc.

We will learn four methods to draw samples realizations of this discrete random variable

1. Inverse transform method
2. The acceptance-rejection technique
3. The composition approach
4. Bernoulli factories

1.1 Inverse transform method

The inverse transform method is the most fundamental way of generating realizations of random variables. The inverse transform method can be use to generate **any** discrete random variable.

We will demonstrate the method with an example first.

Example 1 (Bernoulli distribution). If $X \sim \text{Bern}(p)$, then

$$\Pr(X = 1) = p \text{ and } \Pr(X = 0) = q = 1 - p\,.$$

Let $U \sim U[0, 1]$. Define

$$X = \begin{cases} 0 & \text{if } U \leq q \\ 1 & \text{if } q < U \leq 1 \end{cases}.$$

Then $X \sim \text{Bern}(p)$.

Proof. To show the result we only need to show that $\Pr(X = 1) = p$ and $\Pr(X = 0) = 1-p$. Recall that by the cumulative distribution function of $U[0, 1]$, for any $0 < t < 1$m $\Pr(U < t) = t$. Using this,

$$\Pr(X = 0) = \Pr(U < q) = q\,,$$

and also

$$\Pr(X = 1) = \Pr(q \leq U < 1) = 1 - q = p\,.$$

□

Algorithm 1 Inverse transform for $\text{Bern}(p)$

1: Draw $U \sim U[0, 1]$

2: **if** $U \leq q$ **then** $X = 0$ **else** $X = 1$

Inverse transform method: The principles used in the above example can be extended to any generic discrete distribution. For a distribution with mass function

$$\Pr(X = x_j) = p_j \qquad \text{for } j = 0, 1, \ldots \qquad \text{with} \qquad \sum p_j = 1\,.$$

Let $U \sim U[0,1]$. Set X to be

$$X = \begin{cases} x_0 & \text{if } U \leq p_0 \\ x_1 & \text{if } p_0 < U \leq p_0 + p_1 \\ x_2 & \text{if } p_0 + p_1 < U \leq p_0 + p_1 + p_2 \\ \vdots & \\ x_j & \text{if } \sum_{i=0}^{j-1} p_i < U \leq \sum_{i=0}^{j} p_i \end{cases} .$$

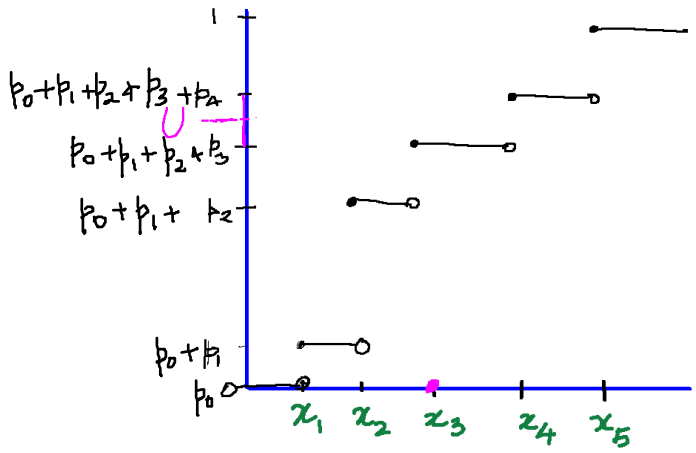
Then $\Pr(X = x_j) = p_j$.

Proof. For any generic step of the algorithm

$$\begin{aligned} \Pr(X = x_j) &= \Pr\left(\sum_{i=0}^{j-1} p_i < U \leq \sum_{i=0}^{j} p_i \right) \\ &= \Pr\left(U \leq \sum_{i=0}^{j} p_i \right) - \Pr\left(U \leq \sum_{i=0}^{j-1} p_i \right) \\ &= \sum_{i=0}^{j} p_i - \sum_{i=0}^{j-1} p_i = p_j \, . \end{aligned}$$

□

This method is called the *Inverse transform method* since the algorithm is essentially looking at the inverse cumulative distribution function of the random variable.



Example 2 (Poisson random variables). The probability mass function for the Poisson random variable is

$$\Pr(X = i) = p_i = \frac{e^{-\lambda}\lambda^i}{i!} \quad i = 0, 1, 2, \ldots,$$

Algorithm 2 Inverse transform for Poisson(λ)

1: Draw $U \sim U[0, 1]$
2: **if** $U \leq p_0$ **then**
3: $\quad X = 0$
4: **else if** $U \leq p_0 + p_1$ **then**
5: $\quad X = 1$
6: $\quad \ldots$
7: **else if** $U \leq \sum_{i=1}^{j} p_i$ **then**
8: $\quad X = j$
9: $\quad \ldots$

Algorithm 2 implements the inverse transform method for Poisson(λ). The above algorithm can be written more neatly using an iterative structure. Note that

$$\Pr(X = i + 1) = \frac{e^{-\lambda}\lambda^{i+1}}{(i+1)!} = \frac{\lambda}{i+1}\frac{e^{-\lambda}\lambda^i}{i!} = \frac{\lambda}{i+1}\Pr(X = i)\,.$$

Algorithm 3 Iterative version of inverse transformation for Poisson(λ)

1: Draw $U \sim U[0, 1]$
2: Set $i = 0, p = e^{-\lambda}, A = p$
3: **if** $U < A$ **then**
4: $\quad X = i$ and stop
5: **else**
6: $\quad$ Set $p = \frac{\lambda}{i+1}p, A = A + p$, and $i = i + 1$ goto Step 3

However, Algorithm 2 outlines a challenge in implementing this algorithm.

Q. What happens when λ is large?

We know that λ is the mean of a Poisson(λ) distribution. Thus, a Poisson(λ) distribution with a large λ will yield p_j to be small for small j. This implies Algorithm 2 can be quite slow here. We will therefore discuss a more computationally efficient algorithm.

We know that most likely, a realization from Poisson will be closer to λ, so it will be beneficial to start from around λ. Set $I = \lfloor \lambda \rfloor$, and check whether

$$\sum_{i=0}^{I-1} p_i < U \leq \sum_{i=0}^{I} p_i \,.$$

If it is, then return $X = I$. Else, if $U > \sum_{i=1}^{I} p_i$, then increase I, otherwise, decrease I and check again.

1.1.1 Questions to think about

- What other example can you think of where the inverse transform method could take a lot of time?
- Can you try and implement this for a Binomial random variable?

MTH511a: Lec 1 - Introduction to Monte Carlo

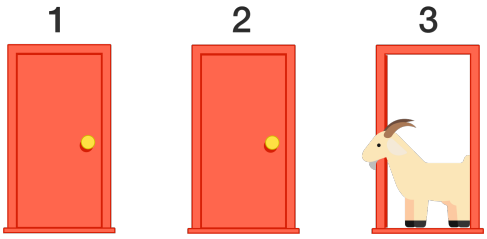
Instructor: Dootika Vats

1 Introduction to Monte Carlo

We will learn many things about Monte Carlo in this course. However, this short introduction is meant to familiarize you with the concept of using simulation to "answer" questions.

Sometimes, answers to certain mathemtical questions are not obtainable by standard/known calculations. Monte Carlo methods, or Monte Carlo experiments, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The idea is use the computer to "simulate" or "mimic" the premises of the question and then based on what the computer returns, guess the answer. This of course, sounds too vague and general, so let's work through a few examples.

Example 1: Monty Hall Problem (aka Khul Ja Sim Sim)



> You are on a game show, being asked to choose between three doors. One door has a car, and the other two have goats. After you choose a door, the host, Monty Hall, opens one of the other doors, which he knows has a goat behind it. Monty then asks whether you would like to switch your choice of door to the other remaining door. Do you choose to switch or not to switch?

Of course, whether you will switch or not depends on which action has the largest probability of winning the car (unless you like goats more than cars!). Now at first glance it seems like it would not matter whether you swtiched or not. However, this is not the case! We can answer this question mathematically, but you may not believe the answer.

Instead, let's try and simulate this situation on a computer. We will write an R code to repeat a Monty Hall experiment multiple times. And in each time, we will see whether swtiching or not swtiching would be more beneficial.

```r
# Monty Hall Calulcations
set.seed(1)
repeats <- 1e4 # We will repeat the experiment 10000 times
win.no.switch <- numeric(length = repeats) # will save 0 or 1 based on winning no switch
win.switch <- numeric(length = repeats)    # will save 0 or 1 based on winning switch

doors <- 3   # three doors
for(r in 1:repeats) # Repeat process many times
{
  # The setup
    prize <- sample(1:doors, 1)  # randomly select the door which has the prize

  # Contestants are ready. Game starts
  chosen.door <- sample(1:doors, 1)   # choose a door

  # reveal a door that is not the chosen door and not the door
  # with a prize in it. If doors let to reveal is 1
  # then no other option. If more than one door, then choose randomly
  doors.left <- (1:doors)[-c(prize, chosen.door)]
  if(length(doors.left) == 1)
  {
    reveal <- doors.left
  }else{
    reveal <- sample(doors.left, size = 1) # randomly choose which door to reveal
  }

  # Checking if you win if you didn't switch
  win.no.switch[r] <- chosen.door == prize  #tracking win if don't change door

  # What would happen if you did switch.
  # If number of doors to switch to is 1, then no other option
  # otherwise choose doors randomly
  other.doors <- (1:doors)[-c(chosen.door, reveal)]

  if(length(other.doors) == 1)
  {
    switch.door <- other.doors
  }else{
    switch.door <- sample(other.doors, size = 1)
  }

  win.switch[r] <- switch.door == prize
}
```

In the above code `win.no.switch` and `win.swtich` contains 10000 1s or 0s depending on whether the player would have won by not switching or switching respectively. To see which option is better, we can look at the mean of those two vectors:

```r
mean(win.no.switch) #Prob of winning if you don't switch
```

```
## [1] 0.3328
```

```r
mean(win.switch) # Prob of winning if you switch
```

```
## [1] 0.6672
```

Voila! The estimated probability of winning if you stay with your door is .3366, but if you switch, it is .6625! It looks like switching is more beneficial! And indeed, you can show mathematically that

$$\Pr\left(\text{Winning if you switch the door}\right) = \frac{2}{3}\,.$$

You'd think a Monte Carlo solution is not needed for this simple problem. But evidently, it was needed in the early 90s to verify the solution. There is a fascinating story to the Monty Hall problem, how it came about, and how it confused mathematicians all over the world [Vos Savant, 1997].

Example 2: Toy Collector Problem

Children (and some adults) are frequently enticed to chips packets in an effort to collect toys found in these packets. Assume there are 15 different kinds of toys and each packet contains exactly one with each toy having probability

Figure	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Probability	.2	.1	.1	.1	.1	.1	.05	.05	.05	.05	.02	.02	.02	.02	.02

Q. What is the expected number of packets needed to collect all 15 action figures.

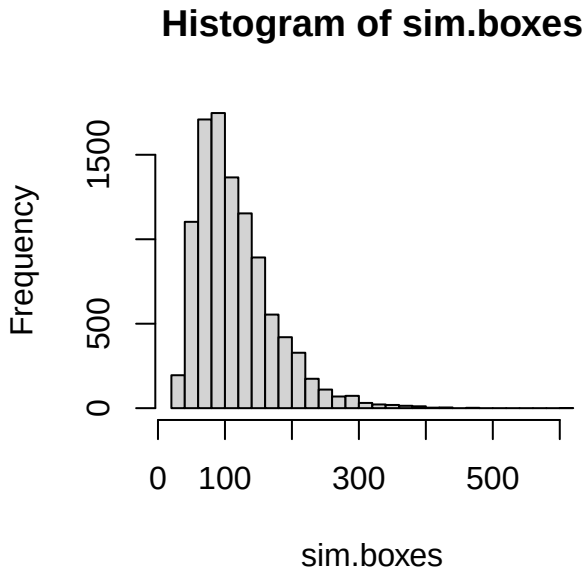
Of course, this is a question of great practical importance since it allows us to know what number of chips packets we will need to buy. Surprisingly, the solution to this problem is mathematically quite complicated. A Monte Carlo solution is often the best bet.

```r
set.seed(1)

# the setup
prob.table <- c(.2, .1, .1, .1, .1, .1, .05, .05, .05, .05, .02, .02, .02, .02, .02)
boxes <- 1:length(prob.table)

# writing the code a little differently now.
# I made a function that will be called repeatedly
box.count <- function(prob)
{
  check <- rep(0, length(prob))
  i <- 0
  while(sum(check) < length(prob)) # check if all toys collected
  {
    x <- sample(boxes, 1, prob = prob) # generate a toy with given prob
    check[x] <- 1    # x has been collected
    i <- i + 1
```

```
  }
  return(i)
}

repeats <- 1e4
sim.boxes <- numeric(repeats)
for(i in 1:repeats)
{
  sim.boxes[i] <- box.count(prob = prob.table)
}
```

Let X = number of chips packets needed to collect all toys. Then X is a random variable, and in the above code `sim.boxes` are 10000 IID realizations of this random variable. We can then, for example, look at the empirical distribution of this random variable:

```
hist(sim.boxes, breaks = 30)
```



From the looks of it X has a skewed distribution with about a 100 chips packets being sufficient most of the times, and in rare circumstances we may need 400-500 chips packets. We can estimate the expected number of chips packages by taking the average:

```
mean(sim.boxes)
```

```
## [1] 116.4749
```

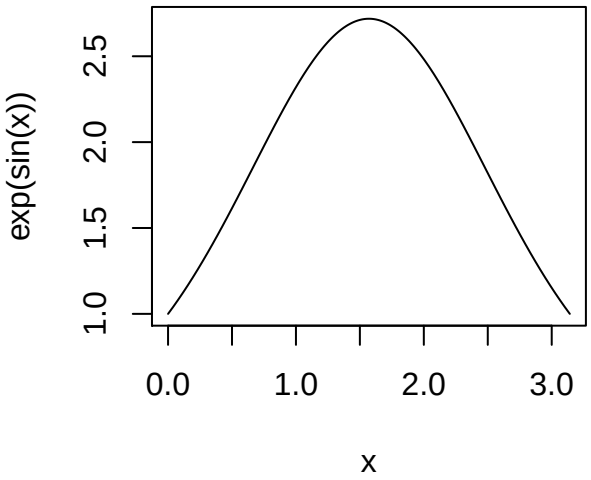
The estimated average number of chips we need to buy in order to obtain at least one of each of the toy is 116.47.

Example 3: Complicated Integral

Consider the integral

$$\theta = \int_0^{\pi} e^{\sin(x)}\, dx\,.$$

```
x <- seq(0,pi,length = 1e3) # grid on x-axis
plot(x, exp(sin(x)), type ='l')
```



The above integral does not have a standard analytical form (although a solution exists in terms of the Bessel function). But suppose we are interested in calculating θ. In Monte Carlo simulation, we take this "simple" mathematical problem with no variability and turn it into a statistical problem. That is, instead of "calculating" θ exactly, we will "estimate" θ.

Let $Y \sim U(0, \pi)$. Then Y has the following probability density function:

$$f(y) = \frac{1}{\pi} I(0 < y < \pi) \, .$$

Notice that we can multiply and divide by $f(y)$ in θ, so that

$$\theta = \int_0^\pi e^{\sin(x)} \, dx = \pi \int_0^\pi \frac{1}{\pi} e^{\sin(x)} \, dx = \pi \int_0^\pi e^{\sin(x)} f(x) dx = \pi \mathrm{E}\left[e^{\sin(x)}\right] \, ,$$

where the last expectation is with respect to $U(0, \pi)$. Thus, the integral problem is now explicitly an expectation problem. In order to estimate θ, we can generate sample from $U(0, \pi)$ repeatedly, calculate $e^{\sin(x)}$ for each draw and then take an average.

```
set.seed(1)
repeats <- 1e4

esin <- numeric(length = repeats)
for(i in 1:repeats)
{
  samp <- runif(1, min = 0, max = pi) # draw from U(0, pi)
  esin[i] <- exp(sin(samp))
}
pi * mean(esin)  #pi*E(exp(sin(x)))
```

```
## [1] 6.18
```

Thus we can conclude that $\theta \approx 6.18$. We couldn't calculate θ exactly, but we could estimate it!

2 The Statistics of Monte Carlo

In each of the three examples, there were three steps

1. identify a distribution that requires sampling

- **Monty Hall** - Bernoulli(p) where p was unknown
- **Toy Collector** - Distribution of the number of chips packets required
- **Integral** - Uniform$(0, \pi)$

2. draw random samples from some this distribution
   - **Monty Hall** - using the Monty Hall game setup
   - **Toy Collector** - by simulating the buying of chips packets
   - **Integral** - `runif` function
3. calculate mean of some particular function of interest
   - **Monty Hall** - identity function
   - **Toy Collector** - identity function
   - **Integral** - $e^{\sin(x)}$

Keeping this in mind, the statistics of Monte Carlo are then simple. Let F be a target distribution identified in Step 1 such that it is defined on some support $\mathcal{X}$ and has a density f. Let $g : \mathcal{X} \to \mathbb{R}$ be a function such that

$$\theta = \int_{\mathcal{X}} g(x) f(x) dx \,,$$

is of interest. The above assumes a continuous random variable. For discrete random variables, the distribution function F is asssumed to have a probability mass function $p(x)$ such that

$$\theta = \sum_{x_i \in \mathcal{X}} g(x_i) p(x_i)$$

is of interest.

In order to estimate θ, suppose that we have a mechanism to draw $X_1, \ldots, X_n \overset{iid}{\sim} F$. Then an estimate of θ is

$$\hat{\theta} := \frac{1}{n} \sum_{t=1}^{n} g(X_t) \,.$$

Q. Why is $\hat{\theta}$ a good estimator?

The simple answer is the weak law of large numbers, which says that as long as $\theta < \infty$

$$\hat{\theta} := \frac{1}{n} \sum_{t=1}^{n} g(X_t) \xrightarrow{p} \theta \qquad \text{as } n \to \infty \,,$$

where $\xrightarrow{p}$ denotes convergence in probability.

Recall: Weak law of large numbers. Let $X_1, \ldots, X_n$ be a sequence of iid random variables having mean $\mu < \infty$. Then for any $\epsilon > 0$,

$$\Pr\left\{ \left| \frac{X_1 + \cdots + X_n}{n} - \mu \right| > \epsilon \right\} \to 0 \text{ as } n \to \infty.$$

Q. How do we generate iid samples from complex or even simple distributions?

This will be the question that we will focus on for the first few weeks of the course. However, keep in mind these examples to motivate *why* we need to generate samples from distributions.

3 Questions to think about

1. What if θ is not finite? What happens then?
2. How should we choose n? In my simulations, I set 'repeats = 1e4'. Is that a good choice? Should I do more? Could I have done less?
3. How can we do Monte Carlo in the third example if the bounds on the integral were 0 to 2π?
4. How can we do Monte Carlo in the third example if the bounds on the integral were 0 to ∞?

References

[Vos Savant, 1997] Vos Savant, M. (1997). *The power of logical thinking: Easy lessons in the art of reasoning... and hard facts about its absence in our lives.* Macmillan.

MTH 511a: L4 - Discrete Acceptance-Rejection

Instructor: Dootika Vats

1 The Acceptance-Rejection Technique

Although we can draw from any discrete distribution using the inverse transform method, you can imagine that for distributions on countably infinite spaces (like the Poisson distribution), the inverse transform method may be very expensive. In such situations, acceptance-rejection sampling may be more reliable.

Let $\{p_j\}$ denote the pmf of the target distribution with $\Pr(X = a_j) = p_j$ and let $\{q_j\}$ denote the pmf of another distribution with $\Pr(Y = a_j) = q_j$. Suppose you can efficiently draw from $\{q_j\}$ and you want to draw from $\{p_j\}$. Let c be a constant such that

$$\frac{p_j}{q_j} \leq c \quad \text{for all } j \text{ such that } p_j > 0\,.$$

If we can find such a $\{q_j\}$ and c, then we can implement an *Acceptance-Rejection* or *Accept-Reject* sampler. The idea is to draw samples from $\{q_j\}$ and accept these samples if they seem likely to be from $\{p_j\}$.

Algorithm 1 Acceptance-Rejection sampler to draw 1 sample from $\{p_j\}$

1: Draw $U \sim U[0,1]$
2: Simulate $Y = y$ with probability mass function q_y
3: **if** $U \leq \dfrac{p_y}{cq_y}$ **then**
4: Return $X = y$ and stop
5: **else**
6: Goto step 1

Theorem 1. *The Accept-Reject method generates a random variable with probability*

$$\Pr(X = a_j) = p_j \,.$$

Further, the number of iterations needed to generate an acceptance is distributed as Geometric$(1/c)$*.*

Proof. First, we look at the second statement. We note that the number of iterations required to stop the algorithm is clearly geometrically distributed by the definition of the geometric distribution – the distribution of the number of Bernoulli trials needed to get one success (with support $1, 2, 3...$).

We will show that the probability of success is $1/c$. "Success" here is an acceptance. First, consider

$$\begin{aligned}
\Pr(Y = a_j, \text{accepted}) &= \Pr(Y = a_j)\ \Pr(\text{Accept} \mid Y = a_j) \\
&= q_j \Pr\left(U \leq \frac{p_j}{cq_j}\right) \\
&= q_j \frac{p_j}{cq_j} = \frac{p_j}{c} \,.
\end{aligned}$$

Using this we can calculate the marginal distribution of accepting is

$$\Pr(\text{accept}) = \sum_j \Pr(Y = a_j, \text{accept}) = \sum_j \frac{p_j}{c} = \frac{1}{c} \,.$$

Thus, the second statement is proved. We will now use this to show the main statement. Note that

$$\Pr(X = a_j) = \sum_{n=1}^{\infty} \Pr(a_j \text{ accepted on iteration } n)$$

$$
\begin{aligned}
&= \sum_{n=1}^{\infty} \Pr(\text{No acceptance until iteration } n-1)\ \Pr\left(Y = a_j, \text{accept}\right) \\
&= \underbrace{\sum_{n=1}^{\infty} \left(1 - \frac{1}{c}\right)^{n-1}}_{c}\ \frac{p_j}{c} \\
&= p_j\,.
\end{aligned}
$$

This completes the proof. □

Note: Since the probability of acceptance in any loop is $1/c$, the expected number of loops for one acceptance is c!

One important thing to note is that within the support $\{a_j\}$ of $\{p_j\}$, the proposal distribution must always be positive. That is, for all a_j in the support of $\{p_j\}$, $\Pr(Y = a_j) = q_j > 0$. In other words, a proposal distribution must have support *larger* than the target distribution.

Example 1 (Sampling from Binomial using AR). The binomial distribution has pmf

$$
\Pr(X = x) = \binom{n}{x}(1-p)^{n-x}p^x \quad \text{for } x = 0, 1, \ldots, n\,.
$$

We will use AR to simulate draws from Binomial(n, p).

We could use any of Poisson, negative-binomial, or geometric distributions. We choose to use the geometric distribution, but we must be a little careful.

We use the version of geometric distribution that is defined as the number of failures before the first success, so that the support of the geometric distribution has 0 in it. The pmf of the geometric distribution is

$$
\Pr(X = x) = (1-p)^x p \qquad x = 0, 1, \ldots\,.
$$

We will first find c. Note that

$$
\begin{aligned}
\frac{p(x)}{q(x)} &= \frac{\binom{n}{x}(1-p)^{n-x}p^x}{(1-p)^x p} \\
&= \binom{n}{x}(1-p)^{n-2x}p^{x-1}\,.
\end{aligned}
$$

Set

$$c = \max_{x=0,1,\ldots,n} \binom{n}{x} (1-p)^{n-2x} p^{x-1} .$$

For $n = 10, p = 0.25$, we yield $c = 2.373\ldots$.

To be safe (since I don't know all the decimal points), we can set $c = 2.5$. Now the AR algorithm can be implemented simply. Below is code for the Accept-Reject sampler.

```
##########################################
## Accept Reject algorithm to draw from
## Binomial(n,p)
##########################################
set.seed(1)
# Function draws one value from Binom(n,p)
# n = number of trials
# p = probability of success
draw_binom <- function(n, p)
{
  accept <- 0 # Will track the acceptance
  try <- 0 # Will track the number of proposals

  # upper bound calculated in the notes
  x <- 0:n
  all_c <- choose(n,x) * (1-p)^(n - 2*x) * p^(x-1)
  c <- max(all_c) + .001 # final c with slight increase for numerical
     stability.

  while(accept == 0)
  {
    try <- try + 1

    U <- runif(1)
    prop <- rgeom(1, prob = p) #draw proposal

    ratio <- dbinom(x = prop, size = n, prob = p)/
      (c* dgeom(x = prop, prob = p))
    if(U < ratio)
    {
      accept <- 1
```

```
      rtn <- prop
    }
  }
  return(c(rtn, try))
}
draw_binom(n = 10, p = .25)
# [1] 4 1

###
# If we want X1, ..., Xn ~ Binom(n.p)
# we need to call the function multiple times

N <- 1e3 # sample size
samp <- numeric(N)
n.try <- numeric(N)
for(t in 1:N)
{
   # I use as a dummy variable often
   foo <- draw_binom(n = 10, p = .25)
   samp[t] <- foo[1]
   n.try[t] <- foo[2]
}
mean(samp) #should be n*p = 2.5
# [1] 2.51

mean(n.try) # should be approx c
# [1] 2.308
```

However, when you play around with the code, and maybe increase n, you will see that the performance of the A-R sampler is no longer good, and the sampler is too slow. (The accompanying `R` demonstrates this).

This is because, as n increases, the mean of the target distribution is np which keeps increasing, but the mean of the proposal is $(1-p)/p$ which doesn't change.

A possible fix, is to consider a Geometric(p^*) proposal so that its mean is equal to np

$$np = \frac{1-p^*}{p^*} \Rightarrow p^* = \frac{1}{np+1}$$

The problem set has a few different problems on the accept-reject sampler.

2 Question to think about

- Why is c always greater than 1?
- Can we always find such a c?
- What happens when c is large or small?

MTH 511a: L6 - The Composition Method and Bernoulli Factories

Instructor: Dootika Vats

1 The composition method

We have now learned two algorithms for sampling from a discrete distribution: the inverse transform method and the accept-reject algorithm. The inverse transform method can be used for *any* distribution and the accept-reject can be efficient if used properly.

For certain special distributions, it is easier to use a *composition method* for sampling.

Suppose we have an efficient way of simulating random variables from two pmfs $\{p_j^{(1)}\}$ and $\{p_j^{(2)}\}$, and we want to simulate from

$$\Pr(X = j) = \alpha p_j^{(1)} + (1-\alpha)p_j^{(2)} \quad j \geq 0 \;\text{ where } 0 < \alpha < 1\,.$$

First you should note that the above *composition pmf* is a valid pmf since $\sum_j \Pr(X = j) = 1$. How would we sample in such a situation?

Let $X_1 \sim P^{(1)}$ and $X_2 \sim P^{(2)}$. Set

$$X = \begin{cases} X_1 & \text{with probability } \alpha \\ X_2 & \text{with probability } 1-\alpha \end{cases}.$$

Algorithm 1 Composition method

1: Draw $U \sim U[0,1]$

2: **if** $U \leq \alpha$ **then** simulate $X_1 \sim P^{(1)}$ **else** simulate X_2 and stop

Proof. Consider

$$\begin{aligned}
&\Pr(X = j) \\
&= \Pr(X_1 = j, U \leq \alpha) + \Pr(X_2 = j, \alpha \leq U \leq 1) \qquad \text{(by law of total probability)} \\
&= \Pr(X_1 = j)\Pr(U \leq \alpha) + \Pr(X_2 = j)\Pr(\alpha < U \leq 1) \qquad \text{(by independence of } U \text{ and } X_1, X_2) \\
&= \alpha p_j^{(1)} + (1-\alpha) p_j^{(2)} .
\end{aligned}$$

□

We can set this up more generally for k different distributions. In general, $F_i, i = 1, \ldots, k$ are distribution functions, and α_i are such that $0 < \alpha_i < 1$ for all i and $\sum_i \alpha_i = 1$. The composition (or mixture) distribution is

$$F(x) = \sum_{i=1}^{k} \alpha_i F_i(x) .$$

Let $X_i \sim F_i$. To simulate from the composition F, set

$$X = \begin{cases} X_1 & \text{with probability } \alpha_1 \\ X_2 & \text{with probability } \alpha_2 \\ \vdots & \\ X_k & \text{with probability } \alpha_k \end{cases} .$$

Example 1 (Zero inflated Poisson distribution). A Poisson(λ) distribution usually has a small mass at 0. But sometimes, we need a counting distribution with large mass at 0. For example, consider the random variable X being the number of COVID-19 patients tested positive every hour. Many hours of the day this number may be 0, and then this number can be quite high for some hours.

In such a case, we may use the *zero inflated Poisson distribution* (ZIP). Recall that if

$X \sim \text{Poisson}(\lambda)$

$$\Pr(X = k) = e^{-\lambda}\frac{\lambda^k}{k!} \quad k = 0, 1, \dots .$$

If $X \sim \text{ZIP}(\pi, \lambda)$

$$\Pr(X = k) = \begin{cases} \pi + (1-\pi)e^{-\lambda} & \text{if } k = 0 \\ (1-\pi)e^{-\lambda}\frac{\lambda^k}{k!} & \text{if } k = \{1, 2, \dots\} \end{cases} .$$

Note that the mean of a ZIP is $(1-\pi)\lambda < \lambda$ since more mass is given at 0.

We will use the composition method to sample from the ZIP distribution. To sample from a ZIP, first $p_j^{(1)}$ be defined as

$$\Pr(X_1 = 0) = 1 \quad \text{and} \quad \Pr(X_1 \neq 0) = 0 \,,$$

and let $X_2 \sim \text{Poisson}(\lambda)$. Define the pmf:

$$\Pr(X = k) = \pi p_k^{(1)} + (1-\pi)p_k^{(2)} \,.$$

Then $X \sim \text{ZIP}(\pi, \lambda)$. To see this, plug in $k = 0$ and $k = 1, 2, \dots$ above:

Algorithm 2 Zero inflated Poisson distribution

1: Draw $U \sim U[0, 1]$

2: **if** $U < \pi$ **then** $X = 0$ **else** simulate $X \sim \text{Poisson}(\lambda)$

Other composition or mixture distributions are also possible. Think about Zero-inflated Binomial, Zero-inflated Geometric, 2-inflated Poisson, etc.

2 Bernoulli factories

We have learned how to sample from a Bernoulli distribution. In this section, we learn some tools to draw from a Bernoulli($f(p)$) for some specific function $0 \leq f(p) \leq 1$ using only Bernoulli(p) draws.

Suppose you have $X_1, X_2, \dots \overset{iid}{\sim} \text{Bern}(p)$. Now suppose we wish to construct a Bernoulli random variable with a parameter that is a function of p, $f(p)$. That is, we want to

simulate $Y \sim \text{Bern}(f(p))$. This process is called a Bernoulli factory.

There is no universal algorithm for all $f(p)$, but we can construct one on a case by case basis.

Example 2. Suppose we can simulate $X \sim \text{Bern}(p)$. Can we simulate a Bernoulli random variable with success probability

$$f(p) = p^2(1-p)\,?$$

We are free to draw as many samples as we want from $\text{Bern}(p)$.

So if we draw three indepenent samples from $\text{Bern}(p)$ and look at the event: $\{X_1 = 1, X_2 = 1, X_3 = 0\}$. Then

$$\Pr\left(X_1 = 1, X_2 = 1, X_3 = 0\right) = \Pr(X_1 = 1)\Pr(X_2 = 1)\Pr(X_1 = 0) = p^2(1-p)\,.$$

Thus, the following algorithm returns 1 with probability $f(p) = p^2(1-p)$.

Algorithm 3 Bernoulli factory for $f(p) = p^2(1-p)$

Draw $X_1 \sim \text{Bern}(p)$
if $X_1 = 0$ **then**
    set $X = 0$, stop
Simulate $X_2 \sim \text{Bern}(p)$.
if $X_2 = 0$ **then**
    set $X = 0$, stop
Simulate $X_3 \sim \text{Bern}(p)$.
if $X_3 = 1$ **then**
    $X = 0$, stop
Set $X = 1$.

The above returns $X = 1$ with probability $p^2(1-p)$. There is another even simpler method.

Note that for $X_1, X_2, X_3 \overset{iid}{\sim} \text{Bern}(p)$. Consider $X = X_1X_2(1-X_3)$, then

$$\Pr[X_1X_2(1-X_3) = 1] = \Pr[X_1 = 1]\Pr[X_2 = 1]\Pr[1 - X_3 = 1] = p^2(1-p)\,,$$

where the decomposition is because the only way $X_1X_2(1-X_3) = 1$ is if all three terms are equal to 1.

Algorithm 4 Another Bernoulli factory for $f(p) = p^2(1-p)$

1: Draw $X_1, X_2, X_3 \overset{iid}{\sim} \text{Bern}(p)$
2: Return $X = X_1X_2(1 - X_3)$

There is yet another way: Suppose $X_1, X_2, X_3 \overset{iid}{\sim} \text{Bern}(p)$. Consider the event $X_1 + X_2 + X_3 = 2$:

$$\begin{aligned}
&\Pr(X_1 + X_2 + X_3 = 2) \\
&= \Pr(X_1 = X_2 = 1, X_3 = 0) + \Pr(X_1 = X_3 = 1, X_2 = 0) + \Pr(X_3 = X_2 = 1, X_1 = 0) \\
&= 3p^2(1-p)\,.
\end{aligned}$$

Consider an independent variable $Y \sim \text{Bern}(1/3)$

$$\Pr\left(Y = 1, X_1 + X_2 + X_3 = 2\right) = \frac{1}{3} \cdot 3p^2(1-p) = p^2(1-p)$$

Thus, we get the following algorithm:

Algorithm 5 Yet another Bernoulli factory for $f(p) = p^2(1-p)$

1: Draw $X_1, X_2, X_3 \overset{iid}{\sim} \text{Bern}(p)$ and $Y \sim \text{Bern}(1/3)$
2: If $X_1 + X_2 + X_3 = 2$ and $Y = 1$, return 1.

Similarly other polynomials of p can be considered.

Real life motivation: Of course if p and $f(p)$ are known, we can generate $\text{Bern}(f(p))$ directly. But sometimes p is unknown. For example, the Monty Hall problem!

3 Questions to think about

1. Can you construct a similar zero inflated Binomial distribution? How would you sample from it?
2. Try setting up a Bernoulli factory for $p^5(1-p)^2$.
3. I claim Algorithm 3 is better than Algorithm 5. Why?

MTH 511a - L7: The Inverse Transform Method (Continuous)

Instructor: Dootika Vats

1 Generating continuous random variables

We will discuss three methods for generating continuous random variables:

1. Inverse transform
2. The accept-reject method
3. Ratio of uniforms

1.1 Inverse transform

The principles of the inverse transform method for discrete distributions, apply similarly to continuous random variables.

Consider a random variable X with probability density function $f(x)$ so that $f(x) \geq 0$, $\int_{-\infty}^{\infty} f(x) = 1$ and distribution function is

$$F(x) = \int_{-\infty}^{x} f(x)\, dx\,.$$

The following theorem will be the foundation for the inverse transform method.

Theorem 1. *Let $U \sim U[0,1]$. For any continuous distribution F, a random variable $X = F^{-1}(U)$ has distribution F.*

Proof. Let F_X be the distribution function of $X = F^{-1}(U)$. Then,

$$\begin{aligned} F_X(x) &= \Pr(X \leq x) \\ &= \Pr(F^{-1}(U) \leq x) \\ &= \Pr(F(F^{-1}(U)) \leq F(x)) \\ &= \Pr(U \leq F(x)) \\ &= F(x)\,. \end{aligned}$$

□

Example 1. **Exponential**(1)**:** For the Exponential(1) distribution, the cdf is $F(x) = 1 - e^{-x}$. Thus,

$$F^{-1}(u) = -\log(1-u)\,.$$

To generate $X \sim \text{Exp}(1)$ we can thus use the following algorithm:

Algorithm 1 Exponential(1) Inverse transform

1: Generate $U \sim U[0,1]$

2: Set $X = -\log(1-U) \sim \text{Exp}(1)$

Example 2. **Cauchy distribution:** Cauchy distribution has pdf

$$f(x) = \frac{1}{\pi}\frac{1}{(1+x^2)}\,,$$

and

$$u = F(x) = \int_{\infty}^{x} f(y)dy = \frac{1}{\pi}\arctan(x) + \frac{1}{2}\,.$$

So, $F^{-1}(u) = \tan(\pi(u - .5)$.

Algorithm 2 Cauchy distribution

1: Generate $U \sim U[0,1]$

2: Set $X = \tan(\pi(U - .5) \sim$ Cauchy

Example 3. **Gamma distribution:** The CDF of a Gamma(n, λ) distribution is

$$F(x) = \int_0^x \frac{\lambda e^{-\lambda y}(\lambda y)^{n-1}}{\Gamma(n)} .$$

Thus, we don't know the CDF in closed form and cannot find the inverse. This is an example where the inverse transform method cannot work.

2 Questions to think about

- Can we use the inverse transform method to generate sample from a normal distribution?

MTH 511:L8-L9 - Accept-Reject (Continuous)

Instructor: Dootika Vats

1 Generating continuous random variables

1.1 Accept-reject method

Suppose we cannot generate from distribution F with pdf $f(x)$, like the Gamma distribution example in inverse transform. We can use accept-reject in a similar way as the discrete case. That is, we choose an appropriate *proposal distribution* with density $g(x)$, and accept or reject it based on certain probabilities.

Let the support of F be $\mathcal{X}$ and choose a proposal distribution G with density $g(x)$ *whose support is larger or the same as the support of* F. That is, if $\mathcal{Y}$ is the support of G then, $\mathcal{X} \subseteq \mathcal{Y}$. If we can fine c such that

$$\sup_{x \in \mathcal{X}} \frac{f(x)}{g(x)} \leq c\,,$$

then an accept-reject sampler can be implemented.

Algorithm 1 Accept-reject for continuous random variables

1: Draw $U \sim U[0,1]$

2: Draw proposal $Y \sim G$, independently

3: **if** $U \leq \dfrac{f(Y)}{c\, g(Y)}$ **then**

4: Return $X = Y$

5: **else**

6: Go to Step 1.

Theorem 1. *Algorithm 1 returns $X \sim F$.*

Proof. Consider any set B in $\mathcal{X}$. We will show that

$$\Pr(X \in B) = F(B)\,.$$

First, we consider the probability of acceptance:

$$\begin{aligned}
\Pr(\text{accept}) &= \Pr\left(U \leq \frac{f(Y)}{cg(Y)}\right) \\
&= \mathrm{E}\left[I\left(U \leq \frac{f(Y)}{cg(Y)}\right)\right] \\
&= \mathrm{E}\left[\mathrm{E}\left[I\left(U \leq \frac{f(Y)}{cg(Y)}\right) \mid Y\right]\right] \\
&= \mathrm{E}\left[\Pr\left(U \leq \frac{f(Y)}{cg(Y)} \mid Y\right)\right] \\
&= \mathrm{E}\left[\frac{f(Y)}{cg(Y)}\right] \\
&= \int_{\mathcal{Y}} \frac{f(y)}{cg(y)} g(y) dy \\
&= \frac{1}{c}\int_{\mathcal{Y}} f(y) dy \\
&= \frac{1}{c}\int_{\mathcal{X}} f(y) dy + \frac{1}{c}\int_{\mathcal{Y}/\mathcal{X}} f(y) dy \\
&= \frac{1}{c}\,.
\end{aligned}$$

Now that we have this established, consider

$$\begin{aligned}
\Pr(X \in B) &= \Pr(Y \in B \mid \text{accept}) \\
&= \frac{\Pr\left(Y \in B, U < \dfrac{f(Y)}{cg(Y)}\right)}{\Pr(\text{accept})} \\
&= c \cdot \mathrm{E}\left[\mathrm{E}\left[I\left(Y \in B, U < \frac{f(Y)}{cg(Y)}\right) \mid Y\right]\right] \\
&= c \cdot \mathrm{E}\left[I\left(Y \in B\right)\mathrm{E}\left[I\left(U < \frac{f(Y)}{cg(Y)}\right) \mid Y\right]\right] \\
&= c \cdot \mathrm{E}\left[I\left(Y \in B\right)\frac{f(Y)}{cg(Y)}\right] \\
&= c \cdot \int_B \frac{f(y)}{cg(y)} g(y) dy \\
&= \int_B f(y) \\
&= F(B)\,.
\end{aligned}$$

□

From the proof, we know that $\Pr(\textit{accept}) = 1/c$, and so, just like the discrete example, the number of attempts it takes to generate an acceptance is distributed Geometric$(1/c)$. Thus

Mean number of loops for an acceptance is $= c\,.$

1.2 Accept-reject method: intuition

At a proposed value y:

- if $f(y)$ is large but $g(y)$ is small means this value will not be proposed often and is a good value to accept for f, so higher probability of accepting it.
- if $f(y)$ is small but $g(y)$ is large, then this value will be proposed often but is unlikely for f, so accept this value less often.

We can choose any g we want as long its support is larger than other the support of f. However, some gs will be better than other gs, based on the expected number of iterations, c.

1.3 Accept-reject method: examples

Example 1. **Beta distribution:** Consider the beta distribution Beta$(4, 3)$, where

$$f(x) = \frac{\Gamma(7)}{\Gamma(4)\Gamma(3)} x^{4-1}(1-x)^{3-1} \quad 0 < x < 1; \quad .$$

Consider a uniform proposal distribution. So that $G = U(0, 1)$ and

$$g(x) = 1 \qquad \text{for } x \in (0, 1)\,.$$

Note that, $\mathcal{X} = \mathcal{Y}$ in this case. For this choice of g,

$$\sup_{x\in(0,1)} \frac{f(x)}{g(x)} = \sup_{x\in(0,1)} f(x)$$

We can show that maximum of $f(x)$ occurs at $x = 3/5$ and

$$\sup_{x\in(0,1)} \frac{f(x)}{g(x)} = \sup_{x\in(0,1)} f(x) = 60\left(\frac{3}{5}\right)^3\left(\frac{2}{5}\right)^2 = c\,.$$

Algorithm 2 Accept-reject for Beta$(4, 3)$

1: Draw $U \sim U[0, 1]$

2: Draw proposal $Y \sim U(0, 1)$

3: **if** $U \leq \dfrac{f(Y)}{c\, g(Y)}$ **then**

4: Return $X = Y$

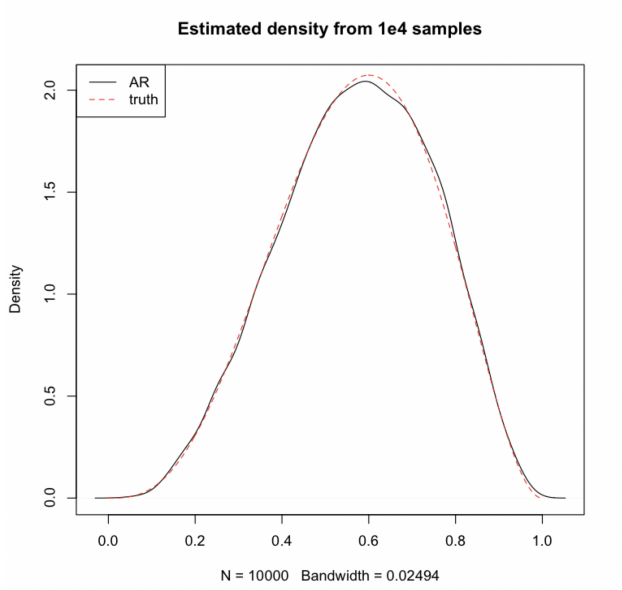
5: **else**

6: Go to Step 1.

```
#########################
### Accept-reject for
## Beta(4,3) distribution
#########################
set.seed(1)
beta_ar <- function()
{
   c <- 60 *(3/5)^3 * (2/5)^2
   accept <- 0
   counter <- 0  # count the number of loop
   while(accept == 0)
   {
      counter <- counter + 1
      U <- runif(1)
      prop <- runif(1)

      ratio <- dbeta(prop, shape1 = 4, shape2 = 3)/c

      if(U <= ratio)
      {
         accept <- 1
         return(c(prop, counter))
      }
   }
}

N <- 1e4
samp <- numeric(length = N)
```

```r
counts <- numeric(length = N)
for(i in 1:N)
{
   foo <- beta_ar() # I use foo as a dummy name
   samp[i] <- foo[1]
   counts[i] <- foo[2]
}
```

```r
x <- seq(0, 1, length = 500)
plot(density(samp), main = "Estimated density from 1e4 samples")
lines(x, dbeta(x, shape1 = 4, shape2 = 3), col = "red", lty = 2)
legend("topleft", lty = 1:2, col = c("black", "red"), legend = c("AR",
    "truth"))
```




```r
# This is c
(c <- 60 *(3/5)^3 * (2/5)^2)
# [1] 2.0736

# This is the mean number of loops required
mean(counts)
# [1] 2.0776
```

```
#They are almost the same!
```

Example 2. **Normal distribution**

What would be a good proposal distribution for $N(0,1)$. It is always good to consider distributions that have "fatter tails" than our target distribution. These may lead to more rejections, but at least we will propose values from the tails.

The target density function is

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\,.$$

We know that the t-distribution has the right support and fatter tails and the "fattest" t distribution is Cauchy. The pdf of a Cauchy distribution is

$$g(x) = \frac{1}{\pi}\frac{1}{1+x^2}\,.$$

We will need to find the supremum of the ratio of the densities. Consider

$$\frac{f(x)}{g(x)} = \frac{\pi}{\sqrt{2\pi}}(1+x^2)e^{-x^2/2}\,.$$

The supremum above occurs at $x = -1, 1$, so

$$\sup_{x\in\mathbb{R}} \frac{f(x)}{g(x)} = \frac{f(1)}{g(1)} = \sqrt{2\pi}e^{-1/2} \approx 1.746 \Rightarrow c = 1.80\,.$$

You can implement the algorithm similarly.

Example 3. **Sampling from a uniform circle** Consider a unit circle centered at $(0,0)$:

$$x^2 + y^2 < 1\,.$$

We are interested in sampling uniformly from within this circle. The target density is

$$f(x,y) = \frac{1}{\pi} I(x^2+y^2<1)\,.$$

Consider the uniform distribution on the square as a proposal distribution

$$g(x,y) = \frac{1}{4} I(-1 < x < 1) I(-1 < y < 1)\,.$$

First, we will find c. Consider

$$\frac{f(x,y)}{g(x,y)} = \frac{4}{\pi} I(x^2 + y^2 < 1) \leq \frac{4}{\pi} := c\,.$$

Next, note that

$$\frac{f(x,y)}{cg(x,y)} = \frac{4}{\pi} I(x^2 + y^2 < 1) \frac{\pi}{4} = I(x^2 + y^2 < 1)\,.$$

So for any (x, y) drawn from within the square, the ratio will be either 1 or 0, thus, no need to draw a uniform at all!

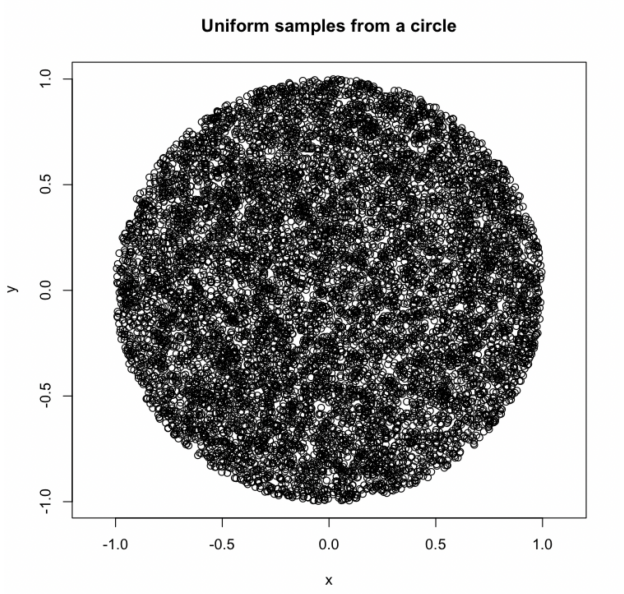
Algorithm 3 Accept-reject for Uniform distribution on a circle

1: Draw proposal $(U_1, U_2) \sim U(-1, 1) \times U(-1, 1)$
2: **if** $U_1^2 + U_2^2 \leq 1$ **then**
3: Return $(X, Y) = (U_1, U_2)$
4: **else**
5: Go to Step 1.

```
set.seed(1)
circle_ar <- function()
{
   accept <- 0
   counter <- 0  # count the number of loop
   while(accept == 0)
   {
      counter <- counter + 1
      prop <- runif(2, min = -1, max = 1) #c(U1, U2)

      in.or.out <- prop[1]^2 + prop[2]^2 < 1

      if(in.or.out)
      {
         accept <- 1
```

```
        return(c(prop, counter))
      }
   }
}

N <- 1e4
samp <- matrix(0, ncol = 2, nrow = N)
counts <- numeric(length = N)
for(i in 1:N)
{
   foo <- circle_ar() # I use foo as a dummy name
   samp[i,] <- foo[1:2]
   counts[i] <- foo[3]
}
```

```
plot(samp[,1], samp[,2], xlab = "x", ylab = "y",main = "Uniform samples
    from a circle", asp = 1)
```




```
4/pi
# [1] 1.27324
mean(counts) # very close
# [1] 1.2783
```

1.4 Questions to think about

- In A-R, do we want c to be large or small?
- Can we always find such a c?
- How can you decide whether one proposal distribution is better than another proposal distribution?
- Try implementing the circle/square example in 3 dimension, 4 dimensions, and a general p dimensions. How happens to c?
- Can a similar A-R algorithm be implemented for Beta(m, n) for all $m, n \in \mathbb{Z}$?

MTH 511a: L12 - Miscellaneous methods in sampling

Instructor: Dootika Vats

1 Miscellaneous methods in sampling

1.1 Known relationships

It is always useful to remember the relationships between different distributions.

1. **Binomial distribution**: We know that if $Y_1, Y_2, \ldots, Y_n \overset{iid}{\sim} \text{Bern}(p)$, then

$$X = Y_1 + Y_2 + \ldots Y_n \sim \text{Bin}(n, p)\,.$$

   So, we can simulate n Bernoulli variables, add them up, and we have a realization from a Binomial(n, p).

2. **Negative binomial distribution**: Number of failures until the rth success. So possibly related to geometric! If $Y_1, Y_2, \ldots, Y_r \overset{iid}{\sim} \text{Geom}(p)$ (on failures), then

$$X = Y_1 + Y_2 + \cdots + Y_r \sim NB(r, p)\,.$$

3. **Beta distribution** If $X \sim \text{Gamma}(a, 1)$ and $Y \sim \text{Gamma}(b, 1)$, then

$$\frac{X}{X+Y} \sim \text{Beta}(a, b)\,.$$

4. **Dirichlet distribution :** The Dirichlet distribution is a distribution over pmf.

$$f(x_1, x_2, \ldots, x_k) = \frac{\Gamma(\alpha_1 + \cdots + \alpha_k)}{\prod_{i=1}^k \Gamma(\alpha_i)} \prod_{i=1}^k x_i^{\alpha_i - 1} \quad 0 \le x_i \le 1, \sum_{i=1}^k x_i = 1\,.$$

The Dirichlet distribution is a generalization of the Beta distribution. Similarly,

$$\begin{aligned} Y_1 &\sim \text{Gamma}(\alpha_1, 1) \\ Y_2 &\sim \text{Gamma}(\alpha_2, 1) \\ &\vdots \\ Y_k &\sim \text{Gamma}(\alpha_k, 1) \end{aligned}$$

Let

$$X_i = \frac{Y_i}{\sum_{i=1}^k Y_i}\,.$$

Then $(X_1, \ldots, X_k) \sim \text{Dir}(\alpha_1, \alpha_2, \ldots, \alpha_k)$.

5. **Chi-squared distribution**: If $Y_1, Y_2, \ldots, Y_k \overset{iid}{\sim} N(0,1)$, then

$$X = Y_1^2 + Y_2^2 + \cdots + Y_k^2 \sim \chi_k^2\,.$$

This way we can simulate χ^2 distributions with integer degrees of freedom.

6. ***t*-distribution** Let $Z \sim N(0,1)$ and $Y \sim \chi_k^2$, then

$$X = \frac{Z}{\sqrt{\frac{Y}{k}}} \sim t_k\,.$$

7. **Location-scale family**: Let F be a distribution in the location-scale family. Then, if Z has CDF $F_Z(z)$ in the sense that it doesn't have any parameters. Then for $\mu \in R$ and $\sigma > 0$,

$$Y = \mu + \sigma Z \text{ has CDF } F_Y(y) = F_Z\left(\frac{z-\mu}{\sigma}\right)\,.$$

If Z has pdf $f(z)$ then Y has pdf $\sigma^{-1} f((z-\mu)/\sigma)$.

So, if $Z \sim N(0,1)$, then $Y = \mu + \sigma Z \sim N(\mu, \sigma^2)$.

2 Questions to think about

- How can you generate samples from an F-distribution?
- Can you generate samples from a Cauchy distribution that has mean 10 (instead of 0)?

MTH 511a: L11-L12 - Box-Muller Transformation and Ratio-of-Uniform

Instructor: Dootika Vats

1 Generating continuous random variables

1.1 The Box-Muller transformation for $N(0,1)$.

A classical method to generate samples from $N(0,1)$ is the Box-Muller transformation method. Here, we will draw random variables (R^2, Θ) from a certain distribution in the polar coordinate system, and then use a transformation h, so that $h(R^2, \Theta) \sim N(0,1)$. First, we will need some theory for this.

Let X and Y be independent and identically distributed $N(0,1)$. The joint density of (X,Y) is

$$f(x,y) = \frac{1}{2\pi} e^{-x^2/2} e^{-y^2/2}\,.$$

Let (R^2, Θ) denote the polar coordinates of (X,Y) so that $X = R\cos\Theta$ and $Y = R\sin\Theta$. Then,

$$R^2 = X^2 + Y^2 \qquad \tan\Theta = \frac{Y}{X}\,.$$

Notationally, we denote a realization from (R^2, Θ) as (d, θ) and find the joint density of $f(d,\theta)$. Thus, let $d = x^2 + y^2$ and $\theta = \tan^{-1}(y/x)$. We know that the density for

(d, θ) can be found by

$$f(d, \theta) = |J| f(x, y) \qquad \text{where } J = \begin{vmatrix} \frac{\partial x}{\partial d} & \frac{\partial y}{\partial d} \\ \frac{\partial x}{\partial \theta} & \frac{\partial y}{\partial \theta} \end{vmatrix}$$

Solving for J,

$$J = \begin{vmatrix} \frac{\partial \sqrt{d} \cos\theta}{\partial d} & \frac{\partial \sqrt{d} \sin\theta}{\partial d} \\ \frac{\partial \sqrt{d} \cos\theta}{\partial \theta} & \frac{\partial \sqrt{d} \sin\theta}{\partial \theta} \end{vmatrix} = \frac{1}{2}.$$

Since $d = x^2 + y^2$, the joint density of (R^2, Θ) is $f(d, \theta)$ with

$$\begin{aligned} f(d, \theta) &= \frac{1}{2} \frac{1}{2\pi} e^{-d/2} \quad 0 < d < \infty, 0 < \theta < 2\pi \\ &= \underbrace{\frac{1}{2\pi} I(0 < \theta < 2\pi)}_{U(0,2\pi)} \underbrace{\frac{1}{2} e^{-d/2} I(0 < d < \infty)}_{\text{Exp}(2)} \end{aligned}$$

This is a separable density, so R^2 and Θ are independent, and $\Theta \sim U[0, 2\pi]$ and $R^2 \sim \text{Exp}(2)$.

To generate from Exp(2), we can use an inverse transform method. If $U \sim U(0, 1)$, then by the inverse transform method, $-2 \log U \sim \text{Exp}(2)$ (verify for yourself). To generate from $U(0, 2\pi)$, we know if $U \sim U(0, 1)$, then $2\pi U \sim U(0, 2\pi)$. The Box-Muller algorithm then is given in Algorithm 1 which produces X and Y from $N(0, 1)$ indendently.

Algorithm 1 Box-Muller algorithm for $N(0, 1)$

1: Generate U_1 and U_2 from $U[0, 1]$ independently

2: Set $R^2 = -2 \log U_1$ and $\Theta = 2\pi U_2$

3: Set $X = R \cos(\Theta) = \sqrt{-2 \log U_1} \cos(2\pi U_2)$

4: and $Y = R \sin(\Theta) = \sqrt{-2 \log U_1} \sin(2\pi U_2)$.

1.2 Ratio-of-Uniforms

Ratio-of-uniforms is a powerful, however not so popular method to generate samples for a continuous random variables. The method is based critically on the following theorem.

Theorem 1. *Let $f(x)$ be a target density with distribution function F. Define the set*

$$C = \left\{ (u, v) : 0 \leq u \leq \sqrt{f\left(\frac{v}{u}\right)} \right\} .$$

Let (U, V) be uniformly distributed over the set C, then $V/U \sim F$.

Proof. We will show that the density of $Z = V/U$ is $f(z)$. Note that by definition, the joint density of (U, V) is

$$f_{(U,V)}(u, v) = \frac{1}{\int\int_C du\, dv} I\left\{(u, v) \in C\right\} .$$

Consider transformation $(U, V) \mapsto (U, Z)$ with $Z = V/U$. Then $U = U$ and $V = UZ$. It's easy to see that the Jacobian for this transformation is U. So

$$f_{(U,Z)}(u, z) = \frac{u}{\int\int_C du\, dv} I\left\{0 \leq u \leq f^{1/2}(z)\right\} .$$

Now that we have the joint distribution of (U, Z), all we need to show is that the marginal distribution of Z is F. Finding the marginal distribution of $Z = V/U$, we integrate out U,

$$\begin{aligned} f_Z(z) &= \int \frac{u}{\int\int_C du\, dv} I\left\{0 \leq u \leq f^{1/2}(z)\right\} du \\ &= \frac{1}{\int\int_C du\, dv} \int_0^{f^{1/2}(z)} u du \\ &= \frac{f(z)}{2\int\int_C du\, dv} . \end{aligned}$$

Since $f_Z(z)$ and $f(z)$ are both densities, this implies that

$$1 = \int f_Z(z) dz = \frac{\int f(z) dz}{2\int\int_C du\, dv} = \frac{1}{2\int\int_C du\, dv} \Rightarrow \int\int_C du\, dv = \frac{1}{2}$$

This implies $f_Z(z) = f(z)$. Thus, $Z = V/U$ has the desired distribution. $\square$

So if we can draw $(U, V) \sim \text{Unif}(C)$, then $V/U \sim F$. But C looks quite complicated, so how do we uniformly draw from C?

Think back to the AR technique used to draw uniformly from a circle! If we enclose C in a rectangle, we can use accept-reject! Find $U[0, a] \times [b, c]$ such that

$$0 \le u \le a \quad b \le v \le c \quad \text{for all } (u, v) \in C\,.$$

We just need to find any such a, b, c.

First, note that if $\sup_x f^{1/2}(x)$ exists, then

$$0 \le u \le f^{1/2}\left(\frac{v}{u}\right) \le \sup_x f^{1/2}(x) := a\,.$$

Note now that if $x = v/u \Rightarrow v/x = u \le f^{1/2}(x)$. This implies that

$$\frac{v}{x} \le f^{1/2}(x)\,.$$

For:

$$\begin{aligned} x \le 0: &\quad v \ge x f^{1/2}(x) \ge \inf_{x \le 0} x f^{1/2}(x) := b \\ x \ge 0: &\quad v \le x f^{1/2}(x) \le \sup_{x \ge 0} x f^{1/2}(x) := c\,. \end{aligned}$$

Note that if $\sqrt{f(x)}$ or $x^2 f(x)$ are unbounded, then C is unbounded, and the method cannot work.

Algorithm 2 Ratio-of-Uniforms

1: Generate $(U, V) \sim U[0, a] \times U[b, c]$
2: If $U \le \sqrt{f(V/U)}$, then set $X = V/U$.
3: Else go to 1.

Steps 1 and 2 in Algorithm 2 are implementing an Accept-Reject to sample uniformly from C. To understand how effective this algorithm will be, we can calculate the

probability of acceptance for the AR. First, note that

$$\sup_{(u,v)\in C} \frac{f(u,v)}{g(u,v)} = \sup_{(u,v)\in C} \frac{\frac{I((u,v)\in C)}{\int_C dudv}}{\frac{I((u,v)\in(0,a)\times(b,c))}{a*(c-b)}} = 2a(c-b)$$

Thus,

$$\Pr\left(\text{Accepting for AR in RoU}\right) = \frac{1}{2a(c-b)}\,.$$

So if a is large and/or $(c-b)$ is large, the probability is small.

Example 1 (Exponential(1)).

$$f(x) = e^{-x} \quad x \geq 0$$

Here,

$$C = \left\{(u,v) : 0 \leq u \leq e^{-v/2u}\right\}.$$

Recall that the set $a = \sup_x e^{-x/2} = 1$, since that is a decreasing function. Additionally,

$$b = \inf_{x\leq 0} xe^{-x/2} = 0 \quad \text{(since suppose is } x \geq 0)$$

and

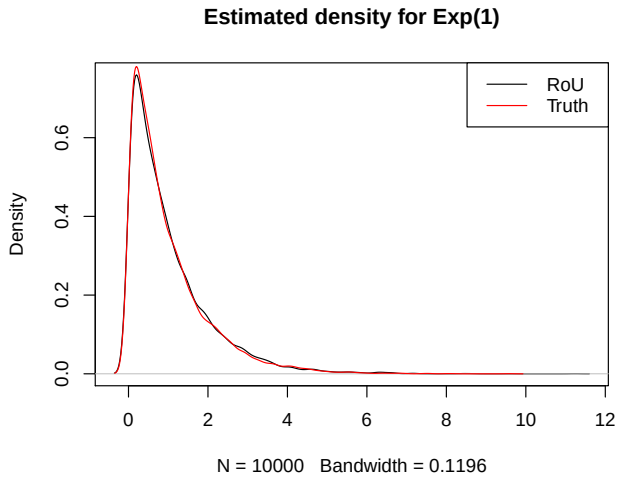
$$c = \sup_{x\geq 0} xe^{-x/2} \Rightarrow c = 2e^{-1} \quad \text{(show for yourself)}\ .$$

So we sample from $U[0,1]\times[0,2/e]$ and then implement accept-reject.

```
#####################################################
### Ratio of Uniforms for Exp(1)
#####################################################
set.seed(1)
# function to sample from the rectangle
drawFromRect <- function(a, b, c)
{
  u <- runif(1, min = 0, max = a)
  v <- runif(1, min = b, max = c)
  return(c(u,v))
}
# sqrt f function
sqrt.f <- function(x) exp(-x/2)
```

```r
# Starting the process for Exp(1)
a <- 1
b <- 0
c <- 2*exp(-1)
prob.of.acceptance <- 1/(2*a*(c-b)) # true prob. of acceptance for AR

N <- 1e4 # number of samples
samp <- numeric(length = N)
i <- 1
counter <- 0 # to check acceptance
while(i <= N)
{
  counter <- counter + 1
  prop <- drawFromRect(a = a, b = b, c = c)
  vbyu <- prop[2]/prop[1]
  if( prop[1] < sqrt.f(vbyu))
  {
    samp[i] <- vbyu
    i <- i + 1
  }
}
```

```r
plot(density(samp), main = "Estimated density for Exp(1)")
lines(density(rexp(1e4, 1)), col = "red")
legend("topright", col = c("black", "red"), lty = 1, legend = c("RoU",
   "Truth"))
```




```
(prob.of.acceptance)
# [1] 0.6795705

N/counter # very close
# [1] 0.6796248
```

Example 2 (Normal(θ, σ^2)). The target density is:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\theta)^2/2} .$$

The set C is

$$C = \left\{ (u, v) : 0 \le u \le \left(\frac{1}{2\pi\sigma^2} \right)^{1/4} e^{-(v-u\theta)^2/4\sigma^2 u^2} \right\}$$

In order to draw the region later, we need to rearrange the bound above, which gives us (by taking log):

$$(v - \theta u)^2 \le 4\sigma^2 u^2 \left(\log u + \frac{1}{2} \log(2\pi\sigma^2) \right) .$$

The above defines the region C. Now, in order to bound the region C, we find the limits a, b, c:

$$a = \sup_{x \in \mathbb{R}} (2\pi\sigma^2)^{-1/4} e^{-(x-\theta)^2/4\sigma^2} = (2\pi\sigma^2)^{-1/4}$$

$$b = \inf_{x \leq 0} \left(\frac{1}{2\pi\sigma^2} \right)^{1/4} x e^{-(x-\theta)^2/4\sigma^2} .$$

Through standard calculus and algebra it can be shown that the infimum occurs at

$$x_b := \frac{\theta - \sqrt{\theta^2 + 8\sigma^2}}{2\sigma^2}$$

Thus,

$$b = x_b f^{1/2}(x_b)$$

Similarly, and by symmetry around θ, we obtain

$$x_c := \frac{\theta + \sqrt{\theta^2 + 8\sigma^2}}{2\sigma^2}$$

with

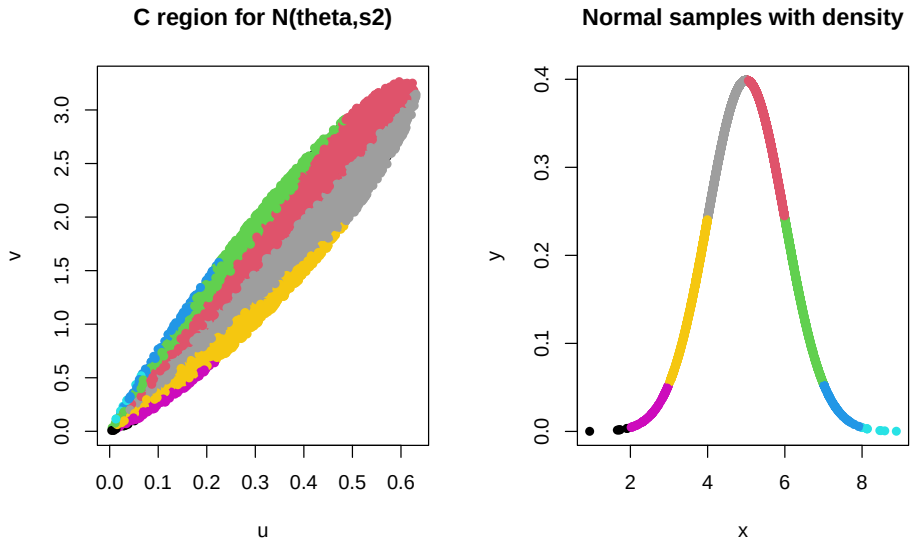
$$c = x_c f^{1/2}(x_c) .$$

All that needs to be done now is to implement Algorithm 2 with these values of a, b, c.

```
set.seed(10)

################################################
#### RoU region for N(theta, s2)
################################################

theta <- 5
s2 <- 1

a <- 1/(2*pi*s2)^(.25)
foo <- (theta - sqrt(theta^2 + 8*s2))/(2*s2)
b <- foo * sqrt(dnorm(foo, theta, sqrt(s2)))

foo <- (theta + sqrt(theta^2 + 8*s2))/(2*s2)
c <- foo * sqrt(dnorm(foo, theta, sqrt(s2)))

N <- 5e3
samples <- matrix(0, nrow = N, ncol = 2)
n <- 0
while(n < N)
```

```
{
  prop.u <- runif(1, min = 0, max = a)
  prop.v <- runif(1, min = b, max = c)
  if(abs(prop.v - theta*prop.u) <= sqrt(-4* s2* prop.u^2 *(log(prop.u) +
      log(2*pi*s2)/4) ) )
  {
    n <- n+1
    samples[n, ] <- c(prop.u,prop.v)
  }
}

x <- samples[,2]/samples[,1]
z <- (x - theta)/sqrt(s2)
# define color based on regions
color <- 0
for(i in 0:3)
{
  color <- color + (i+1)*(z > i & z < (i+1) )
}
for(i in (-1:-3) )
{
  color <- color + (i+8)*(z > (i) & z < (i+1))
}


# to plot the regions
u <- seq(0.00001, (2*pi*s2)^(-.25), length = 1e3)
foo <- sqrt(-4*s2*u^2 *(log(u) + log(2*pi*s2)/4))

par(mfrow = c(1,2))
plot(u, theta * u + foo, type = 'l', main = "C region for N(theta,s2)",
    ylim = range(c(theta * u + foo, theta * u - foo)))
lines(u,theta * u - foo )

points(samples, col = color + 1, pch = 16)


y <- dnorm(x, mean = theta, sd = sqrt(s2))
```

```
plot(x, y, col = color + 1, pch = 16, main = "Normal samples with density")
```



1.3 Questions to think about

1. Construct a similar RoU sampler for Cauchy distribution.
2. Why does RoU fail when C is unbounded?

MTH 511a: L13 - Miscellaneous methods in sampling

Instructor: Dootika Vats

1 Miscellaneous methods in sampling

1.1 Sampling from mixture distributions

Mixture distributions for continuous densities is very similar to the discrete distributions. For $j = 1, \ldots, k$, let $F_j(x)$ be a distribution function and let $f_j(x)$ be the corresponding density function. A mixture distribution function, F is

$$F(x) = \sum_{j=1}^{k} \pi_j F_j(x) \quad \text{where } \pi_j > 0, \sum_j \pi_j = 1$$

and the corresponding density is

$$f(x) = \sum_{j=1}^{k} \pi_j f_j(x)\,.$$

If we can draw from each F_j then we can draw from the mixture distribution F as well using the same steps as in the discrete case.

Algorithm 1 Sampling from a mixture distribution

1: Generate $U \sim U[0,1]$

2: If $U < \pi_1$, generate from F_1

3: If $U < \pi_1 + \pi_2$, generate from F_2

4: ...

5: If $U < \sum_{i=1}^{k-1} \pi_i$, generate from F_{k-1}

6: Else, generate from F_k .

Example 1 (Mixture of normals). Consider two normal distributions $N(\mu_1, \sigma_1^2)$ and $N(\mu_1, \sigma_2^2)$. For some $0 < p < 1$, the mixture density is

$$\begin{aligned} f(x) &= pf_1(x; \mu_1, \sigma_1^2) + (1-p)f_2(x; \mu_2, \sigma_2^2) \\ &= p\frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left\{-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma_1}\right)^2\right\} + (1-p)\frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left\{-\frac{1}{2}\left(\frac{x-\mu_2}{\sigma_2}\right)^2\right\} \end{aligned}$$

Mixture distributions are particularly useful for clustering problems and we will come back to them again in the data analysis part of the course. If we want to sample from this distribution

Algorithm 2 Sampling from a Gaussian mixture

1: Generate $U \sim U[0,1]$

2: If $U < p$, generate $N(\mu_1, \sigma_1^2)$ (using location-scale family trick)

3: Otherwise, generate $N(\mu_2, \sigma_2^2)$.

Example 2 (Zero-inflated gamma distribution). Just like the zero-inflated Poisson distribution, there are zero-inflated normal and Gamma distributions. Let's motivate the zero-inflated Gamma distribution:

Suppose you are an auto-insurance company and you want to study the cost of claims associated with each customer. That is, each customer, if they have an accident, will

come to you and claim insurance money reimbursement for the accident. So

$$\text{Let } X = \text{insurance money asked for by a customer in a month.}$$

However, most customers will not enter into any accidents, so they will claim Rs 0. But when they do, they will claim reimbursement for some amount of money that, say, will follow a Gamma distribution.

The density function can be defined as follows for $0 < p < 1$

$$f(x) = p\mathbb{I}(x = 0) + (1 - p)\frac{\beta^\alpha}{\Gamma(\alpha)}x^{\alpha-1}e^{-x\beta}\,.$$

Algorithm 3 Sampling from a zero-inflated Gamma

1: Generate $U \sim U[0, 1]$

2: If $U < p$, return $X = 0$

3: Otherwise, generate $X \sim \text{Gamma}(\alpha, \beta)$.

1.2 Multidimensional target

We have almost entirely focused on univariate densities, but most often interest is in multivariate/multidimensional target distribution.

- **Conditional Distribution:** Consider a variable $\mathbf{X} = (X_1, X_2, \ldots, X_k)$, with a joint pdf

  $$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_k)\,.$$

  We can use conditional distribution properties:

  $$f(\mathbf{x}) = f_{X_1}(x_1)f_{X_2|X_1}(x_2)\ldots f_{X_k|X_1,\ldots,X_{k-1}}(x_k)\,.$$

Algorithm 4 Sampling $\mathbf{X}$ using conditional distributions

1: Generate $X_1 \sim f_{X_1}(x_1)$
2: Generate $X_2 \sim f_{X_2|X_1}(x_2)$
3: Generate $X_3 \sim f_{X_3|X_2,X_1}(x_3)$
4: $\vdots$
5: Generate $X_n \sim f_{X_k|X_{k-1},\ldots,X_1}(x_k)$
6: Return $\mathbf{X} = (X_1, \ldots, X_k)$

- **Multivariate normal:** Consider sampling from a $N_k(\mu, \Sigma)$ where Σ is positive definite. Then for $|\cdot|$ denoting determinant,

$$f_{\mathbf{X}}(\mathbf{x}) = \left(\frac{1}{2\pi}\right)^{k/2} |\Sigma|^{-1/2} \exp\left\{-\frac{(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)}{2}\right\},$$

is the density of a multivariate normal distribution with mean μ and covariance Σ. First, note that since Σ is a positive-definite (symmetric) matrix, we can use the spectral decomposition

$$\Sigma = Q\Lambda Q^{-1}$$

where Q is the matrix of eigenvectors and since Σ is symmetric, Q is guaranteed to be an orthongal matrix so that $Q^{-1} = Q^T$ and Λ is a diagonal matrix of eigenvalues. Then, we can define the *square-root* of Σ as

$$\Sigma^{1/2} = Q\Lambda^{1/2}Q^{-1},$$

so that

$$\Sigma^{1/2}\Sigma^{1/2} = Q\Lambda^{1/2}Q^{-1}Q\Lambda^{1/2}Q^{-1} = Q\Lambda Q^{-1}.$$

Similarly, the inverse square-root is

$$\Sigma^{-1/2} = Q\Lambda^{-1/2}Q^{-1},$$

Set $\mathbf{Z} = \Sigma^{-1/2}(\mathbf{X} - \mu)$. Then

$$\mathbf{Z} \sim N_k(0, I_k).$$

That is, $\mathbf{Z}$ is a k-dimensional multivariate normal distribution with an identity covariance matrix. Which implies if $\mathbf{Z} = (Z_1, \ldots, Z_k)$, then $\mathrm{Cov}(Z_i, Z_j) = 0$ for

all $i \neq j$.

For the normal distribution, if the covariance is zero, then the random variables are independent! This isn't true in general but is true for normal random variables.

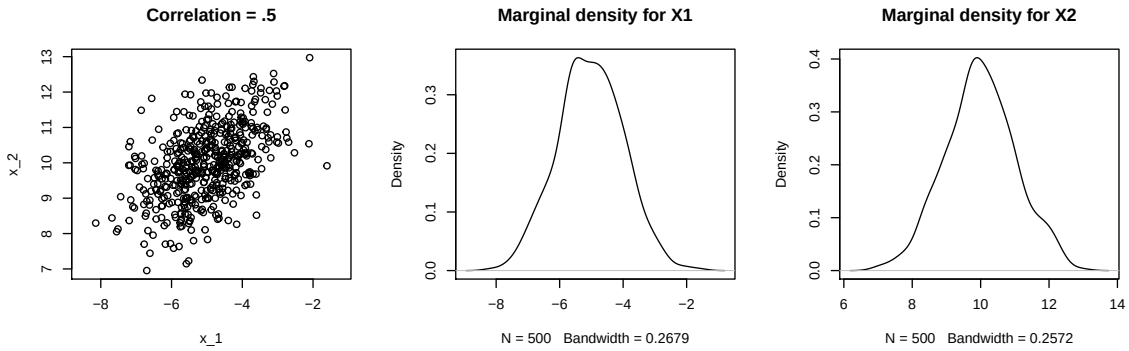
So, to sampling from $N_k(\mu, \Sigma)$, we can sample $Z_1, Z_2, \ldots, Z_k \overset{iid}{\sim} N(0,1)$, and set $\mathbf{Z} = (Z_1, \ldots, Z_k)$. Then

$$\mathbf{X} := \mu + \Sigma^{1/2}\mathbf{Z} \sim N_k(\mu, \Sigma)\,.$$

Then $\mathbf{Z} \sim N_k(\mu, \Sigma)$.

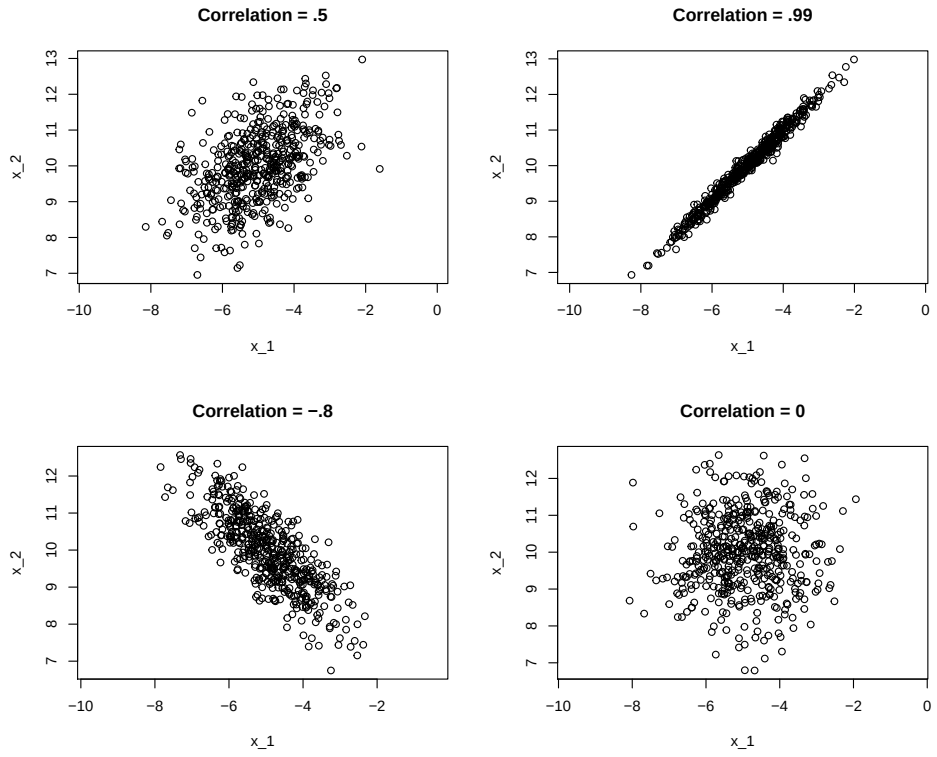
```
#####################################################################
## Generate from multivariate normal distribution
#####################################################################
set.seed(1)
par(mfrow = c(2,2))

# Function produces samples from a multivariate normal
multinorm <- function(mu, Sigma, N = 5e2)
{
  # Eigenvalue (spectral) decomposition
  decomp <- eigen(Sigma)

  # Finding matrix square-root
  Sig.sq <- decomp$vectors %*% diag(decomp$values^(1/2)) %*%
      solve(decomp$vectors)

  samp <- matrix(0, nrow = N, ncol = 2)
  for(i in 1:N)
  {
    Z <- rnorm(2)
    samp[i, ] <- mu + Sig.sq %*%Z
  }
  return(samp)
}

###
# First: Mean (-5, 10) and .5 correlation
```

```
mu <- c(-5,10)
Sigma <- matrix(c(1, .5, .5, 1), nrow = 2, ncol = 2)
samp <- multinorm(mu = mu, Sigma = Sigma)

par(mfrow = c(1,3))
plot(samp, asp = 1, main = "Correlation = .5", xlab = "x_1", ylab =
    "x_2")
plot(density(samp[,1]), main = "Marginal density for X1")
plot(density(samp[,2]), main = "Marginal density for X2")


par(mfrow = c(2,2))
plot(samp, asp = 1, main = "Correlation = .5", xlab = "x_1", ylab =
    "x_2")
```




```
###
# Second: Mean (-5, 10) and .99 correlation
mu <- c(-5,10)
Sigma <- matrix(c(1, .99, .99, 1), nrow = 2, ncol = 2)
plot(samp, asp = 1, main = "Correlation = .99", xlab = "x_1", ylab =
    "x_2")

###
# Third: Mean (-5, 10) and -.8 correlation
mu <- c(-5,10)
Sigma <- matrix(c(1, -.8, -.8, 1), nrow = 2, ncol = 2)
samp <- multinorm(mu = mu, Sigma = Sigma)
plot(samp, asp = 1, main = "Correlation = -.8", xlab = "x_1", ylab =
```

```
    "x_2")

###
# Fourth: Mean (-5, 10) and no correlation
mu <- c(-5,10)
Sigma <- matrix(c(1, 0, 0, 1), nrow = 2, ncol = 2)
samp <- multinorm(mu = mu, Sigma = Sigma)
plot(samp, asp = 1, main = "Correlation = 0", xlab = "x_1", ylab =
    "x_2")
```



2 Questions to think about

- Can you construct a zero-inflated normal distribution and find a suitable application of it?

MTH 511a - 2021: L13-L14 Importance Sampling

Instructor: Dootika Vats

We have so far learned many (many!) ways of sampling from different distributions. As motivated in the first week of the course, these sampling methodologies are useful in Monte Carlo estimation problems.

Suppose π is a distribution with density π (this is an abuse of notation that is commonly made in sampling literature). We are interested in estimating the expectation of a function $h : \mathcal{X} \to \mathbb{R}$ with respect to π. That is, we want to estimate

$$\theta := \mathrm{E}_\pi[h(X)] = \int_{\mathcal{X}} h(x)\pi(x)\, dx < \infty\,,$$

we assume that θ is finite.

Note: there is no "data" here, there is just an integral! We are just interested in estimating an annoying integral.

Note: notation $E_\pi[X]$ means the expectation is with respect to π. From now on, it is very important to keep track of what the expectation is with respect to.

Suppose we can draw iid samples $X_1, \ldots, X_N \overset{iid}{\sim} \pi(x)$ (this we can do using the many methods we have learned). Then define the estimator:

$$\hat{\theta} = \frac{1}{N}\sum_{t=1}^{N} h(X_t)\,.$$

By the law of large numbers we know that as $N \to \infty$

$$\hat{\theta} \xrightarrow{p} \theta \,.$$

In addition, we can find the variance of the estimator:

$$\begin{aligned}
\mathrm{Var}_\pi(\hat{\theta}) &= \mathrm{Var}_\pi\left(\frac{1}{N}\sum_{t=1}^{N} h(X_t)\right) \\
&= \frac{1}{N^2}\sum_{t=1}^{N} \mathrm{Var}_\pi(h(X_t)) \quad \text{because of independence} \\
&= \frac{\mathrm{Var}_\pi(h(X_1))}{N} \quad \text{because of identical} \\
&:= \frac{\sigma^2}{N} \,.
\end{aligned}$$

Naturally, a central limit theorem also holds if $\mathrm{Var}_\pi(h(X_1)) < \infty$, so that as $N \to \infty$

$$\sqrt{N}(\hat{\theta} - \theta) \xrightarrow{d} N(0, \sigma^2) \,,$$

so that we can also make large-sample confidence intervals around $\hat{\theta}$ for θ.

Q. But is there a way we can obtain a better estimator of θ?

A. Possibly by using importance sampling.

1 Importance Sampling

1.1 Basic/simple importance sampling

Let G be a distribution with density g defined on $\mathcal{X}$ so that,

$$\begin{aligned}
\mathrm{E}_\pi[h(X)] &= \int_{\mathcal{X}} h(x)\pi(x)dx \\
&= \int_{\mathcal{X}} \frac{h(x)\pi(x)}{g(x)} g(x)\, dx \\
&= \mathrm{E}_g\left[\frac{h(Z)\pi(Z)}{g(Z)}\right], \qquad Z \sim G
\end{aligned}$$

If $Z_1, \ldots, Z_N$ are iid samples from G, then an estimator of θ is

$$\hat{\theta}_g = \frac{1}{N} \sum_{t=1}^{N} \frac{h(Z_t)\pi(Z_t)}{g(Z_t)} .$$

The estimator $\hat{\theta}_g$ is the *importance sampling estimator*, the method is called *importance sampling* and G is the *importance distribution*.

Example 1 (Moments of Gamma distribution). Suppose we want to estimate the kth moment of a Gamma distribution. That is, let π be the density of a Gamma(α, β) distribution. Then

$$\theta = \int_0^\infty x^k \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} dx .$$

Suppose we set G to be also an Exponential(λ) distribution. Let $Z_1, \ldots, Z_n \sim \text{Exp}(\lambda)$

$$\begin{aligned} \hat{\theta}_g &= \frac{1}{N} \sum_{t=1}^{N} \left[\frac{h(Z_t)\pi(Z_t)}{g(Z_t)} \right] \\ &= \frac{1}{N} \sum_{t=1}^{N} \left[\frac{\beta^\alpha}{\Gamma(\alpha)} \frac{Z_t^k Z_t^{\alpha-1} e^{-\beta Z_t}}{\lambda e^{-\lambda Z_t}} \right] \end{aligned}$$

In the below simulation, I set $\alpha = 2, \beta = 5$ and set $\lambda = 3$ to estimate the $k = 2$ moment.

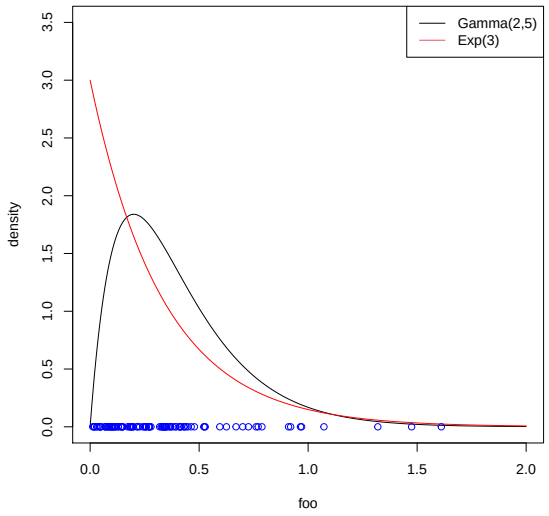
```
set.seed(1)

alpha <- 2
beta <- 5

k <- 2 # second moment
(truth <- (alpha / beta^2) + (alpha/beta)^2) # true second moment
#[1] 0.24

lambda <- 3 #proposal

N <- 1e4
samp <- rexp(N, rate = lambda) # importance samples
```

```
func <- samp^k * dgamma(samp, shape = alpha, rate = beta) / dexp(samp, rate
    = lambda)
mean(func) # truth is .24
#[1] 0.2385123
```

Our estimate is fairly close to the truth!

We can also visually compare the reference density π and the importance density g. Below, I also plot the first 100 draws from G.

```
foo <- seq(0, 2, length = 1e3)
plot(foo, dgamma(foo, shape = alpha, rate = beta),
  type = 'l', col = "black", ylab = "density", ylim = c(0, 5))
lines(foo, dexp(foo, rate = lambda), col = "red")
points(x = samp[1:100],y = rep(0, 100), col = "blue")
legend("topright", legend = c("Gamma(2,10)", "Exp(5)"), col = c(1,2), lty =
    1)
```



Theorem 1 (Unbiasedness)**.** *The importance sampling estimator $\hat{\theta}_g$ is unbiased for θ.*

Proof. To show an estimator is unbiased, we need to show that $\mathrm{E}_g[\hat{\theta}_g] = \theta$. Consider

$$\mathrm{E}_g\left[\hat{\theta}_g\right] = \mathrm{E}_g\left[\frac{1}{N}\sum_{t=1}^{N}\frac{h(Z_t)\pi(Z_t)}{g(Z_t)}\right]$$

$$
\begin{aligned}
&= \frac{1}{N}\sum_{t=1}^{N} \mathrm{E}_g\left[\frac{h(Z_t)\pi(Z_t)}{g(Z_t)}\right] \\
&= \frac{1}{N}\sum_{t=1}^{N} \mathrm{E}_g\left[\frac{h(Z_1)\pi(Z_1)}{g(Z_1)}\right] \\
&= \int_{\mathcal{X}} \frac{h(z)\pi(z)}{g(z)} g(z)\, dz \\
&= \int_{\mathcal{X}} h(z)\pi(z) dz \\
&= \theta\,.
\end{aligned}
$$

□

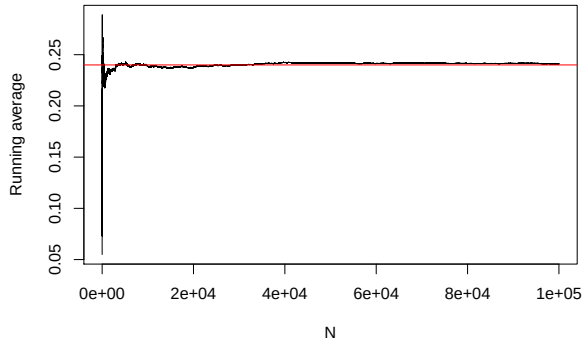
By the law of large numbers, as $N \to \infty$,

$$\hat{\theta}_g \xrightarrow{p} \mathrm{E}[\hat{\theta}_g] = \theta\,.$$

This means that as we get more and more samples from G, our estimator will get increasingly closer to the truth.

Example 2 (Gamma continued...). We can try to "verify" convergence by checking what happens as $N \to \infty$ in one simulation.

```
## Checking convergence
N <- 1e5 # very large N
samp <- rexp(N, rate = lambda) # importance samples
func <- samp^k * dgamma(samp, shape = alpha, rate = beta) / dexp(samp, rate
    = lambda)

# Plotting the running average
plot(1:N, cumsum(func)/(1:N), type = 'l', xlab = "N", ylab = "Running
    average")
abline(h = truth, col = "red")
```



We will also try to "verify" via simulation that $\hat{\theta}_g$ obtained before is indeed unbiased. By definition of unbiasedness, $\mathrm{E}_g[\hat{\theta}_g - \theta] = 0$. Thus, to mimic this in simulation, we will repeat the simulation *multiple times* (r times) so that we obtain

$$\hat{\theta}_g^1, \hat{\theta}_g^2, \ldots, \hat{\theta}_g^r \,.$$

Then by definition, if r is large:

$$\mathrm{Diff}_r = \frac{1}{r}\sum_{k=1}^{r}(\hat{\theta}_g^k - \theta) \approx \mathrm{E}_g[\hat{\theta}_g - \theta] \,.$$

Thus, if $\mathrm{Diff}_r \approx 0$, then we know that the procedure is likely unbiased.

```
## Checking if unbiased or not
N <- 1e4
r <- 1e3
ests <- numeric(length = r)
for(a in 1:r)
{
  samp <- rexp(N, rate = lambda) # importance samples
  func <- samp^k * dgamma(samp, shape = alpha, rate = beta) / dexp(samp,
      rate = lambda)
  ests[a] <- mean(func)
}
mean(ests - truth) # very close to 0
[1] 2.700126e-05
```

However, we should never be happy with a point estimator!

It is essential to quantify the variability in our estimator $\hat{\theta}_g$ in order to ascertain how "erratic" or "stable" the estimator. We also want to make confidence intervals around $\hat{\theta}_g$, but *does a central limit theorem hold?*. Note that, the variance of $\hat{\theta}_g$ is

$$\mathrm{Var}_g(\hat{\theta}_g) = \mathrm{Var}_g\left(\frac{1}{N}\sum_{t=1}^{N}\frac{h(Z_t)\pi(Z_t)}{g(Z_t)}\right) = \frac{1}{N}\mathrm{Var}_g\left(\frac{h(Z_1)\pi(Z_1)}{g(Z_1)}\right) := \frac{\sigma_g^2}{N}\,.$$

A central limit theorem will hold if $\mathrm{Var}_g\left(\dfrac{h(Z_1)\pi(Z_1)}{g(Z_1)}\right) < \infty$.

So the question is, when is this finite?

The following theorem provides a sufficient condition.

Theorem 2. *Suppose* $Var_\pi(h(X)) < \infty$. *If* g *is chosen such that*

$$\sup_{z\in\mathcal{X}}\frac{\pi(z)}{g(z)} \le M < \infty$$

then

$$\sigma_g^2 < \infty\,.$$

Proof. First note that if the second moment is finite, then the variance is finite. So, consider the second moment of $\frac{h(Z)\pi(Z)}{g(Z)}$ where $Z \sim g$.

$$\begin{aligned}
\mathrm{E}_g\left[\left(\frac{h(Z)\pi(Z)}{g(Z)}\right)^2\right] &= \int_{\mathcal{X}}\frac{h(z)^2\pi(z)^2}{g(z)^2}g(z)dz \\
&= \int_{\mathcal{X}} h(z)^2\,\frac{\pi(z)}{g(z)}\,\pi(z)dz \\
&\le M\int_{\mathcal{X}} h(z)^2\pi(z)dz \\
&= M\,\mathrm{E}_\pi(h(X)^2) < \infty \qquad \text{by assumption}\,.
\end{aligned}$$

□

Thus, if an accept-reject on the same support is possible, the variance of the importance sampling method estimator is finite. Now, we have a central limit theorem that can

hold. Recall

$$\sigma_g^2 = \mathrm{Var}_g\left(\frac{h(Z)\pi(Z)}{g(Z)}\right). \tag{1}$$

If $\sigma_g^2 < \infty$, then as $N \to \infty$,

$$\sqrt{N}(\hat{\theta}_g - \theta) \xrightarrow{d} N(0, \sigma_g^2). \tag{2}$$

Example 3 (Gamma continued). We can and "verify" again via simulation if σ_g^2 is finite. Of course, we will not be able to verify this exactly, but we can guess, what's going on.

Recall from the accept-reject example for Gamma(α, β) with $\alpha > 1$ and exponential proposal for an accept-reject sampler will work only if $\lambda < \beta$. That means, when $\lambda < \beta$, there exists a finite M.

We chose $\lambda = 3$ and $\beta = 5$, thus our running example produces finite variance estimator of θ. We can "verify" by using our replications

$$\hat{\theta}_g^1, \hat{\theta}_g^2, \ldots, \hat{\theta}_g^r$$

and noting that the sample variance of these r estimates should be σ_g^2/N. That is

$$\frac{1}{r-1}\sum_{k=1}^{r}\left(\hat{\theta}_g^k - \mathrm{mean}(\hat{\theta}_g^{1:r})\right)^2 \approx \frac{\sigma_g^2}{N}.$$

```
# looking at variance
var(ests) # This is var(theta_g) = sigma^2_g/N
# [1] 1.203824e-05

N*var(ests) # pretty small
# [1] 0.1203824
```

Now let's change things up, let's see what happens when we set $\lambda = 10$! For this setting, the accept-reject does not work, since

$$\sup_z \frac{\pi(z)}{g(z)} = \infty$$

This does not imply that $\sigma_g^2 = \infty$ since the above theorem only provides a necessary condition. Nonetheless,

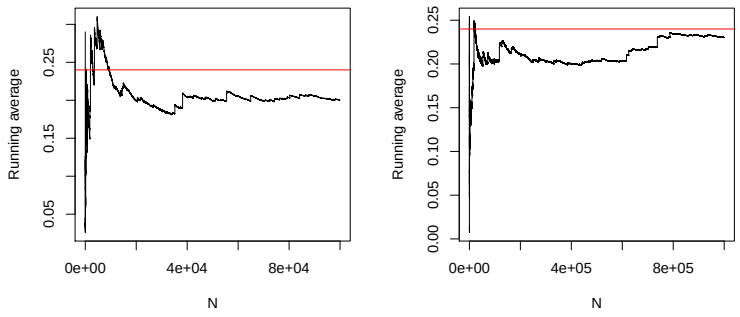
```
## When Accept-reject fails
# If lambda > beta, we know accept-reject fails, let's see what the
    variance is then
lambda <- 10
N <- 1e4
r <- 1e3
ests <- numeric(length = r)
for(a in 1:r)
{
  samp <- rexp(N, rate = lambda) # importance samples
  func <- samp^k * dgamma(samp, shape = alpha, rate = beta) / dexp(samp,
      rate = lambda)
  ests[a] <- mean(func)
}
mean(ests - truth) # close to 0
# [1] -0.01287952

var(ests) # This is var(theta_g) = sigma^2_g/N
#[1] 0.02645068

N*var(ests) # Variance is much larger now!
# [1] 264.5068
```

We can see that the variance blows up! This means that our estimator cannot be trusted from one simulation to another!

Moreover, in this example the convergence plots can be analyzed as well. Note that, to convergence in the law of large numbers, we do not require a finite variance, so convergence will still occur. However, the sample size required to get close will be much large and for finite N, residual variability will remain.

```
## Checking convergence again
# Convergence is not affected, but it takes MUCH longer
# to get good convergence
par(mfrow = c(1,2))
N <- 1e5 # very large N
samp <- rexp(N, rate = lambda) # importance samples
func <- samp^k * dgamma(samp, shape = alpha, rate = beta) / dexp(samp, rate
    = lambda)
```

```r
# Plotting the running average
plot(1:N, cumsum(func)/(1:N), type = 'l', xlab = "N", ylab = "Running
    average")
abline(h = truth, col = "red")


N <- 1e6 # very large N
samp <- rexp(N, rate = lambda) # importance samples
func <- samp^k * dgamma(samp, shape = alpha, rate = beta) / dexp(samp, rate
    = lambda)

# Plotting the running average
plot(1:N, cumsum(func)/(1:N), type = 'l', xlab = "N", ylab = "Running
    average")
abline(h = truth, col = "red")
```



2 Questions to think about

1. Check what happens with $\beta = \lambda$ in this simulation.
2. Why would a CLT be useful here?
3. How would we check whether this importance sampler is better than IID Monte Carlo?

MTH 511a 2021: Lecture 15

Instructor: Dootika Vats

1 Basic/simple importance sampling

1.1 Intuition

Recall from the last lecture that for a distribution π, and a function h, interest is in estimating

$$\theta = \int_{\mathcal{X}} h(x)\pi(x)dx\,.$$

The importance sampling estimator obtains $Z_1, \ldots, Z_N \sim G$, where G is a proposal distribution, then

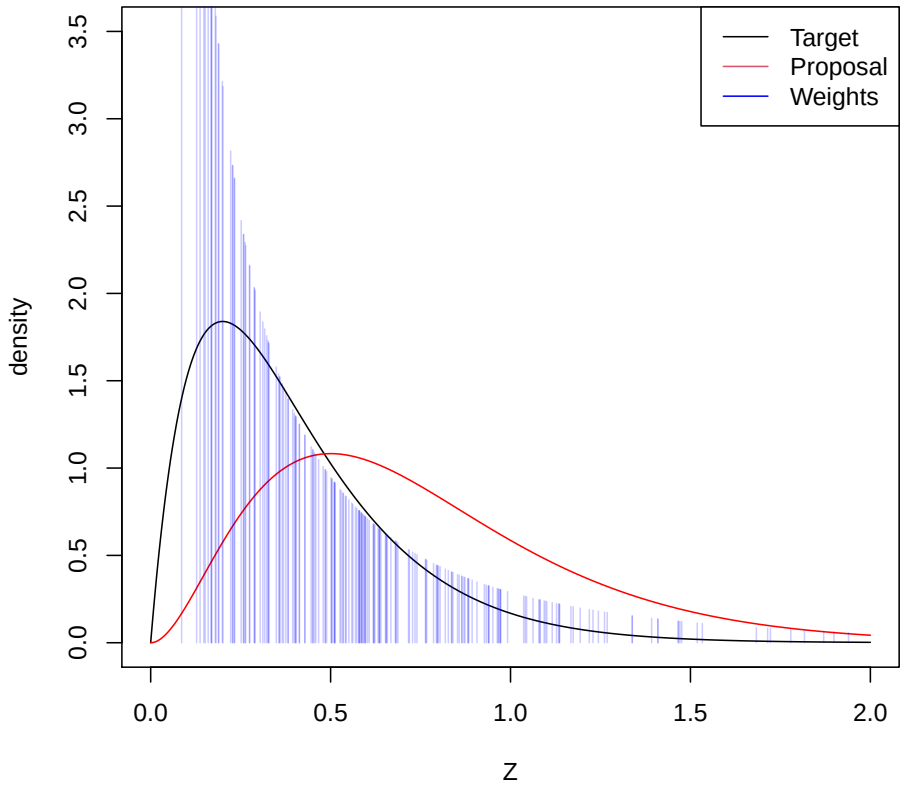
$$\hat{\theta}_g = \frac{1}{N}\sum_{t=1}^{N} \frac{h(Z_t)\pi(Z_t)}{g(Z_t)}\,.$$

Let

$$w(Z_t) = \frac{\pi(Z_t)}{g(Z_t)}$$

when w are the weights and $\hat{\theta}_g$ is a weighted average of of $h(Z_t)$.

Intuitively, this means that depending on how likely a sampled value is for π and g, a weight is assigned to that value. In the plot below, when values on the extreme left are proposed, those values are in an area of high probability for π, but unlikely to be proposed under G, thus they are assigned a large weight. Similarly, values that are likely under G but relatively less likely under π are assigned smaller weights.



1.2 Optimal proposals

How do we choose the importance distribution g? The proposal g should be chosen so that:

- Sampling from G is relatively easy
- $\text{Var}_g(\hat{\theta}_g) = \sigma_g^2/N$ is smaller than regular Monte Carlo variance estimator.

Note that, one reason to use importance sampling would be to obtain smaller variance estimators than the original. So, if we can choose g such that σ_g^2 is minimized that would be ideal!

Let's see this term:

$$\sigma_g^2 = \mathrm{Var}_g\left(\frac{h(Z)\pi(Z)}{g(Z)}\right) = \mathrm{E}_g\left[\frac{h(Z)^2\pi(Z)^2}{g(Z)^2}\right] - \theta^2 = \underbrace{\int_{\mathcal{X}} \frac{h(z)^2\pi(z)^2}{g(z)} dz}_{A} - \theta^2$$

For the above to be small, term A should be close to θ^2.

Theorem 1. *If $\int_{\mathcal{X}} |h(x)|\pi(x)dx \neq 0$, the density g^* that minimizes σ_g^2 is*

$$g^*(z) = \frac{|h(z)|\pi(z)}{E_\pi\left[|h(x)|\right]}.$$

Proof. The second moment

$$\begin{aligned}
&\theta^2 + \sigma_{g^*}^2 \\
&= \mathrm{E}_{g^*}\left[\left(\frac{h(Z)\pi(Z)}{g^*(Z)}\right)^2\right] \\
&= \int_{\mathcal{X}} \frac{h(z)^2\pi(z)^2}{g^*(z)^2} g^*(z)dz \\
&= \int_{\mathcal{X}} \frac{h(z)^2\pi(z)^2}{|h(z)|\pi(z)} \cdot \mathrm{E}_\pi\left[|h(x)|\right] dz \\
&= \mathrm{E}_\pi\left[|h(x)|\right] \int_{\mathcal{X}} |h(z)|\pi(z)dz \\
&= \left[\int_{\mathcal{X}} |h(z)|\pi(z)dz\right]^2 \\
&= \left[\int_{\mathcal{X}} \frac{|h(z)|\pi(z)}{g(z)} g(z)dz\right]^2 \quad \text{for any other } g \\
&= \left(\mathrm{E}_g\left[\frac{|h(z)|\pi(z)}{g(z)}\right]\right)^2 \\
&\leq \mathrm{E}_g\left[\frac{h(z)^2\pi(z)^2}{g^2(z)}\right] \quad \text{By Jensen's inequality: for a convex function } \phi,\ \phi(E[x]) \leq E(\phi(x)) \\
&= \theta^2 + \sigma_g^2 .
\end{aligned}$$

Thus, for any generic proposal g defined on $\mathcal{X}$, we have

$$\sigma_{g^*}^2 \leq \sigma_g^2 .$$

Since this is true for all g, this implies that g^* produces the smallest $\sigma_{g^*}^2$. □

Note that, with this choice of proposal,

$$\begin{aligned}
\sigma^2_{g^*} &= \mathrm{Var}_{g^*}\left(\frac{h(Z)\pi(Z)}{g^*(Z)}\right) \\
&= \mathrm{E}_\pi\left[|h(x)|\right]^2 \mathrm{Var}_{g^*}\left(\frac{h(Z)\pi(Z)}{|h(Z)|\pi(Z)}\right) \\
&= \mathrm{E}_\pi\left[|h(z)|\right]^2 \mathrm{Var}_{g^*}\left(\frac{h(Z)\pi(Z)}{|h(Z)|\pi(Z)}\right) \\
&= 0\,,
\end{aligned}$$

if $h(Z) > 0$ for all Z or $h(Z) < 0$ for all Z.

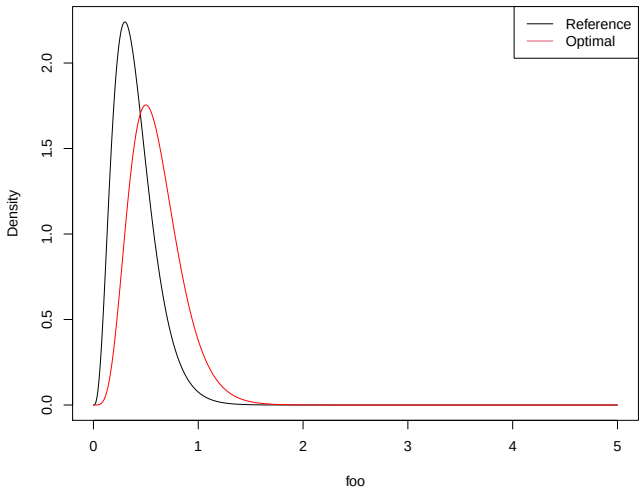
Example 1 (Gamma distribution). Consider estimating moments of a Gamma(α, β) distribution. We actually know the optimal importance distribution here! For estimating the kth moment

$$\begin{aligned}
g^*(z) &\propto |h(z)|\pi(z) \\
&= |x^k| x^{\alpha-1} \exp\left\{-\beta x\right\} \\
&= x^{\alpha+k-1} \exp\left\{-\beta x\right\}\,.
\end{aligned}$$

So the optimum importance distribution is Gamma$(\alpha+k, \beta)$. The variance in this case of the estimator will be quite close to 0.

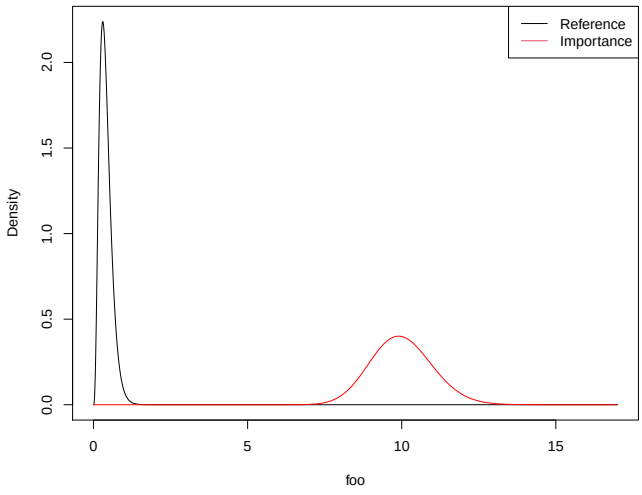
```
##########################################
### Optimal importance sampling from Gamma
##########################################
set.seed(1)

# Function does importance sampling to estimate second moment of a gamma
    distribution
imp_gamma <- function(N = 1e3, alpha = 4, beta = 10, moment = 2, imp.alpha
    = alpha + moment)
{
  fn.value <- numeric(length = N)

  draw <- rgamma(N, shape = imp.alpha, rate = beta) # draw imporance samples
  fn.value <- draw^moment * dgamma(draw, shape = alpha, rate = beta) /
      dgamma(draw, shape = imp.alpha, rate = beta)
```

```r
  return(fn.value) #return all values
}

N <- 1e4
# Estimate 2nd moment from Gamma(4, 10) using Gamma(4, 10)
# this is IID Monte Carlo
imp_samp <- imp_gamma(N = N, imp.alpha = 4)
mean(imp_samp)
# [1] 0.2002069
var(imp_samp)
# [1] 0.04421469

# Estimate 2nd moment from Gamma(4, 10) using Gamma(6, 10)
# this is the optimal proposal
imp_samp <- imp_gamma(N = N)

mean(imp_samp)
# [1] 0.2
var(imp_samp)
# [1] 9.620212e-33

# why is the estimate good
foo <- seq(0.001, 5, length = 1e3)
plot(foo, dgamma(foo, shape = 4, rate = 10), type= 'l', ylab = "Density")
lines(foo, dgamma(foo, shape = 6, rate = 10), col = "red")
legend("topright", col = 1:2, lty = 1, legend = c("Reference", "Optimal"))
```



Careful: Small variance is not all that is important! The estimate should also be close to the right value!

```
# Choosing a horrible proposal
# Estimate 2nd moment from Gamma(4, 10) using Gamma(100, 10)
imp_samp <- imp_gamma(N = N, imp.alpha = 100)
mean(imp_samp) ## estimate is horrible
# [1] 1.107169e-22
var(imp_samp)
# [1] 5.679169e-41
```



Example 2 (Mean of standard normal). Let $h(x) = x$ and let $\pi(x)$ be the density of a standard normal distribution. So we are interested in estimating the mean of the standard normal distribution. The universally optimal proposal in this case is

$$g^*(x) = \frac{|x|e^{-x^2/2}}{\int |x|e^{-x^2/2}}$$

But it may be quite challenging to draw samples from the above distribution!

In order for importance sampling to be useful, we need not find the optimal proposal, as long as we can find a *more* efficient proposal than sampling from the target.

Consider an importance distribution of $N(0, \sigma^2)$ for some $\sigma^2 > 0$. The variance of the importance estimator is

$$\begin{aligned}
\sigma_g^2 &= \int_{-\infty}^{\infty} \frac{h(x)^2\pi(x)^2}{g(x)}dx \\
&= \int_{-\infty}^{\infty} x^2 \frac{\sigma}{\sqrt{2\pi}} \exp\left\{\frac{x^2}{2\sigma^2} - x^2\right\} dx \\
&= \sigma \int_{-\infty}^{\infty} x^2 \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{x^2}{2}\left(2 - \frac{1}{\sigma^2}\right)\right\} dx \\
&= \frac{\sigma}{\sqrt{2-\sigma^{-2}}} \int_{-\infty}^{\infty} x^2 \, N(0, (2-\sigma^{-2})^{-1}) \quad \text{if } \sigma^2 > 1/2 \\
&= \frac{\sigma}{(2-\sigma^{-2})^{3/2}} \quad \text{if } \sigma^2 > 1/2
\end{aligned}$$

else variance is infinite. Also, minimizing the variance:

$$\arg \min_{\sigma > \sqrt{1/2}} \frac{\sigma}{(2-\sigma^{-2})^{3/2}} = \sqrt{2}\,.$$

Thus the optimal proposal has standard deviation $\sigma = \sqrt{2}$, not 1! Also, at $\sigma^2 = 2$, the variance is .7698 which is less than 1.

1.2.1 Questions to think about

- Does this mean that $N(0, 2)$ is the optimal proposal for estimating the mean of a standard normal?
- What is the optimal proposal within the class of *Beta* proposals for estimating the mean of a Beta distribution?

MTH 511a - 2021: L16 - Weighted Importance Sampling

Instructor: Dootika Vats

1 Importance Sampling

1.1 Weighted Importance Sampling

Often for many distributions, we do not know the target distribution fully, but only know it up to a normalizing constant. That is, the target density is

$$\pi(x) = a\tilde{\pi}(x)$$

for some unknown a and the proposal density is

$$g(x) = b\tilde{g}(x)$$

for some known or unknown b. For simplicity, we will assume that b is unknown. Suppose for some function h, the following integral is of interest:

$$\theta := \int_{\mathcal{X}} h(x)\pi(x)dx\,.$$

We still want to use g as the importance distribution, so that

$$\theta = \int_{\mathcal{X}} \frac{h(x)\pi(x)}{g(x)} g(x)\, dx\,.$$

Since a and b are unknown, we can't evaluate $\pi(x)$ and $g(x)$. So our original estimator does not work anymore! We can evaluate $\tilde{\pi}(x)$ and $\tilde{g}(x)$. So if we can estimate a and b as well, that will allow us estimate θ. Instead, we will estimate b/a, which also works!

Consider $Z_1, \ldots, Z_t \sim G$. The *weighted importance sampling* estimator of θ is

$$\hat{\theta}_w := \frac{\displaystyle\sum_{t=1}^{N} \frac{h(Z_t)\tilde{\pi}(Z_t)}{\tilde{g}(Z_t)}}{\displaystyle\sum_{t=1}^{N} \frac{\tilde{\pi}(Z_t)}{\tilde{g}(Z_t)}}\,.$$

Theorem 1. *As $N \to \infty$, $\hat{\theta}_w \xrightarrow{p} \theta$.*

Proof. As a first step, note that as $N \to \infty$, by the law of large numbers

$$\frac{1}{N}\sum_{t=1}^{N} \frac{h(Z_t)\tilde{\pi}(Z_t)}{\tilde{g}(Z_t)} \xrightarrow{p} \mathrm{E}_g\left[\frac{h(Z)\tilde{\pi}(Z)}{\tilde{g}(Z)}\right] \quad \text{and} \quad \frac{1}{N}\sum_{t=1}^{N} \frac{\tilde{\pi}(Z_t)}{\tilde{g}(Z_t)} \xrightarrow{p} \mathrm{E}_g\left[\frac{\tilde{\pi}(Z)}{\tilde{g}(Z)}\right]$$

By an application of the continuous mapping theorem, as $N \to \infty$

$$\hat{\theta}_w \xrightarrow{p} \frac{\mathrm{E}_g\left[\dfrac{h(Z)\tilde{\pi}(Z)}{\tilde{g}(Z)}\right]}{\mathrm{E}_g\left[\dfrac{\tilde{\pi}(Z)}{\tilde{g}(Z)}\right]}\,.$$

We need to show that

$$\theta = \frac{\mathrm{E}_g\left[\dfrac{h(Z)\tilde{\pi}(Z)}{\tilde{g}(Z)}\right]}{\mathrm{E}_g\left[\dfrac{\tilde{\pi}(Z)}{\tilde{g}(Z)}\right]}$$

So we will first find both expectations. First

$$\begin{aligned}\mathrm{E}_g\left[\frac{h(Z)\tilde{\pi}(Z)}{\tilde{g}(Z)}\right] &= \int_{\mathcal{X}} \frac{h(z)\tilde{\pi}(z)}{\tilde{g}(z)}\, g(z)dz \\ &= \frac{b}{a}\int_{\mathcal{X}} \frac{h(z)\pi(z)}{g(z)}\, g(z)dz\end{aligned}$$

$$= \frac{b}{a}\theta\,.$$

Second,

$$\begin{aligned}\mathrm{E}_g\left[\frac{\tilde{\pi}(Z)}{\tilde{g}(Z)}\right] &= \int_{\mathcal{X}} \frac{\tilde{\pi}(z)}{\tilde{g}(z)} g(z)dz \\ &= \frac{b}{a}\int_{\mathcal{X}} \pi(z)dz = \frac{b}{a}\,.\end{aligned}$$

So,

$$\frac{\mathrm{E}_g\left[\frac{h(Z)\tilde{\pi}(Z)}{\tilde{g}(Z)}\right]}{\mathrm{E}_g\left[\frac{\tilde{\pi}(Z)}{\tilde{g}(Z)}\right]} = \frac{\frac{b}{a}\theta}{\frac{b}{a}} = \theta\,.$$

□

We will denote

$$w(Z) = \frac{\tilde{\pi}(Z)}{\tilde{g}(Z)}\,.$$

Then $w(Z)$ is called the importance sampling weight.

However, not knowing b and a comes at a cost. Unlike the simple importance sampling estimator, the weighted importance sampling estimator $\hat{\theta}_w$ is not unbiased.

To see this heuristically, let's assume that we obtained two different samples from G for the numerator and the denominator. That is, assume that $Z_1, \ldots, Z_N \sim G$ and $T_1, \ldots, T_N \sim G$. This allows us to break the expectation as follows:

$$\begin{aligned}\mathrm{E}_g\left[\frac{\sum_{t=1}^{N} h(Z_t)w(Z_t)}{\sum_{t=1}^{N} w(T_t)}\right] &= \mathrm{E}_g\left[\sum_{t=1}^{N} h(Z_t)w(Z_t)\right]\ \mathrm{E}_g\left[\frac{1}{\sum_{t=1}^{N} w(T_t)}\right] \\ &\geq \mathrm{E}_g\left[\sum_{t=1}^{N} h(Z_t)w(Z_t)\right]\ \frac{1}{\mathrm{E}_g\left[\sum_{t=1}^{N} w(T_t)\right]} \quad \text{By Jensen's inequality} \\ &= \theta\,,\end{aligned}$$

where the equality only holds for a constant function only. Thus, in general, $\hat{\theta}_w$ overestimates the true value of θ.

Further, an exact expression for the variance is difficult to obtain. Even if the numerator and estimator are assumed to be from independent samples, we may get an approximation like:

$$\begin{aligned}
\mathrm{Var}_g\left[\hat{\theta}_w\right] &\approx \frac{\mathrm{Var}_g\left(h(Z)w(Z)\right)}{\left[\mathrm{E}_g(w(Z))\right]^2}\left(1+\frac{\mathrm{Var}_g(w(Z))}{\left[\mathrm{E}_g(w(Z))\right]^2}\right) \\
&= \frac{\mathrm{Var}_g\left(h(Z)w(Z)\right)}{b^2/a^2}\left(1+\frac{\mathrm{Var}_g(w(Z))}{\left[\mathrm{E}_g(w(Z))\right]^2}\right) \\
&= \mathrm{Var}_g\left(ah(Z)w(Z)/b\right)\left(1+\frac{\mathrm{Var}_g(w(Z))}{\left[\mathrm{E}_g(w(Z))\right]^2}\right) \\
&= \mathrm{Var}_g\left(h(Z)\frac{\pi(Z)}{g(Z)}\right)\left(1+\frac{\mathrm{Var}_g(w(Z))}{\left[\mathrm{E}_g(w(Z))\right]^2}\right) \\
&= \sigma_g^2\left(1+\frac{\mathrm{Var}_g(w(Z))}{\left[\mathrm{E}_g(w(Z))\right]^2}\right) \\
&\geq \sigma_g^2\,,
\end{aligned}$$

where recall that σ_g^2 is the variance coming from the simple importance sampling estimator. This seems to indicate that weighted importance sampling is always worse than simple importance sampling!

Note: This is actually not true. Since typically the samples from the denominator and numerator are the same, this approximation is not valid. Additionally, it is *sometimes* possible for the numerator and denominator to be negatively correlated, so that weighted importance sampling is better!

Example 1. Consider estimating

$$\theta = \int_0^\pi \int_0^\pi xy\pi(x,y)dxdy\,,$$

where

$$\pi(x,y) \propto e^{(\sin(xy))} \quad 0 \leq x, y \leq \pi$$

The target distribution is not a product of two marginals, so we have to implement multivariate importance sampling. Also, we do not know the normalizing constants. So for some unknown $a > 0$

$$\pi(x,y) = a\, e^{(\sin(xy))} \quad 0 \leq x, y \leq \pi$$

Suppose $h(x, y) = xy$. We will use a weighted importance sampling distribution. Consider the importance distribution $U[0, \pi] \times U[0, \pi]$. So that

$$g(z, t) = \frac{1}{\pi^2} I(0 < z < \pi) I(0 < t < \pi) := b \text{ (assume this is unknown)}$$

Sample $(Z_1, T_1), \ldots, (Z_N, T_N) \sim U[0, \pi] \times U[0, \pi]$. The weights are

$$w(Z_t, T_t) = \frac{\tilde{\pi}(Z_t, T_t)}{\tilde{g}(Z_t, T_t)} = e^{\sin(Z_t T_t)}.$$

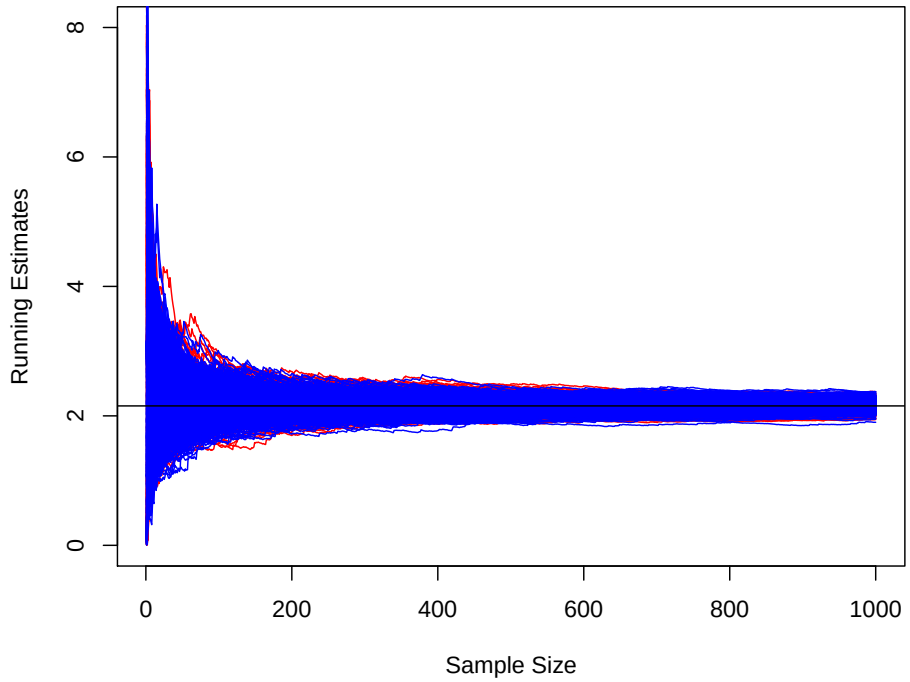
The final estimator is

$$\hat{\theta}_w = \frac{\sum_{t=1}^{N} Z_t T_t w(Z_t, T_t)}{\sum_{t=1}^{N} w(Z_t, T_t)}.$$

```
#####################################################
### xy exp( sin (xy) )
#####################################################

set.seed(1)
N <- 1e5

wis <- function(N = 1e4)
{
  samp <- runif(2*N, min = 0, max = pi)
  x <- head(samp, N)
  y <- tail(samp, N)
  weights <- exp(sin(x*y))

  return(list(weights = weights, samples = cbind(x,y)))
}

output <- wis(N = N)
est <- mean( output$samples[,1]* output$samples[,2] * output$weights) /
   mean(output$weights)
est
# [1] 2.148603
```

Let's try to visually compare simple importance sampling and weighted importance sampling in this case

```
##################################################
### Demonstrating convergence
##################################################
reps <- 5e2
N <- 1e3
ests <- numeric(length = reps)


plot(1:N, rep(est, N), type = "n", ylim = c(0, 8))
for(r in 1:reps)
{
  output <- wis(N = N)
  cumsum.num <- cumsum(output$samples[,1]*output$samples[,2]*
      output$weights)
  cumsum.den <- cumsum(output$weights)
  lines(1:N, cumsum.num/cumsum.den, col = "red")
}


## Comparing with simple importance sampling

for(r in 1:reps)
{
  samp <- runif(2*N, min = 0, max = pi)
  x <- head(samp, N)
  y <- tail(samp, N)
  fn.value <- pi^2* x*y*exp(sin(x*y))/15.5092
  # only numerator here multiplied with a*b
  cumsum.num <- cumsum(fn.value)
  lines(1:N, cumsum.num/(1:N), col = "blue")
}
abline(h = 2.15356, col = "black")


#Simple importance sampling is maybe slightly better!
```



1.2 Questions to think about

- How would you choose a good proposal for weighted importance sampling? Would finding a proposal that yields a small variance suffice?
- Do you have intuition as to why the variance of the weight importance estimator is larger than the variance of the simple importance sampler for the same importance proposal?
- Implement the above example to convince yourself that weighted importance sampling overestimates the truth and converges to the truth from below.

MTH 511a - 2021: L17: Maximum Likelihood Estimators

Instructor: Dootika Vats

We have learned a fair amount about sampling from various distributions and estimating integrals. For the next few weeks we will focus our attention to optimization methods for certain statistical procedures.

One common use of optimization in statistics is when obtaining a maximum likelihood estimator (MLE) for a parameter. Thus, we first introduce MLE below briefly, before going into optimization methods.

1 Maximum Likelihood Estimation

Suppose $X_1, X_2, \ldots, X_n$ is a random sample from a distribution with density $f(x|\theta)$ for $\theta \in \Theta$, where Θ is the parameter space. The "x given θ" implies that given a particular value of θ, $f(\cdot|\theta)$ defines a density.

The parameter θ can be a vector of parameters. After having obtained *real data*, from F, we want to

1. estimate θ
2. construct confidence intervals around the estimator of θ.

A useful method of estimating θ is the method of *maximum likelihood estimation*. Let $\mathbf{X} = (X_1, \ldots, X_n)$. The idea is that we define a function $L(\theta|\mathbf{X})$ which measures "how

likely is a particular value of θ given the data observed" and then find the θ that maximizes this likelihood.

This likelihood is defined as

$$L(\theta|\mathbf{X}) = \prod_{i=1}^{n} f(X_i|\theta)\,.$$

It is important to note that $L(\theta|\mathbf{X})$ is not a distribution over θ, it is just a function of θ. The "most likely" value is the value that maximizes the likelihood

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta\in\Theta} L(\theta|\mathbf{X})\,.$$

MTH 511a - 2021: L18: Maximum Likelihood Estimators

Instructor: Dootika Vats

We have learned a fair amount about sampling from various distributions and estimating integrals. For the next few weeks we will focus our attention to optimization methods for certain statistical procedures.

One common use of optimization in statistics is when obtaining a maximum likelihood estimator (MLE) for a parameter. Thus, we first introduce MLE below briefly, before going into optimization methods.

1 Maximum Likelihood Estimation

Suppose $X_1, X_2, \ldots, X_n$ is a random sample from a distribution with density $f(x|\theta)$ for $\theta \in \Theta$, where Θ is the parameter space. The "x given θ" implies that given a particular value of θ, $f(\cdot|\theta)$ defines a density.

The parameter θ can be a vector of parameters. After having obtained *real data*, from F, we want to

1. estimate θ
2. construct confidence intervals around the estimator of θ.

A useful method of estimating θ is the method of *maximum likelihood estimation*. Let $\mathbf{X} = (X_1, \ldots, X_n)$. The idea is that we define a function $L(\theta|\mathbf{X})$ which measures "how

likely is a particular value of θ given the data observed" and then find the θ that maximizes this likelihood.

This likelihood is defined as

$$L(\theta|\mathbf{X}) = \prod_{i=1}^{n} f(X_i|\theta)\,.$$

It is important to note that $L(\theta|\mathbf{X})$ is not a distribution over θ, it is just a function of θ. The "most likely" value is the value that maximizes the likelihood

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta\in\Theta} L(\theta|\mathbf{X})\,.$$

1.0.1 Visualizing Log-likelihoods

Consider the $\text{N}(\theta, 1)$ distribution. Suppose we obtain one data-point from this distribution and calculate the expected log-likelihood:

$$\log L(\theta|X_1) = \log f(X_1|\theta)\,.$$

Every time we obtain different data points, we will get different log-likelihood functions. Thus, the log-likelihood function is random, where the source of randomness is the data.

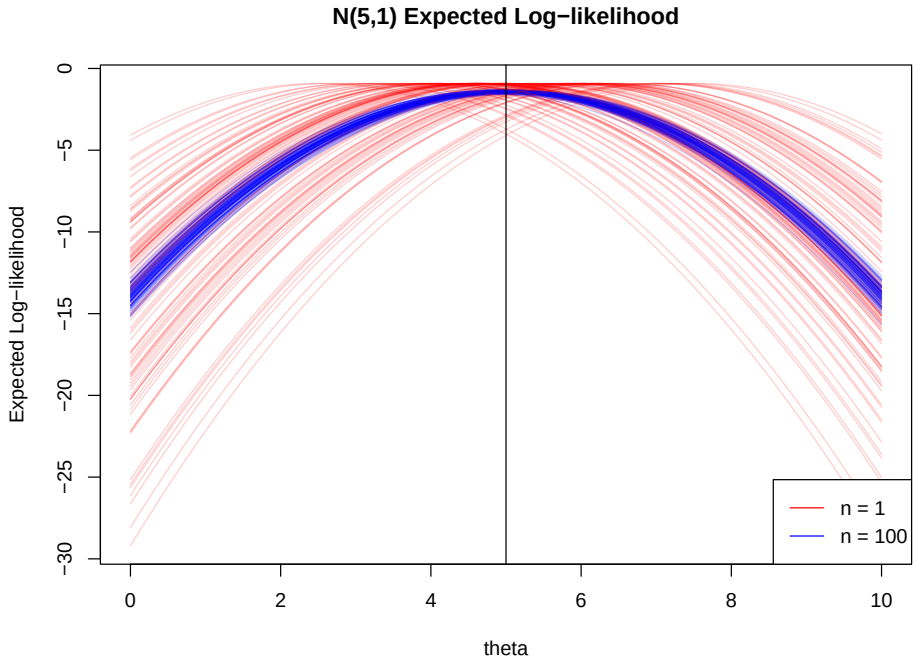
Similarly, suppose we get now $n = 100$ data points from $N(\theta, 1)$. Here again, we obtain the log-likelihood

$$\log L(\theta|X_1, \ldots, X_n) = \sum_{i=1}^{n} \log f(X_i|\theta)\,.$$

Every time we get a data-set of 100 data points, we will get a different likelihood function. Below is a visualiztion of the

$$\text{Log-likelihood}/n$$

from repeated simulations. Note that we divide by n, so make sure the scaling is the same and the plots are comparable.



1.1 Examples

Example 1 (Bernoulli). Let $X_1, \ldots, X_n \overset{iid}{\sim} \text{Bern}(p)$, $0 \leq p \leq 1$. Then the likelihood is

$$\begin{aligned} L(p|\mathbf{x}) &= \prod_{i=1}^{n} p(x_i|p) \\ &= \prod_{i=1}^{n} \left[p^{x_i}(1-p)^{1-x_i}\right] \\ &= p^{\sum x_i}(1-p)^{n-\sum x_i}. \end{aligned}$$

To obtain the MLE of θ, we will maximize the likelihood. Note that maximizing the likelihood is the same as maximizing the log of the likelihood, but the calculations are easier after taking a log. So we take a log:

$$\Rightarrow l(p) := \log L(p|\mathbf{x}) = \left(\sum_i^n x_i\right) \log p + \left(n - \sum_i^n x_i\right) \log(1-p)$$

$$\frac{dl(p)}{dp} = \frac{\sum x_i}{p} - \frac{n - \sum x_i}{1-p} \overset{\text{set}}{=} 0$$

$$\Rightarrow \hat{p} = \frac{1}{n}\sum_{t=1}^{n} x_i \,.$$

Verify for yourself that the second derivative is negative for this $\hat{p}$. Thus,

$$\hat{p}_{\text{MLE}} = \frac{1}{n}\sum_{t=1}^{n} x_i \,.$$

Example 2 (Two parameter exponential). The density of a two parameter exponential distribution is

$$f(x|\mu,\lambda) = \lambda e^{-\lambda(x-\mu)} \quad x \geq \mu, \quad \mu \in \mathbb{R}, \lambda > 0 \,.$$

We want to compute the MLEs of both λ and μ. The likelihood is

$$\begin{aligned} L(\lambda,\mu|\mathbf{x}) &= \prod_{t=1}^{n} f(x_i|\mu,\lambda) \\ &= \prod_{t=1}^{n} \lambda e^{-\lambda(x_i-\mu)}\, I(x_i \geq \mu) \\ &= \lambda^n \exp\left\{-\lambda\left(\sum_{i=1}^{n} x_i - n\mu\right)\right\} I(x_1,\ldots,x_n \geq \mu) \quad \forall \mu \text{ and } \lambda > 0 \,. \end{aligned}$$

But if $X_1,\ldots,X_n \geq \mu \Rightarrow \min\{X_i\} \geq \mu$. So

$$L(\lambda,\mu|\mathbf{x}) = \lambda^n \exp\left\{-\lambda\left(\sum_{i} x_i - n\mu\right)\right\} I\left(\min_i\{x_i\} \geq \mu\right) \quad \forall \mu \text{ and } \lambda > 0 \,.$$

We will first try to maximize with respect to μ and then with respect to λ. Note that $L(\lambda,\mu)$ is an increasing function of μ within the restriction. So that the MLE of μ is the largest value in the support of μ where $\mu \leq \min\{X_i\}$. So

$$\hat{\mu}_{\text{MLE}} = \min_{1\leq i\leq n}\{X_i\} := X_{(1)} \,.$$

Next, note that

$$\begin{aligned} L(X_{(1)},\lambda|\mathbf{x}) &= \lambda^n \exp\left\{-\lambda\left(\sum_{i} X_i - nX_{(1)}\right)\right\} \\ \Rightarrow l(X_{(1)},\lambda) := \log L(X_{(1)},\lambda|\mathbf{x}) &= n\log\lambda - \lambda\left(\sum X_i - nX_{(1)}\right) \end{aligned}$$

$$\Rightarrow \frac{d\,l}{d\,\lambda} = \frac{n}{\lambda} - \left(\sum X_i - nX_{(1)}\right) \stackrel{\text{set}}{=} 0 \quad \text{and}$$
$$\frac{d^2\,l}{d\,\lambda^2} = -\frac{n}{\lambda^2} < 0\,.$$

So, the log-likelihood function is concave, thus there is a unique maximum. Set

$$\begin{aligned} \frac{d\,l}{d\,\lambda} &= 0 \\ \Rightarrow \frac{n}{\lambda} &= \sum_{t=1}^{n} X_i - nX_{(1)} \\ \Rightarrow \hat{\lambda}_{\text{MLE}} &= \frac{n}{\sum X_i - nX_{(1)}}\,. \end{aligned}$$

Why MLE?

One main reason of using MLE is that *often* (not always), maximum likelihood estimators are consistent and asymptotically normal. That is, for a general likelihood $L(\theta|x)$ and under some regularity conditions,

$$\hat{\theta}_{\text{MLE}} \stackrel{p}{\to} \theta \text{ as } n \to \infty$$

and under some additional conditions, we also have that as $n \to \infty$

$$\sqrt{n}(\hat{\theta}_{\text{MLE}} - \theta) \stackrel{d}{\to} N(0, \Sigma^*)\,,$$

where Σ^* is an estimable matrix called the inverse Fisher information matrix. Specifically, $\Sigma^* = I(\theta)^{-1}$, where

$$I(\theta) = -\text{E}\left[\frac{\partial^2}{\partial\theta^2}\log f(X|\theta)\right]\,.$$

$I(\theta)$ essentially measures the expected curvature at the true maximum. If the second derivative is large, then the likelihood function is sharper near the truth, implying less variability in the MLE estimator.

So if we use MLE estimation (and after verifying certain important conditions), we know that we can construct confidence intervals around $\hat{\theta}_{\text{MLE}}$. This is great!

Note: The conditions required for consistency and asymptotic normality are critically important. But we will not be discussing them in this course. Please look at topics

under "Inference" for more information.

1.2 Regression

We will focus a lot on variants of linear regression. Hence, we focus on that specifically here. The following is the setup in regression.

Let $Y_1, Y_2, \ldots, Y_n$ be observations known as the *response*. Let $x_i = (x_{i1}, \ldots x_{ip})^T \in \mathbb{R}^p$ be the ith corresponding vector of covariates for the ith observation. Let $\beta \in \mathbb{R}^p$ be the *regression coefficient* so that for $\sigma^2 > 0$,

$$Y_i = x_i^T \beta + \epsilon_i \quad \text{where } \epsilon_i \sim N(0, \sigma^2)\,.$$

Let $X = (x_1^T, x_2^T, \ldots, x_n^T)^T$. In vector form we have,

$$Y = X\beta + \epsilon \sim N_n(X\beta, \sigma^2 I_n)\,.$$

Note: I use capital Y to denote the population random variable and will use the small y to denote realized observations.

The linear regression model is built to estimate β, which measures the linear effect of X on Y. There is much more to linear regression and multiple courses are required to study all aspects of it. However, here we will just focus on the mathematical properties and optimization tools required to study them.

Example 3 (MLE for Linear Regression). In order to understand the linear relationship between X and β, we will need to estimate β. Since we assumes that the errors are normally distributed, we have a distribution available for Ys and we may use the method of MLE. We have

$$\begin{aligned} L(\beta, \sigma^2|y) &= \prod_{t=1}^{n} f(y_i|X, \beta, \sigma^2) \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left\{-\frac{1}{2}\frac{(y - X\beta)^T(y - X\beta)}{\sigma^2}\right\} \\ \Rightarrow l(\beta, \sigma^2) := \log L(\beta, \sigma^2|y) &= -\frac{1}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \frac{1}{2}\frac{(y - X\beta)^T(y - X\beta)}{\sigma^2} \end{aligned}$$

Note that

$$\begin{aligned}(y - X\beta)^T(y - X\beta) &= (y^T - \beta^T X^T)(y - X\beta) \\ &= y^T y - y^T X\beta - \beta^T X^T y + \beta^T X^T X\beta \\ &= y^T y - 2\beta^T X^T y + \beta^T X^T X\beta \,.\end{aligned}$$

Using this we have (recall your multivariable calculus courses)

$$\begin{aligned}\frac{dl}{d\beta} &= -\frac{1}{2\sigma^2}\left[-2X^T y + 2X^T X\beta\right] = \frac{X^T y - X^T X\beta}{2\sigma^2} \overset{set}{=} 0 \\ \frac{dl}{d\sigma^2} &= -\frac{n}{2\sigma^2} + \frac{(y - X\beta)^T(y - X\beta)}{2\sigma^4} \overset{set}{=} 0 \,.\end{aligned}$$

The first equation leads to $\hat{\beta}_{\text{MLE}}$ satisfying

$$X^T y - X^T X\hat{\beta}_{\text{MLE}} = 0 \Rightarrow \hat{\beta}_{\text{MLE}} = (X^T X)^{-1} X^T y \,,$$

if $(X^T X)^{-1}$ exists. And $\hat{\sigma}^2_{MLE}$ is

$$\hat{\sigma}^2_{MLE} = \frac{(y - X\hat{\beta}_{\text{MLE}})^T(y - X\hat{\beta}_{\text{MLE}})}{n} \,.$$

Verify: that the Hessian matrix is negative definite, and this is indeed the maximum.

Note: What if $(X^T X)^{-1}$ does not exist?

For example, if $p > n$, then the number of observations is less than the number of parameters, and since X is $n \times p$, $(X^T X)$ is $p \times p$ of rank $n < p$. So $X^T X$ is not full rank and cannot be inverted. In this case, the MLE does not exist and other estimators need to be constructed. This is one of the motivations of *penalized regression*, which we will discuss in detail.

1.3 Penalized Regression

Note that in the Linear regression setup, the MLE for β satisfied:

$$\hat{\beta}_{\text{MLE}} = \arg\min_{\beta} (y - X\beta)^T(y - X\beta)$$

Suppose X is such that (X^TX) is not invertible, then we don't know how to estimate β. In such cases, we may used *penalized likelihood*, that penalizes the coefficients β so that some of the βs are "pushed towards zero". The corresponding Xs to those small βs are essentially not important, removing singularity from X^TX.

Instead of looking at the likelihood, we consider a penalized likelihood. Since the optimization of $L(\beta|y)$ only depends on $(y-X\beta)^T(y-X\beta)$ term, a penalized (negative) log-likelihood is used and the final penalized (negative) log-likelihood is

$$Q(\beta) = -\log L(\beta|y) + P(\beta)$$

Here $P(\beta)$ is a penalization function. Note that since we are now looking at the *negative* log-likelihood, we now want to minimize $Q(\beta)$. The penalization function assigns large values for large β, so that the optimization problem favors small values of β.

There are *many* ways of penalizing β and each method yields a different estimator. A popular one is the *ridge* penalty.

Example 4 (Ridge Regression). The ridge penalization term is $P(\beta) = \lambda\beta^T\beta/2$ for some $\lambda > 0$ for

$$Q(\beta) = \frac{(y-X\beta)^T(y-X\beta)}{2} + \frac{\lambda}{2}\beta^T\beta\,.$$

We will minimize $Q(\beta)$ over the space of β and since we are adding an arbitrary term that depends on the size of β, smaller sizes of β will be preferred. Small sizes of β means X are less important, and this will eventually nullify the singularity in X^TX. The larger λ is, the more "penalization" there is for large values of β; λ is typically user-chosen. We will study choosing λ when we cover "cross-validation" later.

We are now interested in finding:

$$\hat{\beta} = \arg\min_{\beta}\left\{\frac{(y-X\beta)^T(y-X\beta)}{2} + \frac{\lambda}{2}\beta^T\beta\right\}.$$

To carry out the minimization, we take the derivative:

$$\begin{aligned}\frac{dQ(\beta)}{d\beta} &= \frac{1}{2}(-2X^Ty + 2X^TX\beta) + \lambda\beta \stackrel{set}{=} 0\\ &\Rightarrow (X^TX + \lambda I_p)\hat{\beta} - X^Ty = 0\\ &\Rightarrow \hat{\beta}_{\text{ridge}} = (X^TX + \lambda I_p)^{-1}X^Ty\,.\end{aligned}$$

Note: Verify that the Hessian matrix is positive definite for yourself.

Note that $(X^TX + \lambda I_p)$ is always positive definite for $\lambda > 0$ since for any $a \in \mathbb{R}^p \neq 0$

$$a^T(X^TX + \lambda I_p)a = a^TX^TXa + \lambda a^Ta > 0$$

Thus, the final ridge solution always exists even if X^TX is not invertible.

Pros:
We have an estimate of β!
In terms of certain criterion, we actually do better than non-penalized estimation even when (X^TX) is invertible.

Cons:
The estimator is not an MLE, so we cannot use distributional properties to construct confidence intervals. This is a big problem, and is addressed by bootstrapping, which we will get to.

We will study one more penalization method later.

Questions to think about

1. Under the normal likelihood, what is the distribution of $\hat{\beta}_{\text{MLE}}$ (when it exists) and $\hat{\beta}_{\text{ridge}}$? Are they unbiased? Which one has a smaller variance (covariance)?
2. What other penalization functions can you think of? Recall that $\beta^T\beta = \|\beta\|_2^2$.

MTH 511a - 2021: L19 - Newton-Raphson's Method

Instructor: Dootika Vats

1 No closed-form MLEs

In the last lecture we introduced maximum likelihood estimation. Obtaining MLE estimates for a problem requires maximizing the likelihood. However, it is possible that no analytical form of the maxima is possible!

This is a common challenge in many models and estimation problems, and requires sophisticated optimization tools. In the next few weeks, we will go over some of these optimization methods.

1.1 Examples

Example 1 (Gamma Distribution). Let $X_1, \ldots, X_n \overset{iid}{\sim} \text{Gamma}(\alpha, 1)$. The likelihood function is

$$\begin{aligned} L(\alpha|x) &= \prod_{i=1}^{n} \frac{1}{\Gamma(\alpha)} x_i^{\alpha-1} e^{-x_i} \\ &= \frac{1}{\Gamma(\alpha)^n} \prod_{i=1}^{n} x_i^{\alpha-1} e^{-\sum x_i} \\ \Rightarrow l(\alpha) := \log L(\alpha|x) &= -n \log(\Gamma(\alpha)) + (\alpha - 1) \sum_{i=1}^{n} \log x_i - \sum_{i=1}^{n} x_i \end{aligned}$$

Taking first derivative

$$\frac{dl(\alpha)}{d\alpha} = -n\frac{\Gamma'(\alpha)}{\Gamma(\alpha)} + \sum_{i=1}^{n} \log X_i \stackrel{set}{=} 0$$

Taking second derivative

$$\frac{d^2\, l(\alpha)}{d\alpha^2} = -n\frac{d^2}{d\alpha^2}\log(\Gamma(\alpha)) < 0 \quad \text{(polygamma function of order 1 is } > 0)$$

Here

$$\frac{d^2}{d\alpha^2}\log(\Gamma(\alpha))$$

is the polygamma function of order 1, which is always positive (look it up). So we know that the function is concave and a unique maximum exists, but not available in closed form. We cannot get an analytical form of the MLE for α. In such cases, we will use optimization methods.

2 Numerical optimization methods

We will cover three optimization methods:

- Newton-Raphson method
- Gradient ascent (descent)
- The MM algorithm

A general numerical optimization problem is framed in the following way. Let $f(\theta)$ be an *objective function* that is the main function of interest and needs to be either maximized or minimized. Then, we want to solve the following maximization

$$\theta_* = \arg\max_{\theta} f(\theta)$$

All the above three algorithms are such that they generate a sequence of $\{\theta_{(k)}\}$ such that the goal is for $\theta_{(k)} \to \theta_*$ in a deterministic manner (non-random convergence).

All methods that we will learn will find a local optima. Some will guarantee a local maxima, but not guarantee a global maxima, some will guarantee a local optima (so max or min), but not a global maxima. If the objective function is concave, then all methods

will guarantee a global maxima!

Recall that a (univariate) function f is concave if $f'' < 0$ and a (multivariate) function f is concave if its Hessian is negative definite: for all $a \neq 0 \in \mathbb{R}^p$,

$$a^T \left[\nabla^2 f\right] a < 0\,.$$

2.1 Newton-Raphson's method

To solve this optimization problem, consider starting at a point $\theta_{(0)}$. Then subsequent elements of the sequence are determined in the following way.

Suppose that the objective function f is such that a second derivative exists. Since $f(\theta)$ is maximized at the unknown θ_*,

$$f'(\theta_*) = 0.$$

Applying first order Taylor's series expansion (and ignoring higher orders) to $f'(\theta_*)$ about the current iterate $\theta_{(k)}$

$$\begin{aligned} &f'(\theta_*) \approx f'(\theta_{(k)}) + (\theta_* - \theta_{(k)}) f''(\theta_{(k)}) = 0 \\ &\Rightarrow \theta_* \approx \theta_{(k)} - \frac{f'(\theta_{(k)})}{f''(\theta_{(k)})}\,, \end{aligned}$$

where the approximation is best when $\theta_{(k)} = \theta^*$ and the approximation is weak when $\theta_{(k)}$ is far from θ_*. Thus, if we start from an arbitrary point using successive updates of the right hand side, we will get closer and closer to θ_*.

The Newton-Raphson method is

$$\theta_{(k+1)} = \theta_{(k)} - \frac{f'(\theta_{(k)})}{f''(\theta_{(k)})}$$

This works because when $f'(\theta_k) < 0$, the function is increasing at $\theta_{(k)}$, and thus Newton-Raphson increases $\theta_{(k)}$; and vice-versa.

You stop iterating when $|\theta_{(k+1)} - \theta_{(k)}| < \epsilon$ for some chosen tolerance ϵ or $|f'(\theta_{(k+1)})| \approx 0$.

> *If the objective function is concave, the N-R method will converge to the global maxima. Otherwise it converges to a local optima or diverges!*

Example 2 (Gamma distribution continuted). Our objective function is the log-likelihood:

$$f(\alpha) = -n\log(\Gamma(\alpha)) + (\alpha - 1)\sum_{i=1}^{n}\log x_i - \beta\sum x_i$$

First derivative

$$f'(\alpha) = -n\frac{\Gamma'(\alpha)}{\Gamma(\alpha)} + \sum_{i=1}^{n}\log X_i$$

Second derivative

$$f''(\alpha) = -n\frac{d^2}{d\alpha^2}\log(\Gamma(\alpha)) < 0\,.$$

Thus the log-likelihood is concave, which implies there is a global maxima! The Newton-Raphson algorithm will converge to this global maxima.

Start with a reasonable starting value: α_0. Then iterate with

$$\alpha_{(k+1)} = \alpha_{(k)} - \frac{f'(\alpha_{(k)})}{f''(\alpha_{(k)})}$$

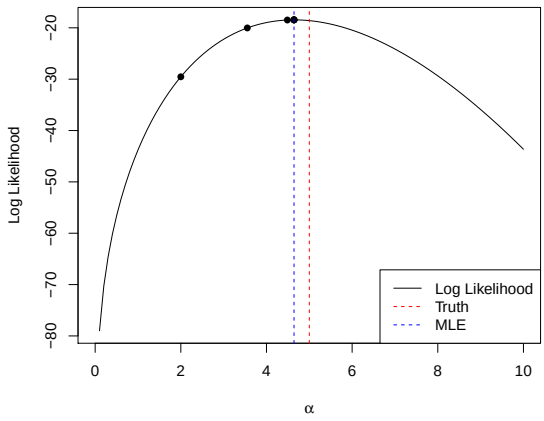
Polygamma functions are in `psi` function in the `pracma` R package.

What is a good starting value α_0? Well, we know that the mean of a Gamma$(\alpha, 1)$ is α, so a good starting value is $\alpha_0 = n^{-1}\sum_{i=1}^{n} X_i$.

```
#######################################################
### MLE for Gamma(alpha, 1)
#######################################################
set.seed(100)
library(pracma) #for psi function

#######################################################
# riginal data sample size is small first
# The NR methods estimates the MLE. Here the
# blue and red lines will not match because
# the data is not large enough for the consistency of
# the MLE to kick in.


alpha <- 5 #true value of alpha
n <- 10 # actual data size is small first
dat <- rgamma(n, shape = alpha, rate = 1)
```

```r
alpha_newton <- numeric()
epsilon <- 1e-8 #some tolerance level preset
alpha_newton[1] <- 2 #alpha_0
count <- 1
tol <- 100 # large number
while(tol > epsilon)
{
  count <- count + 1

  #first derivative
  f.prime <- -n*psi(k = 0, alpha_newton[count - 1]) + sum(log(dat))

  #second derivative
  f.dprime <- -n*psi(k = 1, alpha_newton[count - 1])
  alpha_newton[count] <- alpha_newton[count - 1] - f.prime/f.dprime
  tol <- abs(alpha_newton[count] - alpha_newton[count-1])
}
alpha_newton
# [1] 2.000000 3.552357 4.487264 4.640581 4.643535 4.643536 4.643536

#Plot the log.likelihood for different values of alpha
alpha.grid <- seq(0, 10, length = 100)
log.like <- numeric(length = 100)
for(i in 1:100)
{
  log.like[i] <- sum(dgamma(dat, shape = alpha.grid[i], log = TRUE))
}
plot(alpha.grid, log.like, type = 'l', xlab = expression(alpha), ylab =
    "Log Likelihood")
abline(v = alpha, col = "red", lty = 2)
for(t in 1:count)
{
  points(alpha_newton[t], sum(dgamma(dat, shape = alpha_newton[t], log =
      TRUE)), pch = 16)
}
abline(v = tail(alpha_newton[count]), col = "blue", lty = 2)
legend("bottomright", legend = c("Likelihood", "Truth", "MLE"), lty =
```

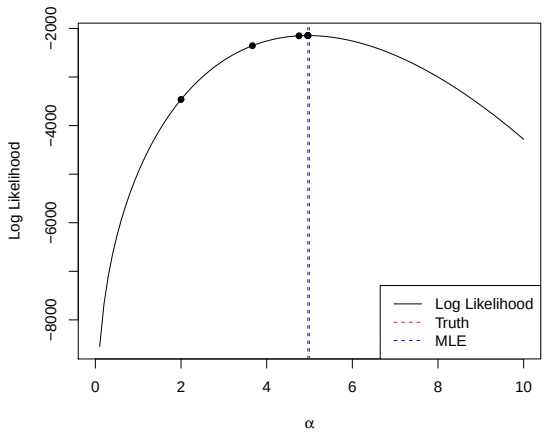
```
  c(1,2,2), col = c("black", "red", "blue"))
```



Note the impact of the sample size of the original data. The Newton-Raphson algorithm converges to the MLE. If the data size is small, the MLE may not be close to the truth. This is why we see that the blue and red lines are far from each other. However, when increase the observed data to be 1000 observations, the consistency of the MLE should kick in and we expect to see the blue and red lines to be similar (below).

```
#######################################################
# Increasing original data sample size.
# Now the MLE is closer to the "truth"
# and our NR method obtains the MLE.
# Blue and red lines should match a lot

# Randomly generate data
alpha <- 5 #true value of alpha
n <- 1000 # actual data size is small first
dat <- rgamma(n, shape = alpha, rate = 1)
alpha_newton <- numeric()
epsilon <- 1e-8 #some tolerance level preset
alpha_newton[1] <- 2 #alpha_0
count <- 1
tol <- 100 # large number
while(tol > epsilon)
{
  count <- count + 1
  f.prime <- -n*psi(k = 0, alpha_newton[count - 1]) + sum(log(dat))
```

```
  f.dprime <- -n*psi(k = 1, alpha_newton[count - 1])
  alpha_newton[count] <- alpha_newton[count - 1] - f.prime/f.dprime
  tol <- abs(alpha_newton[count] - alpha_newton[count-1])
}
alpha_newton
# [1] 2.000000 3.666788 4.755252 4.957521 4.962359 4.962361 4.962361

#Plot the log.likelihood for different values of alpha
alpha.grid <- seq(0, 10, length = 100)
log.like <- numeric(length = 100)
for(i in 1:100)
{
  log.like[i] <- sum(dgamma(dat, shape = alpha.grid[i], log = TRUE))
}
plot(alpha.grid, log.like, type = 'l', xlab = expression(alpha), ylab =
    "Log Likelihood")
abline(v = alpha, col = "red", lty = 2)
for(t in 1:count)
{
  points(alpha_newton[t], sum(dgamma(dat, shape = alpha_newton[t], log =
      TRUE)), pch = 16)
}
abline(v = tail(alpha_newton[count]), col = "blue", lty = 2)
legend("bottomright", legend = c("Likelihood", "Truth", "MLE"), lty =
    c(1,2,2), col = c("black", "red", "blue"))
```



Example 3 (Location Cauchy distribution). Consider the location Cauchy distribution with mode at $\mu \in \mathbb{R}$. The goal is to find the MLE for μ.

$$f(x|\mu) = \frac{1}{\pi} \frac{1}{(1 + (x - \mu)^2)} .$$

First, we find the log-likelihood

$$L(\mu|X) = \prod_{i=1}^{n} f(X_i|\mu) = \pi^{-n} \prod_{i=1}^{n} \frac{1}{1 + (x_i - \mu)^2}$$

$$\Rightarrow l(\mu) := \log L(\mu|X) = -n \log \pi - \sum_{t=1}^{n} \log(1 + (X_i - \mu)^2) =: f(\mu) .$$

It is evident that a closed form solution is difficult. So, we find the derivates.

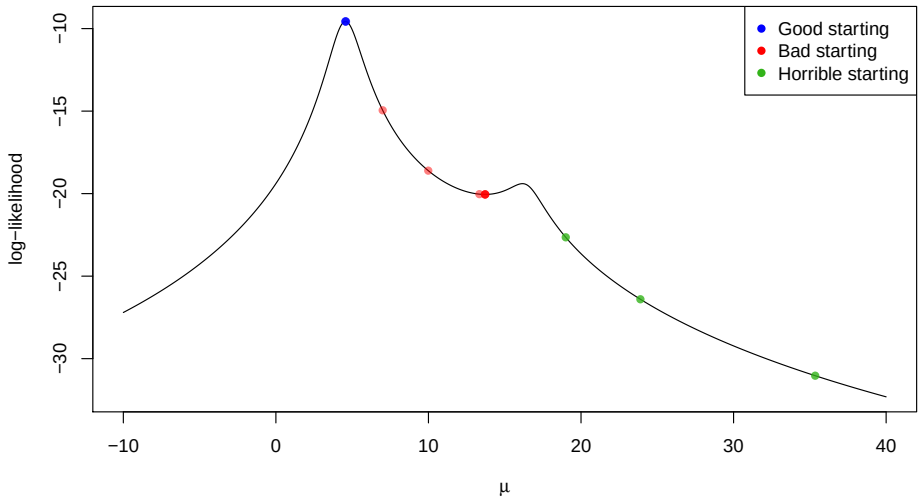
$$f'(\mu) = 2 \sum_{i=1}^{n} \frac{X_i - \mu}{1 + (X_i - \mu)^2} .$$

$$f''(\mu) = 2 \sum_{i=1}^{n} \left[2 \frac{(X_i - \mu)^2}{[1 + (X_i - \mu)^2]^2} - \frac{1}{1 + (X_i - \mu)^2} \right] ,$$

which may be positive or negative. So this is not a concave function. This implies that Newton-Raphson is not guaranteed to converge to the global maxima! It may not even converge to a local maxima, and may converge to a local minima. Thus, we will need to be careful in choosing starting values.

1. Set $\mu_0 = \text{Median}(X_i)$ since the mean of Cauchy does not exist and the Cauchy centered at μ is symmetric around μ.
2. Determine:
$$\mu_{(k+1)} = \mu_{(k)} - \frac{f'(\mu_{(k)})}{f''(\mu_{(k)})}$$
3. Stop when $|\mu_{(k+1)} - \mu_{(k)}| < \epsilon$ for a chosen tolerance level ϵ.

The code for this is attached in the material, but below is a figure of what the log-likelihood looks like and the behavior of the Newton-Raphson algorithm for different starting values. It shows that choosing a good starting value is very important.



Newton-Raphson in Higher Dimensions

The NR method can be found in the same way using the multivariate Taylor's expansion. Let $\theta = (\theta_1, \theta_2, \ldots, \theta_p)$. Then first let ∇f denote the gradient of f and $\nabla^2 f$ denote the Hessian. So

$$\nabla f = \begin{bmatrix} \dfrac{\partial f}{\partial \theta_1} \\ \vdots \\ \dfrac{\partial f}{\partial \theta_p} \end{bmatrix} \quad \text{and} \quad \nabla^2 f = \begin{bmatrix} \dfrac{\partial^2 f}{\partial \theta_1^2} & \dfrac{\partial^2 f}{\partial \theta_1 \theta_2} & \cdots \\ \vdots & \vdots & \vdots \\ \dfrac{\partial^2 f}{\partial \theta_p \theta_1} & \cdots & \dfrac{\partial^2 f}{\partial \theta_p^2} \end{bmatrix}$$

Then, the function f is concave if $\nabla^2 f$ is negative definite. *So always check that first to know if there is a unique maximum.*

Using a similar multivariate Taylor series expansion, the Newton-Raphson update solves the system of linear equations

$$\nabla f(\theta_{(k)}) + \nabla^2 f(\theta_k)(\theta_{k+1} - \theta_k) = 0\,.$$

If $\nabla^2 f(\theta_{(k)})$ is invertible, then you have that

$$\theta_{(k+1)} = \theta_{(k)} - \left[\nabla^2 f(\theta_{(k)})\right]^{-1} \nabla f(\theta_{(k)})\,.$$

Iterations are stopped with when $\|\theta_{(k+1)} - \theta_{(k)}\| < \epsilon$ for some chosen tolerance level, ϵ or when the gradient vector is very close to zero.

3 Questions

1. Can you implement the the Newton-Raphson procedure for linear regression and ridge regression?
2. What are some of the issues in implementing Newton-Raphson? Can we use it for any problem?
3. If the function is not concave and different starting values yield convergence to different points (or divergence), then what do we do?

MTH 511a - 2021: L21 - Gradient Ascent Algorithm

Instructor: Dootika Vats

1 Numerical optimization methods

1.1 Gradient Ascent (Descent)

For concave objective functions, Newton-Raphson is essentially the best algorithm. However, one significant flaw in the algorithm is that information about the Hessian is required to implement it. The Hessian (or the double derivative) may be unavailable or difficult to calculate in some situations.

In such a case, gradient ascent (or gradient descent if the problem is a minimizing problem) is a useful alternative as it does not require the Hessian.

Consider the objective function $f(\theta)$ that we want to maximize and suppose θ_* is the true maximum. Then, by the Taylor series approximation at a fixed θ_0

$$f(\theta) \approx f(\theta_0) + f'(\theta_0)(\theta - \theta_0) + \frac{f''(\theta_0)}{2}(\theta - \theta_0)^2$$

If $f''(\theta)$ is unavailable, consider assuming that the double derivative is a negative constant. That is, assume that f is quadratic and concave. Then for a $t > 0$,

$$f(\theta) \approx f(\theta_0) + f'(\theta_0)(\theta - \theta_0) - \frac{1}{2t}(\theta - \theta_0)^2$$

Maximizing $f(\theta)$ and using this approximation would imply maximizing the right hand side. Taking the derivative with respect to θ setting it to zero:

$$f'(\theta_0) - \frac{\theta - \theta_0}{t} \stackrel{set}{=} 0 \Rightarrow \theta = \theta_0 + tf'(\theta_0)\,.$$

Thus, given a $\theta_{(k)}$, the gradient ascent algorithm does the update

$$\theta_{(k+1)} = \theta_{(k)} + tf'(\theta_{(k)})\,,$$

for $t > 0$. The iteration can be stopped when $|\theta_{(k+1)} - \theta_{(k)}| < \epsilon$ for $\epsilon > 0$ or when $|f'(\theta_{k+1})| \approx 0$.

> *For concave functions, gradient ascent converges to the global maxima. In general, gradient ascent always converges to a local maxima, as long as you don't start from a local minima.*

The algorithm essentially does a local-quadratic approximation at the current point $\theta_{(k)}$ and then maximizes that quadratic equation. The value of t indicates how far do we want to jump and is a tuning parameter. If t is large, we take big jumps, as opposed to t small, where the jumps are smaller.

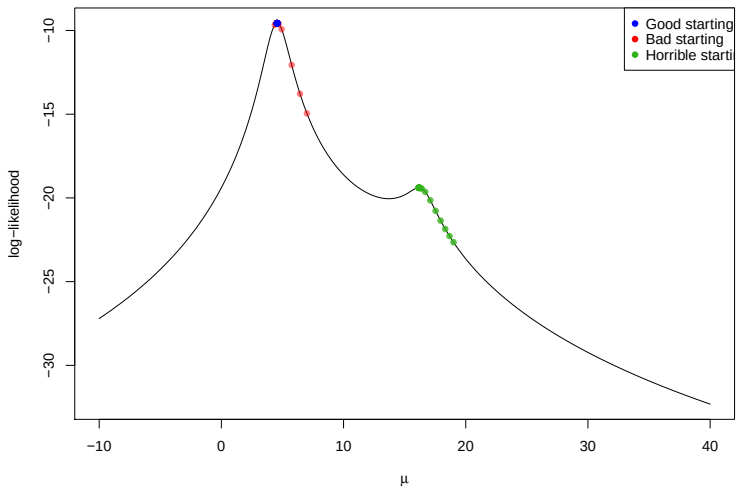
Example 1 (Location Cauchy). Recall the location Cauchy example in the previous lecture where the likelihood is not concave. We can implement gradient ascent here and since gradient ascent always converges to a local maxima, it does something more reasonable here (we set $t = .3$) (the codes have been shared)

```
################################################
## MLE for location Cauchy distribution using
## Gradient Ascent method
## We will plot the likelihood as well
################################################
```

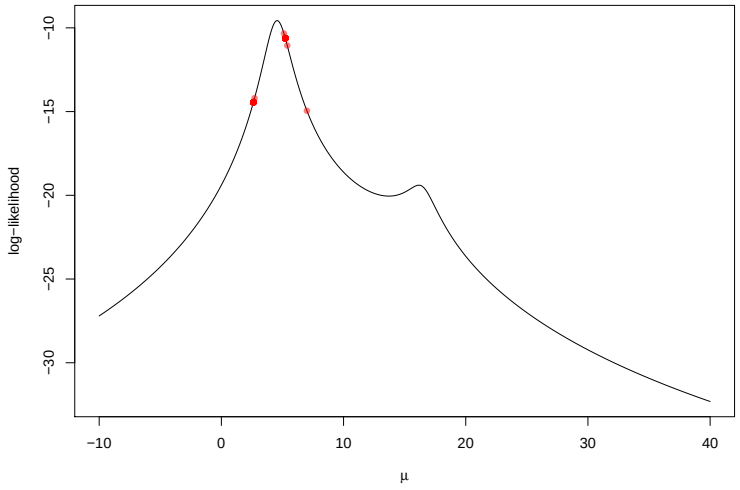
```
set.seed(1)
mu.star <- 5 # Setting true mu
n <- 4 # sample size
X <- rt(n, df = 1) + mu.star

## Function calculates the log-likelihood
log.like <- function(mu, X)
{
  n <- length(X)
  rtn <- -n*log(pi) - sum( log(1 + (X - mu)^2) )
  return(rtn)
}

mu.x <- seq(-10, 40, length = 1e3) # A sequence of mu's
ll.est <- sapply(mu.x, log.like, X) # evaluating log-likelihood at the mus
plot(mu.x, ll.est, type = 'l', ylab = "log-likelihood", xlab =
    expression(mu)) # plotting log-likelihood. Not concave, so we need to
    choose good starting values.


## Starting Gradient-Ascent method
tol <- 1e-5 # tolerance level for when to stop algorithm


## Returns derivate of log-likelihood
f.prime <- function(X, mu)
{
  rtn <- sum(2* (X - mu)/(1 + (X-mu)^2)) #f.prime
  return(rtn)
}

# Returns double derivative of log-likelihood.
f.double.prime <- function(X, mu)
{
  rtn <- sum( 2 * ( 2*(X-mu)^2/ (1 + (X - mu)^2)^2 - (1 + (X-mu)^2)^(-1) ) )
  return(rtn)
}
```

```
## Loop below stops when |mu_(k+1) - mu_(k)| < tol
t <- .3 # change this to 1 and see what happens to the "bad" starting value



## Loop below stops when |mu_(k+1) - mu_(k)| < tol
current <- 7 # Bad starting value
diff <- 100
iter <- 0
mu.k <- current
while( (diff > tol) && iter < 100)
{
  iter <- iter + 1
  update <- current + t*f.prime(X, current)
  mu.k <- c(mu.k, update)
  diff <- abs(current - update)
  current <- update
}
current # final approximation to MLE
evals <- sapply(mu.k, log.like, X)
plot(mu.x, ll.est, type = 'l', ylab = "log-likelihood", xlab =
    expression(mu))
points(mu.k, evals, pch = 16, col = rgb(1,0,0, alpha = .5))


current <- median(X) # Good starting value
diff <- 100 # inital large value for difference
iter <- 0   # counting the number of iterations

mu.k <- current
while( (diff > tol) && iter < 100)
{
  iter <- iter + 1
  update <- current + t*f.prime(X, current) # GD update
  mu.k <- c(mu.k, update)
  diff <- abs(current - update)
  current <- update
}
```

```
current # final approximation to MLE
evals <- sapply(mu.k, log.like, X)
points(mu.k, evals, pch = 16, col = rgb(0,0,1, alpha = .5))


current <- 19 # Worst starting value
diff <- 100
iter <- 0
mu.k <- current
while( (diff > tol) && iter < 100)
{
  iter <- iter + 1
  update <- current + t*f.prime(X, current)
  mu.k <- c(mu.k, update)
  diff <- abs(current - update)
  current <- update
}
current # final approximation to MLE
evals <- sapply(mu.k, log.like, X)
points(mu.k, evals, pch = 16, col = rgb(.2,.7,.1, alpha = .8))

legend("topright", legend = c("Good starting", "Bad starting", "Horrible
    starting"), pch = 16, col = c("blue", "red", rgb(.2,.7,.1)))
```



If we rerun this with the red starting values and $t = 1$, this is too much of a jump size, and the optimization starts oscillating between two points.



So it is really important to try different step sizes t when implementing gradient ascent. There are other solutions to this like adaptive step sizes which we will not discuss.

Gradient Ascent in Higher Dimensions

By a multivariate Taylor series expansion, we can obtain a similar motivation and the iteration in the algorithm is:

$$\theta_{(k+1)} = \theta_{(k)} + t\nabla f(\theta_{(k)}) \, .$$

Example 2 (Logistic regression). We have studied linear regression for modeling continuous responses. But when Y is a response of 1s and 0s (Bernoulli) then assuming the Ys are normally distributed is not appropriate. Instead, when the ith covariate is $(x_{i1}, \ldots, x_{ip})^T$, then for $\beta \in \mathbb{R}^p$ logistic regression assumes the model

$$Y_i \sim \text{Bern}\left(\frac{e^{x_i^T\beta}}{1 + e^{x_i^T\beta}}\right) \, .$$

In other words, the probability that any response takes the value 1 is

$$\Pr(Y_i = 1) = \frac{e^{x_i^T \beta}}{1 + e^{x_i^T \beta}} := p_i \,.$$

Our goal is to obtain the MLE of β. As usual, first we write down the log-likelihood.

$$\begin{aligned}
L(\beta|Y) &= \prod_{i=1}^{n} (p_i)^{y_i} (1 - p_i)^{1-y_i} \\
\Rightarrow l(\beta) &= \sum_{i=1}^{n} y_i \log p_i + \sum_{i=1}^{n} (1 - y_i) \log(1 - p_i) \\
&= \sum_{i=1}^{n} \log(1 - p_i) + \sum_{i=1}^{n} y_i \left(\log p_i - \log(1 - p_i)\right) \\
&= -\sum_{i=1}^{n} \log\left(1 + \exp(x_i^T \beta)\right) + \sum_{i=1}^{n} y_i x_i^T \beta
\end{aligned}$$

Taking derivative:

$$\nabla l(\beta) = \sum_{i=1}^{n} x_i \left[y_i - \frac{e^{x_i^T \beta}}{1 + e^{x_i^T \beta}} \right] \stackrel{set}{=} 0$$

An analytical solution here is not possible, thus a numerical optimization tool is required.

To show that the Hessian is negative-definite, we take derivative again. But first, note that

$$\begin{aligned}
\nabla l(\beta) &= \sum_{i=1}^{n} x_i \left[y_i - \frac{e^{x_i^T \beta}}{1 + e^{x_i^T \beta}} \right] \\
&= \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} \frac{x_i}{1 + e^{-x_i^T \beta}} \\
\Rightarrow \nabla^2 l(\beta) &= 0 - \sum_{i=1}^{n} x_i \frac{x_i^T e^{-x_i^T \beta}}{(1 + e^{-x_i^T \beta})^2} \\
&= -\sum_{i=1}^{n} p_i (1 - p_i) x_i x_i^T \\
&= -X^T P X \,,
\end{aligned}$$

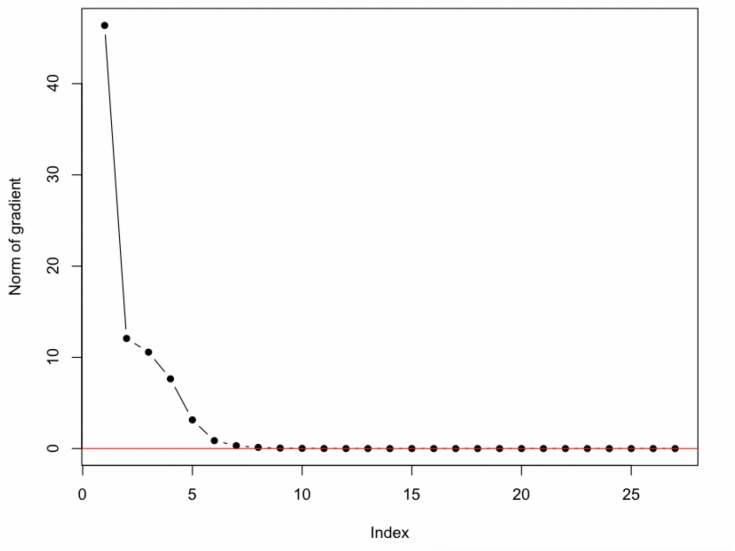
where $P = \text{diag}(p_i(1 - p_i)$. Note that $-X^T P X = -(P^{1/2} X)^T (P^{1/2} X)$, which is guar-

anteed to be negative-definite (not strictly). Thus the target function is concave!

Note: Thus, we can use either Newton-Raphson or Gradient Ascent successfully. We will use gradient ascent, and you should on your own, implement N-R. Since the function is concave, N-R is infact far more effective and requires no tuning.

```
################################################
## MLE for logistic regression
## Using gradient ascent
################################################
library(mcmc) #to load a dataset
data(logit)
head(logit) # y is response and 4 covariates
#   y     x1     x2     x3     x4
# 1 0 -0.162 -0.348 -0.524 -0.312
# 2 1  0.187 -0.410 0.362 -0.366
# 3 1  0.160  1.649 -0.664 0.051
# 4 1  1.536  0.084  0.403  1.732
# 5 1 -0.162 -0.061 0.192 0.256
# 6 0 -2.855 -3.341 -2.549 -1.461

y <- logit$y
X <- as.matrix(logit[, 2:5])
p <- dim(X)[2]


f.gradient <- function(y, X, beta)
{
  beta <- matrix(beta, ncol = 1)
  pi <- exp(X %*% beta) / (1 + exp(X%*%beta))
  rtn <- colSums(X* as.numeric(y - pi))
  return(rtn)
}

store.beta <- matrix(0, nrow = 1, ncol = p)
store.grads <- NULL
beta_k <- rep(0, p) # start at all 0s
grads <- 100 # large values
```

```r
t <- .1
tol <- 1e-8
iter <- 0
while((grads > tol) && iter < 1e4) #not too many iterations
{
  iter <- iter+1
  old <- beta_k
  foo <- f.gradient(y = y, X= X, beta = old)
  grads <- sqrt(sum(foo^2))
  store.grads <- c(store.grads, grads) ## storing the gradients
  beta_k = old + t* foo
  store.beta <- rbind(store.beta, beta_k)
}
iter # number of iterations
# [1] 27
beta_k # last estimate
#       x1        x2        x3        x4
# 0.7178916 0.8789924 0.4227914 0.5874717

plot(store.grads, type= "b", pch = 16, ylab = "Norm of gradient")
abline(h = 0, col = "red")
```



1.2 Questions to think about

1. Can you implement Newton-Raphson for the logistic regression problem? Which of the two algorithms is better?
2. What might be a way to adapt the step size t in a problem? Try and implement the logistic regression example with other t.
3. If gradient-descent is not guaranteed to converge to a global maxima, how can we better our chances to obtaining a good estimate of the MLE?

MTH 511a - 2021: L22 - The MM Algorithm

Instructor: Dootika Vats

The two optimization techniques we've learned have utilized derivatives. But what if the objective function is not differentiable? Or the derivatives are too complicated to write down explicitly. What do we do then?

1 MM Algorithm

Consider obtaining a solution to

$$\theta_* = \arg\max_{\theta} f(\theta)$$

The "Minorize/Maximize algorithm" algorithm at a current iterate, finds a "minorizing" function at that point, and then maximizes that minorizing function. That is, at any given iteration, consider a *minorizing function* $\tilde{f}(\theta|\theta_{(k)})$ such that:

- $f(\theta_k) = \tilde{f}(\theta_k|\theta_k)$
- $f(\theta) \geq \tilde{f}(\theta|\theta_k)$ for all other θ

Then, $\theta_{(k+1)}$ is obtained as

$$\theta_{(k+1)} = \arg\max_{\theta} \tilde{f}(\theta|\theta_{(k)})\,.$$

The algorithm has the ascent property in that every update increases the objective value. That is

$$\begin{aligned} f(\theta_{(k+1)}) &\geq \tilde{f}(\theta_{(k+1)} \mid \theta_{(k)}) \\ &\geq \tilde{f}(\theta_{(k)} \mid \theta_{(k)}) \\ &= f(\theta_{(k)}) \,. \end{aligned}$$

When minimizing an objective function, we the opposite: we find a majorizing function and then minimize it. The key to implementing the MM algorithm is to finding a good *minorizing* function. This can be done in a few different ways and generally application specific.

1.1 Bridge Regression (including Lasso)

Example 1 (Bridge regression). Recall the case of the penalized (negative) log-likelihood problem during ridge regression, where the objective function was:

$$Q(\beta) = \frac{(y - X\beta)^T (y - X\beta)}{2} + \frac{\lambda}{2} \sum_{i=1}^{p} \beta_i^2 \,.$$

By changing the penalty function, the above ridge regression objective function can be generalized as *bridge regression* when the objective function is

$$Q_B(\beta) = \frac{(y - X\beta)^T (y - X\beta)}{2} + \frac{\lambda}{\alpha} \sum_{i=1}^{p} |\beta_i|^\alpha \,,$$

for $\alpha \in [1, 2]$ and $\lambda > 0$. When $\alpha = 2$, this is ridge regression, and when $\alpha = 1$, this is the popular *lasso* regression. Different choices of α, lead to different style of penalization. For a given λ, smaller values of α push the estimates closer towards zero.

We need to find

$$\arg\min_{\beta} Q_B(\beta).$$

First note that, $(y - X\beta)^T (y - X\beta)$ is a convex function and $|\beta_i|^\alpha$ is convex for $\alpha \geq 1$. Since the sum of positive convex functions is convex, the objective function is convex, thus our optimization algorithms will find a global *minima*.

Note that for $\alpha = 1$, the objective function is not differentiable at 0, and for $\alpha \in$

$(1, 2)$, the function is not twice differentiable at 0. Thus, using Newton-Raphson and gradient descent is not possible. We will instead use an MM algorithm. Since this is a minimization problem, we will find a *majorizing function* and then minimize the majorizing function.

We will try to find a majorizing function that upper bounds the objective $Q_B(\beta)$, and then minimize the majorizing function. Intuitively, optimizing the majorizing function will again require derivatives of the majorizing function. Thus our goal is to find a majorizing function that *does not* contain an absolute value, and is thus differentiable.

Consider a function $h(u) = u^{\alpha/2}$ for $u \geq 0$ and see that

$$h'(u) = \frac{\alpha}{2} u^{\alpha/2-1}$$

and

$$h''(u) = \frac{\alpha}{2}\left(\frac{\alpha}{2} - 1\right) u^{\alpha/2-2} \leq 0$$

so $h(u)$ is a concave function for $\alpha \in [1, 2]$. For a concave function, by the "Rooftop theorem", the first order Taylor series creates a tangent line that is above the function. Thus for a u^*,

$$h(u) \leq h(u^*) + h'(u^*)(u - u^*) = h(u^*) + \frac{\alpha}{2}(u^*)^{\alpha/2-1}(u - u^*)\,.$$

For any given iteration of the optimization given $\beta_{(k)}$, taking $u = |\beta_i|^2$ and $u^* = |\beta_{i,(k)}|^2$ where β_i is the ith component of the vector β. Then,

$$\begin{aligned} h(u) = |\beta_i|^\alpha &\leq |\beta_{i,(k)}|^\alpha + \frac{\alpha}{2}|\beta_{i,(k)}|^{\alpha-2}\left(\beta_i^2 - \beta_{i,(k)}^2\right) \\ &= |\beta_{i,(k)}|^\alpha - \frac{\alpha}{2}|\beta_{i,(k)}|^\alpha + \frac{\alpha}{2}|\beta_{i,(k)}|^{\alpha-2}\beta_i^2 \\ &= \text{constants} + \frac{m_i}{2}\beta_i^2 \end{aligned}$$

where $m_i = \alpha|\beta_{i,(k)}|^{\alpha-2}$. (You will see that the constants will not be important.)

Now that we have upper bounded the the penalty function, we have an upper bound on the full objective function! So, the objective function can be bounded above by:

$$Q_B(\beta) \leq \text{constants} + \frac{(y - X\beta)^T(y - X\beta)}{2} + \frac{\lambda}{2\alpha}\sum_{j=1}^{p} m_j\beta_j^2\,.$$

Why is this upper bound useful?

- Remember that at any given iteration, the optimization is with respect to β. Thus, the constants are truly constants.
- The upper bound has no absolute values and is easily differentiable!
- Recall that we obtained the upper bound function using a derivative of $h(u)$. This derivative is not defined at $u = 0$. However, we're only using the derivative function at $u^* = |\beta_{i,(k)}|^2$, which is the previous iteration. So as long as we DO NOT START at zero, this upper bound is valid.
- Finally, the upper bound is easily optimizable, as it is similar to ridge. (See below)

The objective function is similar to ridge regression, except it is "weighted". Following the same steps as in ridge optimization, you can show that the minimum occurs at

$$\beta_{(k+1)} = (X^T X + \lambda M_{(k)})^{-1} X^T y \,,$$

where $M_{(k)} = \text{diag}(m_1, m_2, \ldots, m_p)$. Note that here $M_{(k)}$ is what drives the direction of the optimization.

```
################################################
## Bridge regression and the MM algorithm
## Compare for different values of alpha
################################################
set.seed(1)
n <- 100
p <- 5
beta.star <- c(0,0,0,rnorm(p-3, sd = 1))
beta.star # to output

# Making design matrix, first column is 1
X <- cbind(1, matrix(rnorm(n*(p-1)), nrow = n, ncol = (p-1)))

# Generating response
y <-  X %*% beta.star + rnorm(n, mean = 0, sd = 1)
```

In the above code we generate *synthetic* data following the model

$$Y = X\beta^* + \epsilon$$

where $\epsilon \sim N_n(0, I_n)$ and β^* is the ground-truth chosen value of the regression coefficient. We will now obtain MLE, Ridge and Bridge estimates for various values of α, using a given value of λ.

```
###############
# First MLE
###############
# MLE of beta
beta.mle <- solve( t(X) %*%X ) %*% t(X) %*%y

# Bridge regression alphas
alpha.vec <- c(1, 1.5, 1.8)

lambda <- 10
beta.bridge <- matrix(0, nrow = p, ncol = length(alpha.vec))
tol <- 1e-5


for(i in 1:length(alpha.vec) ) # loop for each alpha
{
  current <- solve( t(X) %*%X + diag(lambda,p) ) %*% t(X) %*%y # start at
     ridge solution
  iter <- 0
  diff <- 100
  while( (diff > tol) && iter < 1000)
  {
    iter <- iter + 1

    # M matrix diagonals
    ms <- as.vector( lambda/2 *( abs(current))^(alpha.vec[i] - 2) )

    # MM update -- using qr.solve for numerical stability
    update <- qr.solve(t(X) %*% X + diag(ms, p)) %*% t(X) %*% y

    diff <- sqrt(sum( (current - update)^2 ) )
    current <- update
  }
  print(c(i,iter)) # to see if any alpha maxed out
  beta.bridge[,i] <- current
```

```
}

## Comparing MLE, ridge, and Bridge for different alpha values
beta.ridge <- solve( t(X) %*%X + diag(lambda,p) ) %*% t(X) %*%y
ests <- round(cbind(beta.mle, beta.bridge, beta.ridge, beta.star), 4)
colnames(ests) <- c("MLE", alpha.vec, 2, "Truth")
ests
#         MLE      1    1.5    1.8      2   Truth
# [1,] -0.0951 -0.0327 -0.0761 -0.0848 -0.0822 0.0000
# [2,] 0.2037 0.1292 0.1732 0.1848 0.1776 0.0000
# [3,] 0.1652 0.1142 0.1468 0.1550 0.1521 0.0000
# [4,] -0.6224 -0.5699 -0.5834 -0.5893 -0.5657 -0.6265
# [5,] 0.2317 0.1671 0.2018 0.2117 0.2026 0.1836
```

Here we start at the corresponding ridge solution to ensure the algorithm works even when $(X^T X)^{-1}$ does not exist. In the above code, if you start from any of the βs being 0, the algorithm will not work since at some point, we would be diving by zero.

Further, try increasing λ to see what happens.

2 Questions to think about

- How do you think we can choose α in any given problem?
- How do you think we can choose λ in any given problem?
- Will the MM algorithm always converge to a global maxima?

MTH 511a - 2021: L23 - The EM Algorithm

Instructor: Dootika Vats

1 The EM algorithm

An important application of the MM algorithm is the Expectation-Maximization (EM) algorithm. Since the EM algorithm is an integral part of statistics on its own, and hence we study it separately. We will first motivate the EM algorithm with an example.

1.1 Gaussian mixture likelihood

Suppose $X_1, X_2, \ldots, X_n \overset{iid}{\sim} F$, where F is mixture of two normal distributions so that the density is:

$$f(x \mid \mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \pi^*) = \pi^* f_1(x \mid \mu_1, \sigma_1^2) + (1-\pi^*) f_2(x \mid \mu_2, \sigma_2^2)\,,$$

where $f_i(x \mid \mu_i, \sigma_i^2)$ is the density of $N(\mu_i, \sigma_i^2)$ distribution for $i = 1, 2$. Given the data, suppose we will to find the maximum likelihood estimates of all 5 parameters: $(\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \pi^*)$. That is, we want to maximize:

$$\begin{aligned} l(\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \pi^* | X) &= \sum_{i=1}^{n} \log f(x_i \mid \mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \pi^*) \\ &= \sum_{i=1}^{n} \log \left[\pi^* f_1(x \mid \mu_1, \sigma_1^2) + (1-\pi^*) f_2(x \mid \mu_2, \sigma_2^2)\right]\,. \end{aligned}$$

There is no analytical solution to the above optimization problem and we have to resort to numerical techniques. Instead of trying to use gradient-based tools, we use a common trick called the *latent variable* or *missing data* trick.

Recall that the data likelihood is a mixture of Gaussians. An interpretation of this is that with probability π^*, any observed X_i is from f_1 and with probability $1 - \pi^*$ it is from f_2.

Suppose we have the information about the the class of each x_i. Thus, suppose the *complete data* was of the form

$$(X_1, Z_1), (X_2, Z_2), \ldots, (X_n, Z_n)\,,$$

where each $Z_i = k$ means that X_i is from population k. If this complete data is available to us, then first note that the joint density is

$$f(x_i, z_i = k) = f(x_i | z_i = k) \Pr(Z_i = k)\,.$$

Suppose $\mathcal{D}_1 = \{i : 1 \le i \le n, z_i = 1\}$ and $\mathcal{D}_2 = \{i : 1 \le i \le n, z_i = 2\}$, with cardinality d_1 and d_2 respectively. The set $\mathcal{D}_1$ and $\mathcal{D}_2$ have the indices of the data that belong to each component of the mixture.

Then the likelihood of the complete data is

$$\begin{aligned}
&L(\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \pi^* | X, Z) \\
&= \prod_{i=1}^{n} f(x_i, z_i) \\
&= \prod_{i \in \mathcal{D}_1} f(x_i, z_i = 1) \prod_{j \in \mathcal{D}_2} f(x_i, z_i = 2) \\
&= \prod_{i \in \mathcal{D}_1} f(x_i | z_i = 1) \Pr(Z_i = 1) \prod_{i \in \mathcal{D}_2} f(x_i | z_i = 2) \Pr(Z_i = 2) \\
&= \prod_{i \in \mathcal{D}_1} \left[\pi^* f_1(x_i | \mu_1, \sigma_1^2)\right] \prod_{j \in \mathcal{D}_2} \left[f_2(x_i | \mu_2, \sigma_2^2)(1 - \pi^*)\right] \\
&= (\pi^*)^{d_1} (1 - \pi^*)^{d_2} \prod_{i \in \mathcal{D}_1} \left[f_1(x_i | \mu_1, \sigma_1^2)\right] \prod_{i \in \mathcal{D}_2} \left[f_2(x_i | \mu_2, \sigma_2^2)\right]\,.
\end{aligned}$$

This means that the log-likelihood of the complete data is

$$\Rightarrow \log L = d_1 \log(\pi^*) + d_2 \log(1 - \pi^*) + \sum_{i \in \mathcal{D}_1} \log f_1(x_i | \mu_1, \sigma_1^2) + \sum_{i \in \mathcal{D}_2} \log f_2(x_i | \mu_2, \sigma_2^2)\,.$$

This log-likelihood is in a far nicer format, so that closed-form estimates are available.

So, if the complete data was available to us, we can easily find the MLE of all the 5 parameters. Unfortunately, the Zs are usually not observed, and only the Xs have been observed. The EM algorithm will solve this problem by estimating the unobserved z_i corresponding to each x_i in an iterative manner.

We will come back to this Gaussian problem again.

1.2 The Expectation-Maximization Algorithm

Suppose, we have a vector of parameters θ, and we have observed the marginal data $X_1, \ldots, X_n$ from the complete data (X_i, Z_i). The objective function is to maximize is

$$\begin{aligned} l(\theta|X) &= \log f(\mathbf{x}|\theta) \\ &= \log \int f(\mathbf{x}, \mathbf{z}|\theta) d\nu_z \,, \end{aligned}$$

where the $\int \cdot \, d\nu_z$ denotes integral or summation based on whether Z is continuous or discrete. Writing the target objective function in this form allows us to do the optimization using EM.

The EM algorithm iterates through the following: Consider a starting value θ_0. Then for any $k+1$ iteration

1. **E-Step**: Compute

$$q(\theta|\theta_{(k)}) = \mathrm{E}_{Z|X}\Big[\log f(\mathbf{x}, \mathbf{z}|\theta) \mid X = x, \theta_{(k)} \Big]$$

   where the expectation is computed with respect to the conditional distribution of Z given $X = x$ for the current iterate $\theta_{(k)}$.

2. **M-Step**: Compute

$$\theta_{(k+1)} = \arg\max_{\theta \in \Theta} q\left(\theta|\theta_{(k)}\right) \,.$$

3. Stop when $\|\theta_{(k+1)} - \theta_{(k)}\| < \epsilon$.

The following theorem will convince us that running the above algorithm will ensure

the $\theta_{(k)}$ converges to a local maxima. The trick to implementing the EM algorithm for a general problem is in finding an appropriate joint distribution (X, Z) for which the E-step is computable.

Theorem 1. *The EM algorithm is an MM algorithm and thus has the ascent property.*

Proof. In order to show the result we first need to find a minorizing function.

The objective function is $\log f(\mathbf{x}|\theta)$. So we need to find $\tilde{f}(\theta|\theta_{(k)})$ such that $\tilde{f}(\theta_{(k)}|\theta_{(k)}) = \log f(\mathbf{x}|\theta_{(k)})$ and in general

$$\tilde{f}(\theta|\theta_{(k)}) \leq \log f(\mathbf{x}|\theta)$$

We will show that $\tilde{f}(\theta|\theta_{(k)})$ is such that

$$\tilde{f}(\theta|\theta_{(k)}) = q(\theta|\theta_{(k)}) \ + \text{constants.}$$

Then, maximizing $\tilde{f}(\theta|\theta_{(k)})$ is equivalent to maximizing $q(\theta|\theta_{(k)})$ (the M step of both EM and MM).

Let

$$\tilde{f}(\theta|\theta_{(k)}) = \int_z \log\{f(\mathbf{x}, \mathbf{z}|\theta)\} f(\mathbf{z}|\mathbf{x}, \theta_{(k)}) dz + \log f(\mathbf{x}|\theta_{(k)}) - \int_z \log\{f(\mathbf{x}, \mathbf{z}|\theta_{(k)})\} f(\mathbf{z}|\mathbf{x}, \theta_{(k)}) dz \,.$$

(The proof technique is setup for continuous Z, but the same proof works for discrete Z as well.)

Naturally, we can see that at $\theta = \theta_{(k)}$, $\tilde{f}(\theta_{(k)}|\theta_{(k)}) = f(\mathbf{x}|\theta_{(k)})$. We will now show that minorizing property.

$$\begin{aligned}
&\tilde{f}(\theta|\theta_{(k)}) \\
&= \int_z \log\{f(\mathbf{x}, \mathbf{z}|\theta)\} f(\mathbf{z}|\mathbf{x}, \theta_{(k)}) dz + \log f(\mathbf{x}|\theta_{(k)}) - \int_z \log\{f(\mathbf{x}, \mathbf{z}|\theta_{(k)})\} f(\mathbf{z}|\mathbf{x}, \theta_{(k)}) dz \\
&= \int_z \log\{f(\mathbf{x}, \mathbf{z}|\theta)\} f(\mathbf{z}|\mathbf{x}, \theta_{(k)}) dz + \int_z \log f(\mathbf{x}|\theta_{(k)}) f(\mathbf{z}|\mathbf{x}, \theta_{(k)}) dz - \int_z \log\{f(\mathbf{x}, \mathbf{z}|\theta_{(k)})\} f(\mathbf{z}|\mathbf{x}, \theta_{(k)}) dz \\
&= \int_z \log \left\{ \frac{f(\mathbf{x}, \mathbf{z}|\theta)\, f(\mathbf{x}|\theta_{(k)})}{f(\mathbf{x}, \mathbf{z}|\theta_{(k)})} \right\} f(\mathbf{z}|\mathbf{x}, \theta_{(k)})
\end{aligned}$$

$$= \int_z \log \left\{ \frac{f(\mathbf{x}, \mathbf{z}|\theta)\, f(\mathbf{x}|\theta_{(k)})}{f(\mathbf{x}, \mathbf{z}|\theta_{(k)})} \right\} f(\mathbf{z}|\mathbf{x}, \theta_{(k)}) + \log f(\mathbf{x}|\theta) - \log f(\mathbf{x}|\theta)$$

$$= \int_z \log \left\{ \frac{f(\mathbf{x}, \mathbf{z}|\theta)\, f(\mathbf{x}|\theta_{(k)})}{f(\mathbf{x}, \mathbf{z}|\theta_{(k)}) f(\mathbf{x}|\theta)} \right\} f(\mathbf{z}|\mathbf{x}, \theta_{(k)}) + \log f(\mathbf{x}|\theta)$$

By Jensen's inequality,

$$\begin{aligned}
&\leq \log \left[\int_z \left\{ \frac{f(\mathbf{x}, \mathbf{z}|\theta)\, f(\mathbf{x}|\theta_{(k)})}{f(\mathbf{x}, \mathbf{z}|\theta_{(k)}) f(\mathbf{x}|\theta)} \right\} f(\mathbf{z}|\mathbf{x}, \theta_{(k)}) \right] + \log f(\mathbf{x}|\theta) \\
&= \log \left[\int_z \frac{f(\mathbf{z}|\mathbf{x}, \theta)}{f(\mathbf{z}|\mathbf{x}, \theta_{(k)})} f(\mathbf{z}|\mathbf{x}, \theta_{(k)}) \right] + \log f(\mathbf{x}|\theta) \\
&= \log \int_z f(\mathbf{z}|\mathbf{x}, \theta) dz + \log f(\mathbf{x}|\theta) \\
&= \log f(\mathbf{x}|\theta)\,.
\end{aligned}$$

Thus, $\tilde{f}(\theta|\theta_{(k)})$ is a minorizing function, and the next iterate is

$$\theta_{(k+1)} = \arg\max_{\theta} \tilde{f}(\theta|\theta_{(k)}) = \arg\max_{\theta} q(\theta|\theta_{(k)})$$

□

MTH 511a - 2021: L25 - EM Algorithm Continued

Instructor: Dootika Vats

1 The EM algorithm

Recall, the objective function to maximize is

$$l(\theta|X) = \log \int f(\mathbf{x}, \mathbf{z}|\theta) d\nu_z \,,$$

The EM algorithm iterates through the following: Consider a starting value θ_0. Then for any $k+1$ iteration

1. **E-Step**: Compute

$$q(\theta; \theta_{(k)}) = \mathrm{E}_{Z|x}\Big[\log f(\mathbf{x}, \mathbf{z}|\theta) \mid X = x, \theta_{(k)} \Big]$$

   where the expectation is computed with respect to the conditional distribution of Z given $X = x$ for the current iterate $\theta_{(k)}$.

2. **M-Step**: Compute

$$\theta_{k+1} = \arg\max_{\theta \in \Theta} q(\theta; \theta_k) \,.$$

We will look at another EM algorithm example.

1.1 Censored data example

A light bulb company is testing the failure times of their bulbs and know that failure times follow Exp(λ) for some $\lambda > 0$. They test n light bulbs, so the failure time of each light bulb is

$$Z_1, \ldots, Z_n \overset{iid}{\sim} \text{Exp}(\lambda)\,.$$

However, officials recording these failure times walked into the room only at time T and observed that $m < n$ of the bulbs had already failed. Their failure time cannot be recorded. So observed data is

$$E_1 = 1, \ldots, E_m = 1, Z_{m+1}, Z_{m+2}, \ldots, Z_n\,,$$

where $E_i = I(Z_i < T)$. Our goal is to find the MLE for λ. Note that

- If we ignore the first m light bulbs, then not only do we have a smaller sample size, but we also have a biased sample which do not contain the bottom tail of the distribution of failure times.

So we must account for the "missing data". First, we find the observed data likelihood.

Note that $E_i \sim \text{Bern}(p)$ where $p = \Pr(E_i = 1) = \Pr(Z_i \leq T) = 1 - e^{-\lambda T}$ (from the CDF of an exponential distribution). So,

$$\begin{aligned}
L(\lambda|E_1, \ldots, E_m, Z_{m+1}, \ldots, Z_n) &= \prod_{i=1}^{m} \Pr(E_i = e_i) \cdot \prod_{j=m+1}^{n} f(z_j|\lambda) \\
&= \prod_{i=1}^{m} \left(e^{-\lambda T}\right)^{1-e_i} \left(1 - e^{-\lambda T}\right)^{e_i} \prod_{j=m+1}^{n} \lambda \exp\left\{-\lambda Z_j\right\} \\
&= (1 - e^{-\lambda T})^m \lambda^{n-m} \exp\left\{-\lambda \sum_{j=m+1}^{n} Z_j\right\}\,.
\end{aligned}$$

Closed-form MLEs are difficult here, and some sort of numerical optimization is useful. We will resort to the EM algorithm, that has the added advantage that the estimates of $Z_1, \ldots, Z_m$ may be obtained as well.

Also, note that if we choose to "throw away" the censored data, then the likelihood is

$$L_{bad}(\lambda|Z_{m+1} \ldots Z_n) = \lambda^{n-m} \exp\left\{-\lambda \sum_{j=m+1}^{n} Z_j\right\}$$

and the MLE is

$$\lambda_{\text{MLE, bad}} = \frac{n-m}{\sum_{j=m+1}^{n} Z_j}$$

The above MLE is a bad estimator since the data thrown away is not censored at random, and in fact, all those bulbs fused early. So the bulb company cannot just throw that data away, as that would be dishonest!

Now we implement the EM algorithm for this example. First, note that the complete unobserved data is $Z_1, \ldots, Z_n$ and the complete log likelihood is

$$\log f(z|\lambda) = \log \left\{ \prod_{i=1}^{n} \lambda e^{-\lambda z_i} \right\} = n \log \lambda - \lambda \sum_{i=1}^{n} Z_i \,.$$

In order to implement the EM algorithm, we need the distribution of

$$\begin{aligned}
\Pr(\text{unobserved} \mid \text{observed}) &= f(Z_1, \ldots, Z_m \mid E_1, \ldots, E_m, Z_{m+1}, \ldots, Z_n) \\
&= f(Z_1, \ldots, Z_m \mid E_1, \ldots, E_m) \\
&= \prod_{i=1}^{m} f(Z_i \mid E_i) \\
&= \prod_{i=1}^{m} f(Z_i \mid Z_i \le T) \\
&= \prod_{i=1}^{m} \frac{\lambda e^{-\lambda z_i}}{1 - e^{-\lambda T}} \mathbb{I}(z_i \le T) \,.
\end{aligned}$$

Further,

$$\begin{aligned}
\mathrm{E}\left[Z_i | E_i = 1\right] &= \mathrm{E}\left[Z_i | Z_i \le T\right] \\
&= \int_0^T z_i \frac{\lambda e^{-\lambda z_i}}{1 - e^{-\lambda T}} = \cdots = \frac{1}{\lambda} - \frac{T e^{-\lambda T}}{1 - e^{-\lambda T}} \,.
\end{aligned}$$

Implementing the EM steps now

1. **E-Step**: In the E-step, we find the expectation of the complete log likelihood under $Z_{1:m}|(E_{1:m}, Z_{(m+1):n})$. That is

$$\begin{aligned}
q(\lambda \mid \lambda_{(k)}) &= \mathrm{E}\Big[\log f(Z_1, \ldots, Z_n|\lambda) \mid E_1, \ldots, E_m, Z_{m+1}, \ldots, Z_n \Big] \\
&= n \log \lambda - \lambda \mathrm{E}_{\lambda_{(k)}} \left[\sum_{i=1}^{n} Z_i | E_1 = 1, \ldots, E_m = 1, Z_{m+1}, \ldots, Z_n \right]
\end{aligned}$$

$$= n \log \lambda - \lambda \sum_{i=m+1}^{n} Z_i - \lambda \sum_{i=1}^{m} [\mathrm{E}[Z_i | Z_i \leq T]$$

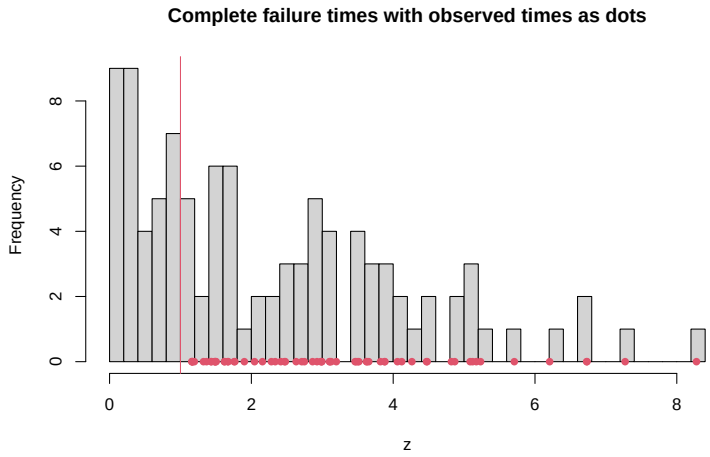
2. **M-Step**: To implement the M-step:

$$\lambda_{(k+1)} = \arg\max_{\lambda} \left[n \log \lambda - \lambda \sum_{i=m+1}^{n} Z_i - \lambda \sum_{i=1}^{m} [\mathrm{E}[Z_i | Z_i \leq T] \ . \right]$$

It is then easy to show that the M step makes the following update (show by yourself):

$$\lambda_{(k+1)} = \frac{n}{\sum_{i=m+1}^{n} Z_i + \sum_{i=1}^{m} [\mathrm{E}[Z_i | Z_i \leq T]} = \frac{n}{\sum_{i=m+1}^{n} Z_i + m \left[\frac{1}{\lambda_{(k)}} - \frac{T e^{-\lambda_{(k)} T}}{1 - e^{-\lambda_{(k)} T}} \right]}$$

```
################################################
## EM Algorithm for the Censored exponential data
################################################

# First we will simulate data
set.seed(10)
n <- 100
lam <- .45
z <- rexp(n, rate = lam)
T <- 1

# Observed
obs.z <- z[(z > T)]
m <- n - length(obs.z) # to find m
hist(z, breaks = 30, main = "Complete failure times with observed times as
    dots")
points(obs.z,rep(0, length(obs.z)), col = 2, pch = 16)
abline(v = 1, col = 2)
```




```
# EM Algorithm

diff <- 100
tol <- 1e-5
iter <- 0
current <- 1/mean(obs.z) # good starting values
lam.k <- current
store.z <- T
while(diff > tol)
{
  iter <- iter + 1
  # E-step
   Estep <- 1/current - T*exp(-current * T)/(1 - exp(-current * T))

   # Storing Zs
  store.z <- c(store.z, Estep)

  #Update
  update <- n/(sum(obs.z) + m*Estep)

  diff <- abs(current - update)
  current <- update
  lam.k <- c(lam.k, update)
}

current
```

```r
# [1] 0.442948

(false.mle <- (n-m)/sum(obs.z)) # MLE if you ignore the Es (bad MLE)
# [1] 0.3142692

# Estimate of Z_i not observed
(Z_unobs <- 1/current - T*exp(-current * T)/(1 - exp(-current * T)))
# [1] 0.4632078
```

1.2 Monte Carlo EM

Sometimes, the E-step is not analytically tractable, in which case the following step is not obtainable.

$$q(\theta|\theta_{(k)}) = \mathrm{E}_{\theta_{(k)}}\left[\log f(x, z|\theta) \mid X\right]$$

Instead, we *estimate this expectation* using draws from $Z|X$ via Monte Carlo. That is, the new E-step is

$$\hat{q}(\theta|\theta_{(k)}) = \frac{1}{m}\sum_{t=1}^{m} \log(f(x, z^{(t)})|\theta)\,,$$

where $z^{(t)}$ are drawn from $Z|X$.

Example 1 (Censored Exponential). Although EM is possible in this example, we can choose to do MCEM. The original steps are:

1. **E-Step**:

$$q(\lambda \mid \lambda_{(k)}) = n\log\lambda - \lambda\sum_{i=m+1}^{n} Z_i - \lambda\sum_{i=1}^{m}\mathrm{E}_{\lambda_{(k)}}[Z_i|Z_i \le T]$$

2. **M-Step**: It is then easy to show that the M step makes the following update:

$$\lambda_{(k+1)} = \frac{n}{\sum_{i=m+1}^{n} Z_i + m\sum_{i=1}^{m}\mathrm{E}_{\lambda_{(k)}}[Z_i|Z_i \le T]}$$

Instead, we can estimate $\mathrm{E}_{\lambda_{(k)}}[Z_i|Z_i \le T]$ using Monte Carlo:

1. **MCE-Step**: Draw $z^{(1)} = (z_1^{(1)}, \ldots z_m^{(1)}), \ldots, z^{(K)}$ from $Z|Z \le T$ (from a trun-

cated exponential).

$$q(\lambda \mid \lambda_{(k)}) = n \log \lambda - \lambda \sum_{i=m+1}^{n} Z_i - \lambda \sum_{i=1}^{m} \left(\frac{1}{K} \sum_{t=1}^{K} z_i^{(t)} \right) .$$

2. **M-Step**: It is then easy to show that the M step makes the following update:

$$\lambda_{(k+1)} = \frac{n}{\sum_{i=m+1}^{n} Z_i + \left(\frac{1}{K} \sum_{t=1}^{K} z_i^{(t)} \right)} .$$

```
################################################
## Monte Carlo EM Algorithm for the
## Censored exponential data
################################################
set.seed(1)
#function draws from truncated exponential
trexp <- function(nsim, T, lambda)
{
  count <- 1
  samp <- numeric(length = nsim)
  while(count < nsim)
  {
    draw <- rexp(1, rate = lambda)
    if(draw <= T)
    {
      samp[count] <- draw
      count <- count + 1
    } else{
      next
    }
  }
  return(samp)
}

diff <- 100
tol <- 1e-5
iter <- 0
```

```r
current <- 2
lam.k <- current
store.z <- T
mc.size <- 100 # Monte Carlo size
while(diff > tol)
{
  iter <- iter + 1

  # E-step with Monte Carlo

  trunExp.samples <- trexp(nsim = mc.size, T = T, lambda = current)
  Estep <- mean(trunExp.samples)
  #Estep <- 1/current - T*exp(-current * T)/(1 - exp(-current * T))

  # Storing Zs
  store.z <- c(store.z, Estep)

  #Update
  update <- n/(sum(obs.z) + m*Estep)

  diff <- abs(current - update)
  current <- update
  lam.k <- c(lam.k, update)
}

current
# [1] 0.4460627
```

2 Questions to think about

- In Monte Carlo EM, how can you improve the expectation step?
- What is an added benefit of doing EM rather than NR or Gradient Ascent on the observed likelihood?

MTH 511a - 2021: L26: Loss Functions

Instructor: Dootika Vats

1 Resampling Methods

We will take a break from optimization methods to discuss methods for making *inference* in some of the data analysis problems we have discussed.

There are two main questions to ask:

1. *How do we choose model tuning parameters?*: In ridge/bridge/lasso regression, we require choosing λ and/or α. In Gaussian mixture models, we need to choose the number of clusters C.

2. *How good are our estimates?*: MLEs usually yield asymptotic normality for our estimates, so that that

$$\sqrt{n}(\hat{\theta}_{\text{MLE}} - \theta) \xrightarrow{d} N(0, \Sigma_{\text{MLE}}) \,,$$

   where Σ_{MLE} is the asymptotic covariance matrix. If we estimate Σ_{MLE}, we can assess the large-sample error in $\hat{\theta}_{\text{MLE}}$.

   However, for estimators that are not MLEs, like bridge regression, an asymptotic distribution is difficult to construct. So we need a method to assess understand the distribution of estimators.

Both points above can be addressed by *resampling* techniques which we will cover in this week. We will cover two resampling methods each to address two different concepts: *model selection* and *model performance*. Although both the methods can be used to address both things, we will focus on

- **Cross-validation:** for model selection – choosing tuning parameters
- **Bootstrapping:** to estimate model performance – obtain empirical distributions of estimators.

Cross-validation will be used to find the best tuning parameters. However, before we can proceed, we need to understand how to assess the quality of a model. First, we will learn a little about *loss functions*.

1.1 Loss functions

A loss function is a measure of error of an estimator from the true value. Having observed data, y, let $\hat{f}$ be the model fit. Then a loss function is a function $L(y, \hat{f})$ that quantifies the *distance* between y and $\hat{f}$. The form of the loss function may depend on the type of data. We will focus only on binary/Gaussian regression and the Gaussian mixture models.

- **Linear regression:** In penalized or non-penalized regression, we have an observation y, covariates X, and a regression coefficient, β. Let $\hat{\beta}$ be the estimated regression coefficient – this could be obtained via ridge, bridge, or regular regression. Then the estimated response is

  $$\hat{f}(x) = x^T \hat{\beta}\,.$$

  A loss function measures the error between y and the estimated response $\hat{y} = \hat{f}(x)$. For continuous response y, there are two popular loss functions:

  - Squared error: $L(y, \hat{f}(x)) = (y - \hat{f}(x))^2$
  - Absolute error: $L(y, \hat{f}(x)) = |y - \hat{f}(x)|$

  We will focus mainly on the squared error loss.

- **Binary data classification**: For binary data models, like logistic regression, a popular loss function is the *misclassification* also known as the $0-1$ loss. Given

response y which are Bernoulli(p) where

$$p = \frac{e^{X\beta}}{1 + e^{X\beta}},$$

we can find the estimated p, $\hat{p}$ by setting

$$\hat{p} = \frac{e^{X\hat{\beta}_{\text{MLE}}}}{1 + e^{X\hat{\beta}_{\text{MLE}}}},$$

where $\hat{\beta}_{\text{MLE}}$ are the MLE estimates obtained using an optimization algorithm. These $\hat{p}$ are the estimated probability of success. Set $\hat{f}(x) = 1 \cdot \mathbb{I}\{\hat{p} \geq .5\} + 0 \cdot \mathbb{I}\{\hat{p} < .5\}$. (The cutoff, .5, is the default cutoff, but can be changed depending on the problem). Then the *misclassification* loss function checks whether the model has misclassified the observation

$$\text{Misclassification or } 0-1 \text{ loss} : L(y, \hat{f}(x)) = \mathbb{I}(y \neq \hat{f}(x)).$$

- **Gaussian mixture model:** Where there are no "response" and "covariates", like the Gaussian mixture model (GMM) example, it is difficult to implement the above two loss functions. But recall, that in the GMM example, we used the log-likelihood to see whether our estimates were good. Thus, we can use negative log-likelihood to see how bad our estimates are. Thus, here the loss function is:

$$L(x, \hat{\theta}) = -\log f(x|\hat{\theta}).$$

  We will discuss the specific GMM case in more detail later.

How do we use these loss functions?

Given a dataset, we are interested in estimating the *test error* which is the expected loss, of an independent dataset given estimates from the current dataset. If our given dataset is $\mathcal{D}$

$$\text{Err}_{\mathcal{D}} = \text{E}\left[L(y, \hat{f}(x)) \mid \mathcal{D}\right],$$

where $\hat{f}(x)$ denotes the estimated y for a new independent dataset, given estimators from the current dataset. For example, using $\hat{\beta}$ from a given dataset $\mathcal{D}$, then $\hat{f}(x) = X_{\text{new}}\hat{\beta}$.

Test error is built for a given dataset $\mathcal{D}$. We are interested in the *expected prediction*

error which is the expected test error over all such $\mathcal{D}$:

$$\text{Err} = \text{E}\left[L(y, \hat{f}(x))\right] = \text{E}\left[\text{E}\left[L(y, \hat{f}(x)) \mid \mathcal{D}\right]\right] = \text{E}\left[\text{Err}_{\mathcal{D}}\right]$$

The expected prediction error is the expected error in predicting y for new datasets given models built from any dataset.

So, given different values of a tuning parameter, our goal is to compare the prediction error, and choose the tuning parameter which yields the lowest test error. But, the test error requires obtaining the loss on *independent datasets*. We just have one dataset available to use.

This is where cross-validation is critically useful.

1.2 Questions to think about

- What happens to the log-likelihood for Gaussian mixture model if we keep increasing C?

MTH511a: EM Algorithm for Multivariate Gaussian Mixture Model

Dootika Vats

22/10/2020

The problem statement

In class, we did the univariate Gaussian mixture model example. In this document, we go through the steps for the multivariate Gaussian mixture model. Note that, fundamentally, the steps are exactly the same, however, there are some additional nuances in the M-step and the practical implementations are important to consider.

Suppose for some C classes/clusters/components, we observe $X_1, \ldots, X_n$ from a mixture of multivariate normal distributions. The joint distribution of the observed $\mathbf{x}$ is:

$$f(\mathbf{x}|\theta) = \sum_{c=1}^{C} \pi_c f_1(\mathbf{x}|\mu_c, \Sigma_c)$$

where $\mu_c \in \mathbb{R}^p$, $\Sigma_c \in \mathbb{R}^{p\times p}$ and

$$f_c(\mathbf{x} \mid \mu_c, \sigma_c^2) = \left(\frac{1}{2\pi}\right)^{p/2} \frac{1}{|\Sigma_c|^{1/2}} \exp\left\{-\frac{(\mathbf{x}-\mu_c)^T\Sigma_c^{-1}(\mathbf{x}-\mu_c)}{2}\right\}.$$

Similar to the one dimensional case, set up the EM algorithm for this p-dimensional case, and then implement this on the Old Faithful dataset.

Introduce latent variable Z_i such that

$$Z_i = c \text{ if } X_i \text{ is in class } c\,.$$

EM Algorithm

The E-step is to find q:

$$q(\theta|\theta_{(k)}) = \mathbb{E}\left[\log f(\mathbf{x}, \mathbf{z}|\theta) \mid X = \mathbf{x}, \theta = \theta_{(k)}\right]$$

$$\begin{aligned}
q(\theta|\theta_{(k)}) &= \mathbb{E}_{Z|x}\left[\log f(x, z|\theta) \mid X = x, \theta_{(k)}\right] \\
&= \mathbb{E}_{Z|x}\left[\sum_{i=1}^{n} \log f(x_i, z_i|\theta) \mid X = x, \theta_{(k)}\right] \\
&= \sum_{i=1}^{n} \mathbb{E}_{Z_i|x_i}\left[\log f(x_i, z_i|\theta) \mid X = x_i, \theta_{(k)}\right] \\
&= \sum_{i=1}^{n}\sum_{c=1}^{C} \log\left\{f_c(x_i|\mu_c, \Sigma_c)\pi_c\right\} \frac{f_c(x_i|\mu_{c,k}, \Sigma_{c,k})\pi_{c,k}}{\sum_{j=1,2} f_j(x_i|\mu_{j,k}, \Sigma_{j,k})\pi_{j,k}} .
\end{aligned}$$

For the E-step above, we need to find

$$\Pr(Z = c|X = x_i, \theta = \theta_{(k)}) = \frac{f(x_i|Z = c, \theta = \theta_{(k)})\Pr(Z = c)}{f(x_i)} = \frac{f_c(x_i|\mu_{c,k}, \Sigma_{c,k})\pi_{c,k}}{\sum_{j=1}^{C} f_j(x_i|\mu_{j,k}, \Sigma_{j,k})\pi_{j,k}} := \gamma_{i,c,k} .$$

And then in the M-step, we do the maximization step:

$$q(\theta|\theta_{(k)}) = \text{const} - \frac{1}{2}\sum_{i=1}^{n}\sum_{c=1}^{C} \log|\Sigma_c|\gamma_{i,c,k} - \sum_{i=1}^{n}\sum_{c=1}^{C} \frac{(x_i - \mu_c)\Sigma_c^{-1}(x_i - \mu_c)^T}{2}\gamma_{i,c,k} + \sum_{i=1}^{n}\sum_{c=1}^{C} \log \pi_c \gamma_{i,c,k} .$$

Taking derivative with respect to μ_c

$$\begin{aligned}
\frac{\partial q}{\partial \mu_c} &= -\sum_{i=1}^{n} \Sigma_c^{-1}(x_i - \mu_c)\gamma_{i,c,k} \overset{set}{=} 0 \\
\Rightarrow \mu_{c,(k+1)} &= \frac{\sum_{i=1}^{n} \gamma_{i,c,k}\ x_i}{\sum_{i=1}^{n} \gamma_{i,c,k}}
\end{aligned}$$

Taking derivatives with respect to a matrix, Σ_c is tricky. But the following properties will be useful to remember for compatible matrices A, B, C

- $\text{tr}(ACB) = \text{tr}(CAB) = \text{tr}(BC)$
- for scalar x, $x^T Ax = \text{tr}(x^T Ax) = \text{tr}(x^T xA)$
- $\frac{\partial}{\partial A}\text{tr}(AB) = B^T$
- $\frac{\partial}{\partial A}\log(|A|) = A^{-T}$

Using these properies and taking derivative w.r.t Σ_c^{-1} (for convenience), we get

$$\begin{aligned}
\frac{\partial q}{\partial \Sigma_c^{-1}} &= -\frac{n}{2}\Sigma_c \gamma_{i,,c,k} - \frac{\partial}{\partial \Sigma_c^{-1}}\left[\sum_{i=1}^{n} \frac{\text{tr}((x_i-\mu_c)^T(x_i-\mu_c)\Sigma_c^{-1})}{2}\gamma_{i,c,k}\right] \\
&= -\frac{n}{2}\Sigma_c \gamma_{i,,c,k} - \left[\sum_{i=1}^{n} \frac{(x_i-\mu_c)^T(x_i-\mu_c)\gamma_{i,c,k}}{2}\right] \stackrel{set}{=} 0 \\
\Rightarrow \Sigma_{c,(k+1)} &= \frac{\sum_{i=1}^{n} \gamma_{i,c,k}(x_i-\mu_{c,(k+1)})(x_i-\mu_{c,(k+1)})^T}{\sum_{i=1}^{n}\gamma_{i,c,k}}.
\end{aligned}$$

The optimization for π_c is the same as the univariate case. Hence, we get the following updates:

$$\mu_{c,(k+1)} = \frac{\sum_{i=1}^{n} \gamma_{i,c,k}\ x_i}{\sum_{i=1}^{n}\gamma_{i,c,k}} \tag{1}$$

$$\Sigma_{c,(k+1)} = \frac{\sum_{i=1}^{n} \gamma_{i,c,k}(x_i-\mu_{c,(k+1)})(x_i-\mu_{c,(k+1)})^T}{\sum_{i=1}^{n}\gamma_{i,c,k}} \tag{2}$$

$$\pi_{c,(k+1)} = \frac{1}{n}\sum_{i=1}^{n}\gamma_{i,c,k} \tag{3}$$

Some practical considerations

Here are some practical things to keep in mind

- **Multivariate normal density**: Given a starting value, the algorithm updates the components using the above update equations. Notice that we need to evaluate the density of a multivariate normal. You can use ‘mvtnorm‘ package and ‘dmvnorm‘ function for that.
- **Local solutions**: Further note that the target log-likelihood is not concave. Thus, there will be multiple local maximas. The EM algorithm is an MM algorithm, so it does converge to a local maxima, but it need not converge to a global maxima. In order to try to ensure that we reach a reasonable maxima, it will be useful to start from multiple starting points and see where you converge. Then note down the log-likelihood and compare the log-likehood with other runs from different starting points. The optima with the largest log-likelihood should be the chosen maxima.
- **Starting values**: In addition to testing this on multiple starting values, one must also make sure the starting values are realistic. That is, if most of the data is centered near (50,100), then it is unreasonable to start at (0,0), and infact such starting values will lead to numerical inaccuracies. Additionally, if your starting values for the μ_c are the same for all c, then all normal distribution components are centered

at the same spot and this leads to only one cluster, even if you have specified $C > 1$.

- **Lack of positive definiteness**: In the update for Σ_c, note that we are calculating a covariance matrix. Particularly when C is chosen to be larger than the actual number of clusters, an iteration value of $\Sigma_{c,(k+1)}$ may yield singular matrices. To correct for this, a common solution is to set

$$\Sigma_{c,(k+1)} = \frac{\sum_{i=1}^{n} \gamma_{i,c,k}(x_i - \mu_{c,(k+1)})(x_i - \mu_{c,(k+1)})^T}{\sum_{i=1}^{n} \gamma_{i,c,k}} + \text{diag}(\epsilon, p)\,.$$

  That is, we add a diagonal matrix with small entries ϵ in order to make sure it's positive definite.

- In order to declare convergence, we will monitor the change in the log-likelihood values.

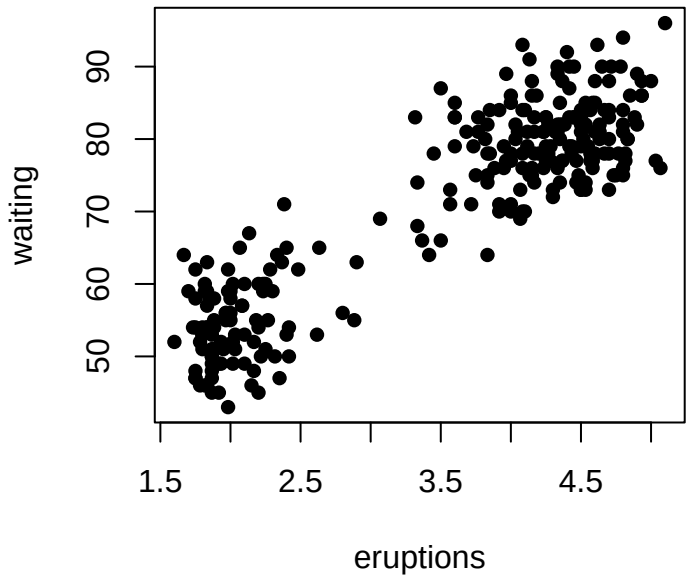
Old daithful dataset

```
set.seed(10)
library(mvtnorm)
library(ellipse)
```

```
##
## Attaching package: 'ellipse'
## The following object is masked from 'package:graphics':
##
##     pairs
```

We now continue a bivariate analysis of the old faithful dataset

```
data(faithful)
plot(faithful, pch = 16)
```



Below are the functions that calculate the log-likelihood and that run the EM algorithm for any p and any C.

```
# calculate log-likelihood
# of mixture of multivariate normal
log_like <- function(X, pi.list, mu.list, Sigma.list, C)
{
  foo <- 0
  for(c in 1:C)
  {
    foo <- foo + pi.list[[c]]*dmvnorm(X, mean = mu.list[[c]], sigma = Sigma.list[[c]])
  }
  return(sum(log(foo)))
}


# X is the data
# C is the number of clusters
mvnEM <- function(X, C = 2, tol = 1e-5, maxit = 1e3)
{
  n <- dim(X)[1]
  p <- dim(X)[2]
  ######## Starting values ####################
  ## pi are equally split over C
```

```r
  pi.list <- rep(1/C, C)

  mu <- list()  # list of all the means
  Sigma <- list()  # list of all variances

  # The means for each C cannot be the same,
  # since then the three distributions overlap
  # Hence adding random noise to colMeans(X)
  for(i in 1:C)
  {
    # cannot start with the same mean
    # for each component. Thus adding
    # random noise to the starting means
    # this also encourages convergence to
    # different solutions
    mu[[i]] <-  rnorm(p, sd = 3) + colMeans(X)

    # another way to choose the starting means
    # mu[[i]] <-  rnorm(p, sd = apply(X, 2, sd)) + colMeans(X)

    Sigma[[i]] <- var(X)
  }
  # Choosing good starting values is important since
  # The GMM likelihood is not concave, so the algorithm
  # may converge to a local optima.

######## EM algorithm steps ###################

  iter <- 0
  diff <- 100
  old.mu <- mu
  old.Sigma <- Sigma
```

```
old.pi <- pi.list
epsilon <- 1e-5

Ep <- matrix(0, nrow = n, ncol = C)  # gamma_{i,c}
save.loglike <- 0
while((diff > tol) && (iter < maxit) )
{
  iter <- iter + 1
  ## E step: find gammas
  for(c in 1:C)
  {
    Ep[ ,c] <- pi.list[c]*apply(X, 1, dmvnorm , mu[[c]], Sigma[[c]])
  }
  Ep <- Ep/rowSums(Ep)

  ### M-step
  pi.list <- colMeans(Ep)
  for(c in 1:C)
  {
    mu[[c]] <- colSums(Ep[ ,c] * X )/sum(Ep[,c])
  }

  for(c in 1:C)
  {
    foo <- 0
    for(i in 1:n)
    {
      foo <- foo + (X[i, ] - mu[[c]]) %*% t(X[i, ] - mu[[c]]) * Ep[i,c]
    }
    Sigma[[c]] <- foo/sum(Ep[,c])

    if(min(eigen(Sigma[[c]])$values) <=0)
```

```
      {
        # Below is to ensure the estimator is positive definite
        # otherwise next iteration gamma_i,c,k cannot be calculated
        Sigma[[c]] <- Sigma[[c]] + diag(epsilon, p)
        print("Matrix not positive-definite")
      }
    }

    save.loglike <- c(save.loglike,
                      log_like(X = X, pi.list = pi.list, mu.list = mu, Sigma.list = Sigma,  C = C))
    # Difference in the log-likelihoods as the difference criterion
    diff <- abs(save.loglike[iter+1] - save.loglike[iter])

    old.mu <- mu
    old.Sigma <- Sigma
    old.pi <- pi.list
  }

  # Final allocation updates
  for(c in 1:C)
  {
    Ep[ ,c] <- pi.list[c]*apply(X, 1, dmvnorm , mu[[c]], Sigma[[c]])
  }
  Ep <- Ep/rowSums(Ep)

  return(list("pi" = pi.list, "mu" = mu, "Sigma" = Sigma, "Ep" = Ep,
              "log.like" = tail(save.loglike,1)))
}
```

Now for the old faithful dataset, we will try and fit $C = 2$ clusters. Since the starting values can lead to different local optimas, the random starting values of the μ_c chosen up are incredibly helpful. They allow each run to possibly converge to different local solutions. We will then choose the one with the largest log-likelihood

```r
# We now fit this to the
data(faithful)

X <- as.matrix(faithful)

C <- 2
class2.1 <- mvnEM(X = X, C = C)
```

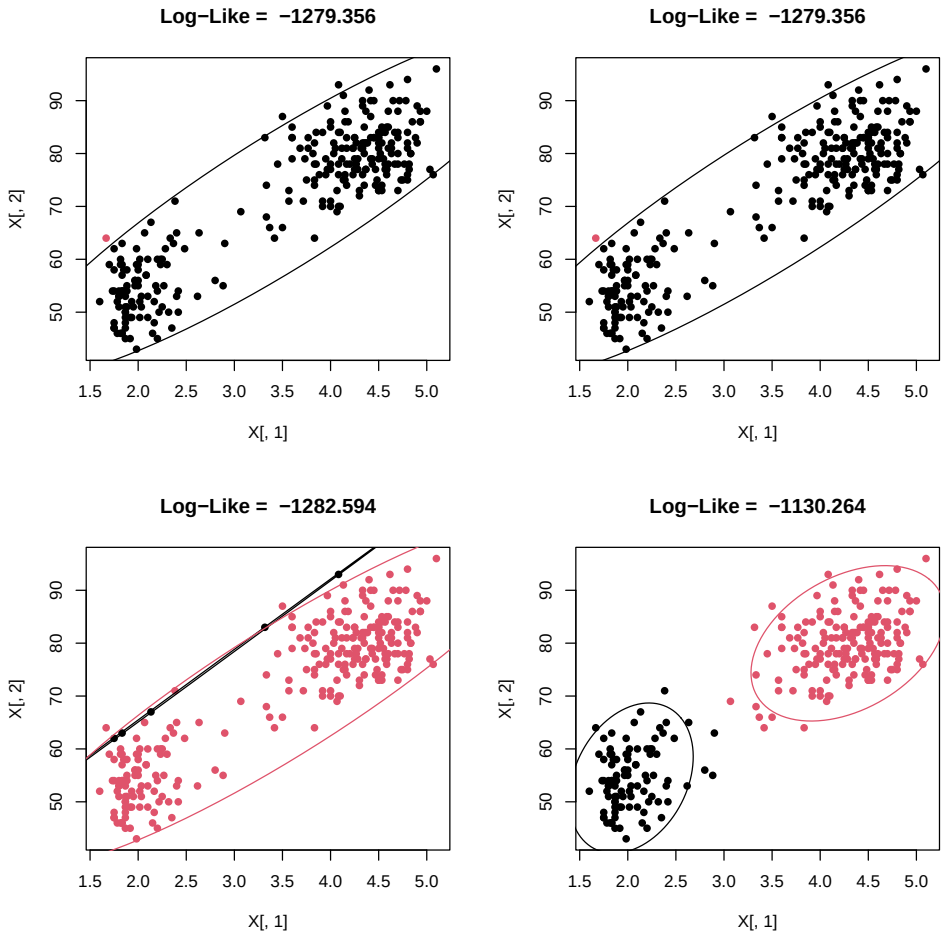
```
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
```

```r
class2.2 <- mvnEM(X = X, C = C)
```

```
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
```

```r
class2.3 <- mvnEM(X = X, C = C)
class2.4 <- mvnEM(X = X, C = C)

# model 1,2,3 converge to a local maxima with
# log-likelihood values smaller
# model 4 converges to a point with the highest
# log-likelihood value. Hence choose model 4

par(mfrow = c(2,2))
for(i in 1:4)
{
  model <- get(paste("class2.",i, sep = ""))
  allot <- apply(model$Ep, 1, which.max)  ## Final allotment of classification
  plot(X[,1], X[,2], col = allot, pch = 16,
       main = paste("Log-Like = ", round(model$log.like,3))) # plot allotment

  ell <- list()
  for(c in 1:C)
```

```
  {
    ell[[c]] <- ellipse(model$Sigma[[c]], centre = as.numeric(model$mu[[c]]))
    lines(ell[[c]], col = c)
  }
}
```



Of the above different convergences, we converge to the largest log-likelihood values in Model 4, so that is the output we will treat as the MLE.

Now we will repeat the same with $C = 3$. Note that $C = 3$ classes is slightly more annoying to fit since it looks like naturally, there are only $C = 2$ classes. Thus, we will need to work harder to find good local maximas. We will run from 6 different (randomly chosen) starting values.

```
C <- 3


class3.1 <- mvnEM(X = X, C = C)
```

```r
class3.2 <- mvnEM(X = X, C = C)
class3.3 <- mvnEM(X = X, C = C)
```

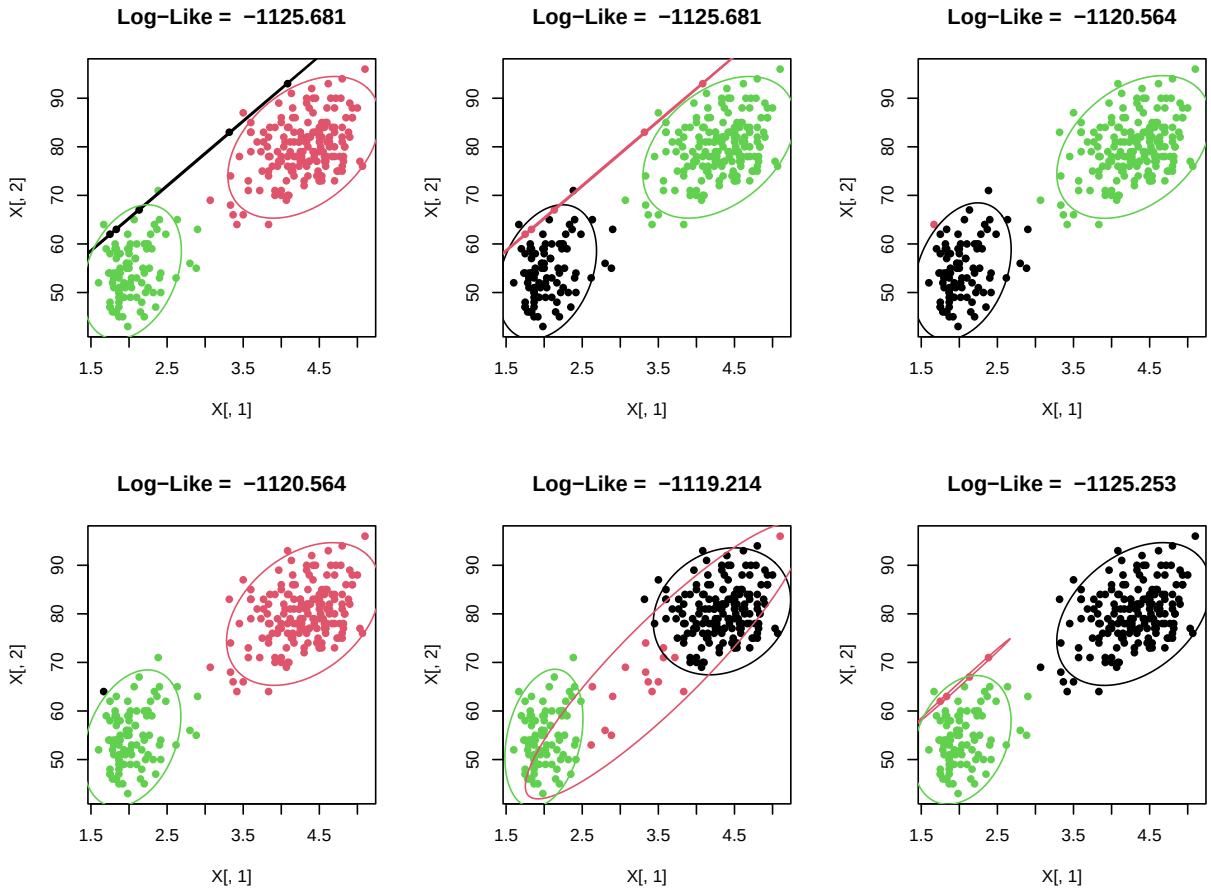
```
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
```

```
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
```

```
class3.4 <- mvnEM(X = X, C = C)
```

```
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
## [1] "Matrix not positive-definite"
```

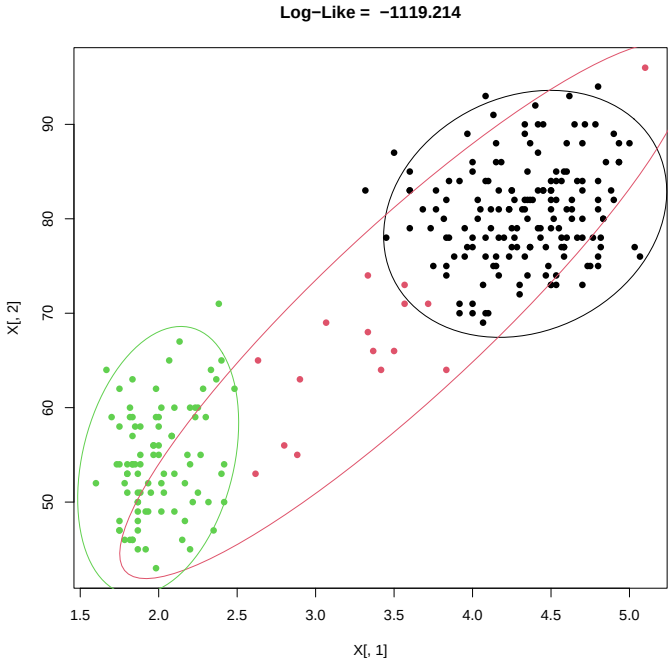
```r
class3.5 <- mvnEM(X = X, C = C)
class3.6 <- mvnEM(X = X, C = C)

par(mfrow = c(2,3))
for(i in 1:6)
{
  model <- get(paste("class3.",i, sep = ""))
  allot <- apply(model$Ep, 1, which.max)  ## Final allotment of classification
  plot(X[,1], X[,2], col = allot, pch = 16,
       main = paste("Log-Like = ", round(model$log.like,3))) # plot allotment

  ell <- list()
  for(c in 1:C)
  {
    ell[[c]] <- ellipse(model$Sigma[[c]], centre = as.numeric(model$mu[[c]]))
    lines(ell[[c]], col = c)
  }
}
```



The fifth run has the largest log-likelihood so that is chosen. Here I present the final allocation. This is a reminder that we are not fitting different models, we are just running the same algorithm multiple times in order to find the largest local maxima. It's possible our final estimate corresponds to a local maxima, and not a global maxima.

```
par(mfrow = c(1,1))


model <- class3.5
allot <- apply(model$Ep, 1, which.max)  ## Final allotment of classification
plot(X[,1], X[,2], col = allot, pch = 16,
     main = paste("Log-Like = ", round(model$log.like,3))) # plot allotment


ell <- list()
for(c in 1:C)
{
  ell[[c]] <- ellipse(model$Sigma[[c]], centre = as.numeric(model$mu[[c]]))
```

```
  lines(ell[[c]], col = c)
}
```



MTH 511a - 2021: L27: Cross-validation

Instructor: Dootika Vats

1 Resampling Methods

We will take a break from optimization methods to discuss methods for making *inference* in some of the data analysis problems we have discussed.

There are two main questions to ask:

1. *How do we choose model tuning parameters?*: In ridge/bridge/lasso regression, we require choosing λ and/or α. In Gaussian mixture models, we need to choose the number of clusters C.

2. *How good are our estimates?*: MLEs usually yield asymptotic normality for our estimates, so that that

$$\sqrt{n}(\hat{\theta}_{\text{MLE}} - \theta) \xrightarrow{d} N(0, \Sigma_{\text{MLE}}) \,,$$

   where Σ_{MLE} is the asymptotic covariance matrix. If we estimate Σ_{MLE}, we can assess the large-sample error in $\hat{\theta}_{\text{MLE}}$.

   However, for estimators that are not MLEs, like bridge regression, an asymptotic distribution is difficult to construct. So we need a method to assess understand the distribution of estimators.

Both points above can be addressed by *resampling* techniques which we will cover in this week. We will cover two resampling methods each to address two different concepts: *model selection* and *model performance*. Although both the methods can be used to address both things, we will focus on

- **Cross-validation:** for model selection – choosing tuning parameters
- **Bootstrapping:** to estimate model performance – obtain empirical distributions of estimators.

Cross-validation will be used to find the best tuning parameters. However, before we can proceed, we need to understand how to assess the quality of a model. First, we will learn a little about *loss functions*.

1.1 Loss functions

A loss function is a measure of error of an estimator from the true value. Having observed data, y, let $\hat{f}$ be the model fit. Then a loss function is a function $L(y, \hat{f})$ that quantifies the *distance* between y and $\hat{f}$. The form of the loss function may depend on the type of data. We will focus only on binary/Gaussian regression and the Gaussian mixture models.

- **Linear regression:** In penalized or non-penalized regression, we have an observation y, covariates X, and a regression coefficient, β. Let $\hat{\beta}$ be the estimated regression coefficient – this could be obtained via ridge, bridge, or regular regression. Then the estimated response is

  $$\hat{f}(x) = x^T \hat{\beta}\,.$$

  A loss function measures the error between y and the estimated response $\hat{y} = \hat{f}(x)$. For continuous response y, there are two popular loss functions:

  - Squared error: $L(y, \hat{f}(x)) = (y - \hat{f}(x))^2$
  - Absolute error: $L(y, \hat{f}(x)) = |y - \hat{f}(x)|$

  We will focus mainly on the squared error loss.

- **Binary data classification**: For binary data models, like logistic regression, a popular loss function is the *misclassification* also known as the $0-1$ loss. Given

response y which are Bernoulli(p) where

$$p = \frac{e^{X\beta}}{1 + e^{X\beta}},$$

we can find the estimated p, $\hat{p}$ by setting

$$\hat{p} = \frac{e^{X\hat{\beta}_{\text{MLE}}}}{1 + e^{X\hat{\beta}_{\text{MLE}}}},$$

where $\hat{\beta}_{\text{MLE}}$ are the MLE estimates obtained using an optimization algorithm. These $\hat{p}$ are the estimated probability of success. Set $\hat{f}(x) = 1 \cdot \mathbb{I}\{\hat{p} \geq .5\} + 0 \cdot \mathbb{I}\{\hat{p} < .5\}$. (The cutoff, .5, is the default cutoff, but can be changed depending on the problem). Then the *misclassification* loss function checks whether the model has misclassified the observation

$$\text{Misclassification or } 0-1 \text{ loss} : L(y, \hat{f}(x)) = \mathbb{I}(y \neq \hat{f}(x)).$$

- **Gaussian mixture model:** Where there are no "response" and "covariates", like the Gaussian mixture model (GMM) example, it is difficult to implement the above two loss functions. But recall, that in the GMM example, we used the log-likelihood to see whether our estimates were good. Thus, we can use negative log-likelihood to see how bad our estimates are. Thus, here the loss function is:

$$L(x, \hat{\theta}) = -\log f(x|\hat{\theta}).$$

  We will discuss the specific GMM case in more detail later.

How do we use these loss functions?

Given a dataset, we are interested in estimating the *test error* which is the expected loss, of an independent dataset given estimates from the current dataset. If our given dataset is $\mathcal{D}$

$$\text{Err}_{\mathcal{D}} = \text{E}\left[L(y, \hat{f}(x)) \mid \mathcal{D}\right],$$

where $\hat{f}(x)$ denotes the estimated y for a new independent dataset, given estimators from the current dataset. For example, using $\hat{\beta}$ from a given dataset $\mathcal{D}$, then $\hat{f}(x) = X_{\text{new}}\hat{\beta}$.

Test error is built for a given dataset $\mathcal{D}$. We are interested in the *expected prediction*

error which is the expected test error over all such $\mathcal{D}$:

$$\text{Err} = \text{E}\left[L(y, \hat{f}(x))\right] = \text{E}\left[\text{E}\left[L(y, \hat{f}(x)) \mid \mathcal{D}\right]\right] = \text{E}\left[\text{Err}_{\mathcal{D}}\right]$$

The expected prediction error is the expected error in predicting y for new datasets given models built from any dataset.

So, given different values of a tuning parameter, our goal is to compare the prediction error, and choose the tuning parameter which yields the lowest test error. But, the test error requires obtaining the loss on *independent datasets*. We just have one dataset available to use.

This is where cross-validation is critically useful.

1.2 Cross-validation

In many data analysis techniques, there are tuning/nuisance parameters that need to be chosen in order to fit the model. How can we choose these nuisance parameters? For example:

- λ, the tuning parameter in penalized regression
- α, the bridge regression penalization factor
- C, the number of classes in mixture of Gaussians, EM algorithm.

Or, we can fit multiple different models to the same dataset. For example, we could fit linear regression, bridge regression, or ridge regression. Which fit is the best? How can we make that assesment?

Cross-validation provides a way to estimate the *test error* without requiring new data. In cross-validation, our original dataset is broken into chunks in order to emulate independent datasets. Then the model is fit on one chunk and tested on another, with the loss recorded. The way the data is broken into chunks can lead to different methods of cross-validation.

1.2.1 Holdout method

One way is to use the holdout method. In this,

- Choose a loss function

- We split the data into two parts: a *training set* and a *test set*.
- Fit the model on the training set, and obtain estimates of the observation y for the test set. Compute loss on the test set.
- Repeat the process for different models and choose the model with smallest error.

This method has two drawbacks:

1. We do not often have the luxury of "throwing away" observations for the test data, specially in small data problems.
2. It is possible, that just by chance a bad split in the data makes the test/train split data drastically different.

1.2.2 Leave-one-out Cross-validation

In leave-one-out cross-validation (LOOCV), the data $\mathcal{D}$ of size n is split into a *training set* of size $n-1$ and a *test set* of size 1. This is repeated for systematically all observations so that there are n such splits possible.

For each split, the test error is estimated, and the average error over all splits is calculated, which estimates the expected prediction error for a model fit $\hat{f}(x)$. Let $\hat{f}^{-i}(x_i)$ denote the predicted value of y_i using the model that removes the ith data point. Then CV estimate of the prediction error is

$$\mathrm{CV}_1(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}^{-i}(x_i)) \qquad \approx \mathrm{E}\left[L(y, \hat{f}(x))\right] .$$

Note that each $\hat{f}^{-i}(x_i)$ represents model fits using different datasets, with testing on one observation.

$\mathrm{CV}_1(\hat{f})$ can be calculated for different models or tuning parameters. Let θ denote a generic tuning parameter utilized in determining the model fit $\hat{f}$, and let $\hat{f}^{-i}(x_i, \theta)$

denote the predicted value for y_i using a training data that doesn't include the ith observation and uses θ as a tuning parameter. Then:

$$\mathrm{CV}_1(\hat{f}, \theta) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{f}^{-i}(x_i, \theta)) \,.$$

The chosen model is the one with θ such that

$$\theta_{\text{chosen}} = \arg\min_{\theta} \left\{ \mathrm{CV}_1(\hat{f}, \theta) \right\}$$

The final model is $\hat{f}(X, \theta_{\text{chosen}})$ fit to *all* the data. In this way we can accomplish two things: obtain an estimate of the prediction error and choose a model.

Points:

- $\mathrm{CV}_1(\hat{f})$ is an approximately unbiased estimator of the test error. This is because the expectation is being evaluated over n samples of the data that are very close to the original data $\mathcal{D}$.
- LOOCV is computationally burdensome since the model is fit n times for each θ. If the number of possible values of θ is large, this becomes computationally expensive.

1.2.3 K-fold cross-validation

The data is randomly split into K roughly equal-sized parts. For any kth split, the rest of the $K-1$ parts make up the *training set* and the model is fit to the *training set*. We then estimate the prediction error for each element in the kth part. Repeating this for all $k = 1, 2, \ldots, K$ parts, we have an estimate of the prediction error.

Let $\kappa : \{1, \ldots, N\} \mapsto \{1, \ldots, K\}$ indicates the partition to which each ith observation belongs. Let $\hat{f}^{-\kappa(i)}(x)$ be the fitted function for the $\kappa(i)th$ partition removed. Then,

the estimated prediction error is

$$\mathrm{CV}_K(\hat{f}, \theta) = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{n/K} \sum_{i \in k^{\text{th}}\text{split}} L(y_i, \hat{f}^{-\kappa(i)}(x_i, \theta)) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{f}^{-\kappa(i)}(x_i, \theta)) \,.$$

The chosen model is the one with θ such that

$$\theta_{\text{chosen}} = \arg\min_{\theta} \left\{ \mathrm{CV}_K(\hat{f}, \theta) \right\}$$

The final model is $\hat{f}(X, \theta_{\text{chosen}})$ fit to *all* the data.

Points:

- For small K, the bias in estimating the true prediction error is large since each training data is quite different from the given dataset $\mathcal{D}$.
- The computational burden is lesser when K is small.

Usually, for large datasets, 10-fold or 5-fold CV is common. For small datasets, LOOCV is more common.

1.3 Questions to think about

- Which algorithms do you think would be time-consuming to do LOOCV for?

MTH 511a - 2021: L28: Cross-validation continued

Instructor: Dootika Vats

1 Cross-validation

1.1 Comparing different cross-validations

Before going into examples of using cross-validation, we will demonstrate the estimation quality of various cross-validation techniques: Leave-one-out cross-validation (LOOCV), 10-fold CV, and 5-fold CV.

We will do this by simulating a linear regression model

$$Y = X\beta + \epsilon$$

where Y, X, β and ϵ are as usual. In order to compare the cross-validation prediction error to the true prediction error, we repeat the following many times to estimate the true prediction error.

1. We simulate data (X, Y) according to the above model
2. We estimate β using MLE and obtain $\hat{\beta_{\text{MLE}}}$
3. We generate a new dataset (X_{new}, Y_{new}) and obtain the predicted value of Y_{new} using $\hat{\beta}_{\text{MLE}}$. That is
$$\hat{Y}_{new}^{pred} = X_{new}\hat{\beta}_{\text{MLE}}$$

4. We measure loss $L(Y_{new}, \hat{Y}_{new}^{pred})$. We will use squared error loss.

The above process gives us an idea of the true prediction error. We will then compare it to the prediction error using various cross-validation techniques.

```
##################################################
## Implementing CV methods for a simulated dataset
## We will generate X and y and generating new independent
## X.new and y.new, we will calculate the "true" prediction error
## Then we implement the LOOCV, and K-fold CV
##################################################
set.seed(10)

n <- 100
p <- 50
sigma2.star <- 4
beta.star <- rnorm(p, mean = 2)
beta.star # to output

# repeat 500 times
B <- 5e2
foo <- 0



# we will use the following code to make our K-fold splits
permutation <- sample(1:n, replace = FALSE) # random permutation of 1:n
K <- 5
# Making a list of indices for each split
(test.index <- split(permutation, rep(1:K, length = n, each = n/K)))

#$`1`
# [1] 31 87 1 59 25 39 16 98 99 65 12 52 58 95 94 24 22 13 9 89
#
#$`2`
# [1] 100 61  85  97  91  47  92  70  75  88  68   4  29  21  30  56
#[17]   7  26  28  57
#
#$`3`
```

```
# [1] 62 38 34 53 27 2 73 23 74 55 33 81 10 44 76 60 90 17 20 80
#
#$`4`
# [1] 18 36 48 64 86 46 45 66 11 93 49 84 40 5 19 96 41 77 63 50
#
#$`5`
# [1] 42 3 54 43 35 14 51 37 67 72 79 83 8 78 6 82 15 32 71 69


CV.error <- matrix(0, nrow = B, ncol = 4)
time <- matrix(0, nrow = B, ncol = 3)
colnames(CV.error) <- c( "Truth","LOOCV", "10-fold", "5-fold")
```

In the above code, we have obtained a true β^* with $p = 50$ and we have set $n = 100$. I also provide the code I will use to make CV splits.

```
for(b in 1:B)
{
    # code takes a while, hence printing for feedback
    if(b %% 100 == 0) print(b)

    # Generate new d
    # Making design matrix, first column is 1
    X <- cbind(1, matrix(rnorm(n*(p-1)), nrow = n, ncol = (p-1)))

    # Generating response
    y <-  X %*% beta.star + rnorm(n, mean = 0, sd = sqrt(sigma2.star))

    beta.mle <- solve( t(X) %*% X) %*% t(X) %*% y

    # independent new X and y
    X.new <- cbind(1, matrix(rnorm(n*(p-1)), nrow = n, ncol = (p-1)))
    y.new <- X.new %*% beta.star + rnorm(n, mean = 0, sd = sqrt(sigma2.star))

    CV.error[b, 1] <- mean( (y.new - X.new %*% beta.mle)^2 )

  ## The above is only to obtain the true prediction error
```

```r
## Now the following code obtains the CV estimates.
## the code below does not utilize the new data
## since new data is not available when we do CV.


  ################################################
  # Leave-one-out Cross-validation

  time0 <- proc.time()[3]
  foo2 <- 0
  for(i in 1:n)
  {
    # Making training data
    X.train <- X[-i,] # removing ith X
    y.train <- y[-i] #removing ith y

    # fitting model for training data
    beta.train <- solve(t(X.train) %*% X.train) %*% t(X.train) %*% y.train

    # test error
    foo2 <- foo2 + (y[i] - X[i,] %*% beta.train)^2
  }
  CV.error[b, 2] <- foo2/n
  time[b,1] <- proc.time()[3] - time0


  ################################################
  # 10-fold Cross-validation
  permutation <- sample(1:n, replace = FALSE)
  K <- 10

  # Making a list of indices for each split
  test.index <- split(permutation, rep(1:K, length = n, each = n/K))

  time0 <- proc.time()[3]
  foo3 <- 0
  for(k in 1:K)
```

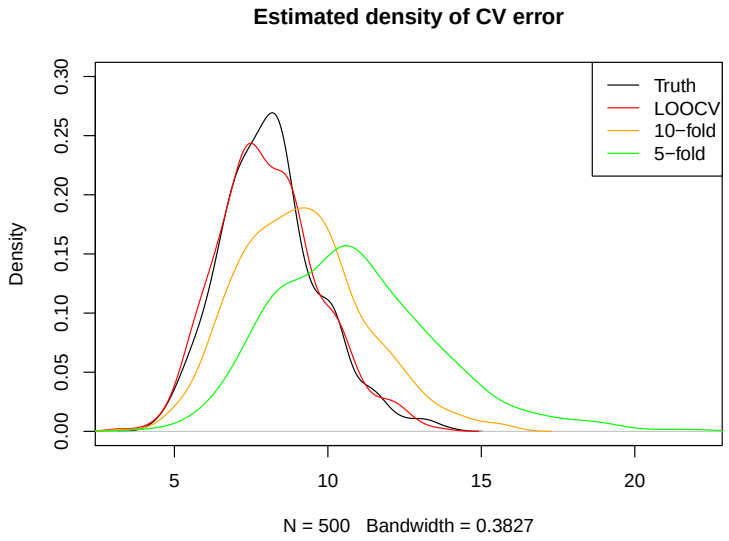
```
  {
    X.train <- X[-test.index[[k]], ]
    y.train <- y[-test.index[[k]]]
    X.test <- X[test.index[[k]], ]
    y.test <- y[test.index[[k]]]

    beta.train <- solve(t(X.train) %*% X.train) %*% t(X.train) %*% y.train

    foo3 <- foo3 + sum( (y.test - X.test %*% beta.train)^2)
  }
  CV.error[b, 3] <- foo3/n
  time[b,2] <- proc.time()[3] - time0

  ################################################
  # 5-fold Cross-validation

  # Making a permutation from 1:n
  permutation <- sample(1:n, replace = FALSE)
  K <- 5
  test.index <- split(permutation, rep(1:K, length = n, each = n/K))

  time0 <- proc.time()[3]
  foo4 <- 0
  for(k in 1:K)
  {
    X.train <- X[-test.index[[k]], ]
    y.train <- y[-test.index[[k]]]
    X.test <- X[test.index[[k]], ]
    y.test <- y[test.index[[k]]]

    beta.train <- solve(t(X.train) %*% X.train) %*% t(X.train) %*% y.train

    foo4 <- foo4 + sum( (y.test - X.test %*% beta.train)^2)
  }
  CV.error[b, 4] <- foo4/n
  time[b,3] <- proc.time()[3] - time0
################################################
}
```

```
# LOOCV is most accurate
colMeans(CV.error)
#    Truth    LOOCV  10-fold   5-fold
# 8.146910 8.101374 9.124679 10.882481

# LOOCV is expensive
colMeans(time)
# [1] 0.069552 0.006628 0.002676
```

We see that LOOCV yields estimates closest to the truth, with 5-fold being the farthest from the truth. However, 5-fold is the fastest to implement and LOOCV is significantly slower.

Since we have $B = 500$ estimates of the prediction error, we can compare their overall distribution as well:

```
# LOOCV is the closest
plot(density(CV.error[,1]), xlim = range(CV.error), ylim = c(0, .30), main
    = "Estimated density of CV error")
lines(density(CV.error[,2]), col = "red")
lines(density(CV.error[,3]), col = "orange")
lines(density(CV.error[,4]), col = "green")
legend("topright", legend = colnames(CV.error), col = c("black", "red",
    "orange", "green"), lty = 1)
```



Thus, through this simulation, we see that LOOCV is the closest to the truth, however, if the dataset is large, it can be too time consuming to implement it. On the other hand, 5-fold CV is not that accurate. If computation allows it, 10-fold CV is a good sacrifice between computation time and accuracy.

We will present examples of implementation of cross-validation for regression and Gaussian mixture model. Specifically, in the next lecture, we will now use cross-validation to choose tuning parameters.

1.2 Applying Cross-validation

We will cover some examples of using cross-validation to choose tuning parameters.

1.2.1 Regression model selection

Ridge regression:

Recall the regression model:

$$Y = X\beta + \epsilon\,,$$

where $\epsilon \sim N_n(0, \sigma^2\, I_n)$. In this example subsection, we will focus our attention only on ridge regression estimators. Similar cross-validations can be done for bridge regression.

Recall the ridge objective function for a given λ is

$$Q_\lambda(\beta) = \frac{(y - X\beta)^T(y - X\beta)}{2} + \frac{\lambda}{2}\beta^T\beta\,.$$

Recall, the ridge estimator of β is

$$\hat{\beta}_\lambda = (X^TX + \lambda I_n)^{-1}X^Ty\,.$$

Here, the solution for β depends on the value of λ. We want to choose λ so that prediction error is minimized. We choose the squared error loss function.

To choose the best model in this case, we set a vector of $\lambda : \lambda_1, \ldots, \lambda_m$.

Since closed-form solution for the ridge-regression estimator is available, cross-validation steps will be fairly fast. Thus we choose LOOCV.

For each λ_i, we will implement LOOCV and estimate the prediction error. Whichever λ minimzies the prediction error, will be the chosen λ.

Consider the `mtcars` dataset in `R`. The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (197374 models). The response is the miles per gallon of the 32 cars. Run a `?mtcars` in `R` to learn more about the dataset.

```
#####################################################
## Choosing lambda in ridge regression
## using LOOCV
## dataset is mtcars
#####################################################
data(mtcars)
y <- mtcars$mpg
X <- cbind(1, as.matrix(mtcars[ ,-1]))

n <- dim(X)[1]
p <- dim(X)[2]
lam.vec <- c(10^(seq(-8, 8, by = .1)))
CV.error <- numeric(length = length(lam.vec))

for(l in 1:length(lam.vec))
{
```

```
  foo2 <- 0
  lam <- lam.vec[l]
  for(i in 1:n)
  {
    # Making training data
    X.train <- X[-i,] # removing ith X
    y.train <- y[-i] #removing ith y

    # fitting model for training data
    beta.train <- solve(t(X.train) %*% X.train + lam*diag(p)) %*% t(X.train)
        %*% y.train

    # test error
    foo2 <- foo2 + (y[i] - X[i,] %*% beta.train)^2
  }
  CV.error[l] <- foo2/n
}

# lambda that yields minimum error is chosen
(chosen.lam <- lam.vec[which.min(CV.error)] )
# [1] 5.011872


# Remember, refit the model with the final value of lambda
beta.final <- solve(t(X) %*% X + chosen.lam*diag(p)) %*% t(X) %*% y
beta.final
#             [,1]
#      0.247777096
# cyl  0.346710987
# disp -0.007378144
# hp   -0.008715992
# drat 1.420911900
# wt   -1.653648857
# qsec 0.938547096
# vs   -0.077511677
# am    1.443975953
# gear 1.530167865
# carb -0.745846809
```

Bridge regression:

For the same dataset, we now implement bridge regression with a grid of values for α and λ. We will use 4-fold cross validation. (We use 4-fold instead of 5-fold since 32 is easily divisible by 4).

```
##################################################
## Choosing lambda and alpha in bridge regression
## using 4-fold dataset is mtcars
##################################################
set.seed(12)
data(mtcars)
y <- mtcars$mpg
X <- cbind(1, as.matrix(mtcars[ ,-1]))

n <- dim(X)[1]
p <- dim(X)[2]

## MM algorithm for bridge regression
## for any alpha and lambda
bridge <- function(y, X, lambda, alpha, tol = 1e-8)
{
  current <- solve( t(X) %*%X + diag(lambda,p) ) %*% t(X) %*%y # start at
     ridge solution
  iter <- 0
  diff <- 100
  while( (diff > tol) && iter < 1000)
  {
    iter <- iter + 1
    # M matrix diagonals
    ms <- as.vector( lambda/2 *( abs(current))^(alpha - 2) )

    # MM update -- using qr.solve for numerical stability
    update <- qr.solve(t(X) %*% X + diag(ms, p)) %*% t(X) %*% y

    diff <- sum( (current - update)^2 )
    current <- update
  }
  return(current)
```

```
}



lam.vec <- 1:10
alpha <- seq(1,2, by =.05)

CV.error <- matrix(0, nrow = length(lam.vec), ncol = length(alpha))
permutation <- sample(1:n, replace = FALSE)
K <- 4

# Making a list of indices for each split
test.index <- split(permutation, rep(1:K, length = n, each = n/K))

for(l in 1:length(lam.vec))
{
  lam <- lam.vec[l]
  for(a in 1:length(alpha))
  {
    foo3 <- 0
    for(k in 1:K)
    {
      X.train <- X[-test.index[[k]], ]
      y.train <- y[-test.index[[k]]]
      X.test <- X[test.index[[k]], ]
      y.test <- y[test.index[[k]]]

      beta.train <- bridge(y.train, X.train, lambda = lam, alpha = alpha[a])

      foo3 <- foo3 + sum( (y.test - X.test %*% beta.train)^2)
    }
    CV.error[l,a] <- foo3/n
  }
}

# lambda that yields minimum error is chosen
ind <- which(CV.error == min(CV.error), arr.ind = TRUE)
(chosen.alpha <- alpha[ind[2]])
(chosen.lam <- lam.vec[ind[1]] )
```

```
# Final estimates
bridge(y, X, lambda = chosen.lam, alpha = chosen.alpha)
#              [,1]
#      0.0040899424
# cyl  0.2506111993
# disp 0.0038803710
# hp  -0.0145184786
# drat 1.3829350298
# wt  -2.7347035965
# qsec 1.0656202488
# vs  -0.0001061083
# am   1.8796458185
# gear 1.3880810512
# carb -0.4745591302
```

1.2.2 Choosing C in GMM

We have discussed before that for choosing C, the usual loss functions don't work since there is not response and covariate. However, we know that models that yield the highest log-likelihood or the lowest negative log-likelihood are preferred. Thus, the loss function is:

$$L(x, \hat{\theta}) = -\log f(x|\hat{\theta})\,.$$

We observe $X_1, \ldots, X_n$ from a mixture of normals and split the data up into testing and training CV sets. We fit the EM algorithm to estimate the MLE from $c = 2, \ldots, C$ classes over the training sets and estimate the prediction error.

Note that if use the negative log-likelihood on the full dataset to choose the number of classes C, we can keep reducing the loss by continually increasing C, until each data point is a cluster in itself. Thus the CV algorithm is critical here.

Below are functions that calculate the negative log-likelihood and that implement the EM algorithm for multivariate GMM.

```
##########################################
## EM algorithm to fit mixture of Gaussians
## to multivariate data (old faithful)
##########################################
```

```r
set.seed(12)


# calculate negative log-likelihood
# of mixture of multivariate normal
log_like <- function(X, pi.list, mu.list, Sigma.list, C)
{
  foo <- 0
  for(c in 1:C)
  {
    foo <- foo + pi.list[[c]]*dmvnorm(X, mean = mu.list[[c]], sigma =
        Sigma.list[[c]])
  }
  return(-sum(log(foo)))
}

# X is the data
# C is the number of clusters
GLMMforC <- function(X, C, tol = 1e-3, maxit = 1e3)
{
  n <- dim(X)[1]
  p <- dim(X)[2]
  ######## Starting values ###################
  ## pi are equally split over C
  pi.list <- rep(1/C, C)

  mu <- list()
  Sigma <- list()

  # The means for each C cannot be the same,
  # since then the three distributions overlap
  # Hence adding random noise to colMeans(X)
  for(i in 1:C)
  {
    mu[[i]] <- rnorm(p, sd = 3) + colMeans(X)
    Sigma[[i]] <- var(X)
  }
  # Choosing good starting values is important since
```

```r
# The GMM likelihood is not concave, so the algorithm
# may converge to a local optima.

######## EM algorithm steps ###################

iter <- 0
diff <- 100
old.mu <- mu
old.Sigma <- Sigma
old.pi <- pi.list

Ep <- matrix(0, nrow = n, ncol = C) # gamma_{i,c}
save.loglike <- 0
while((diff > tol) && (iter < maxit) )
{
  iter <- iter + 1
  ## E step: find gammas
  for(c in 1:C)
  {
    Ep[ ,c] <- pi.list[c]*apply(X, 1, dmvnorm , mu[[c]], Sigma[[c]])
  }
  Ep <- Ep/rowSums(Ep)

  ### M-step
  pi.list <- colMeans(Ep)
  for(c in 1:C)
  {
    mu[[c]] <- colSums(Ep[ ,c] * X )/sum(Ep[,c])
  }

  for(c in 1:C)
  {
    foo <- 0
    for(i in 1:n)
    {
      foo <- foo + (X[i, ] - mu[[c]]) %*% t(X[i, ] - mu[[c]]) * Ep[i,c]
    }
    Sigma[[c]] <- foo/sum(Ep[,c])
```

```
    # Below is to ensure the estimator is positive definite
    # otherwise next iteration gamma_i,c,k cannot be calculated
    Sigma[[c]] <- Sigma[[c]] + diag(1e-5, p)
  }

  save.loglike <- c(save.loglike, log_like(X = X, pi.list = pi.list,
     mu.list = mu, Sigma.list = Sigma, C = C))
  # Difference in the log-likelihoods as the difference criterion
  diff <- abs(save.loglike[iter+1] - save.loglike[iter])

  old.mu <- mu
  old.Sigma <- Sigma
  old.pi <- pi.list
 }

 # Final allocation updates
 for(c in 1:C)
 {
   Ep[ ,c] <- pi.list[c]*apply(X, 1, dmvnorm , mu[[c]], Sigma[[c]])
 }
 Ep <- Ep/rowSums(Ep)

 return(list("pi" = pi.list, "mu" = mu, "Sigma" = Sigma, "Ep" = Ep,
     "log.like" = tail(save.loglike,1)))
}
```

Next, we will do CV to choose the number of clusters in the `faithful` dataset. We choose between $2, 3, 4$ classes.

NOTE: Because EM for GLMM does not converge to a global maxima, for each cross-validation set, we run the algorithm multiple times from different starting values, and choose the run that produced the lowest negative log-likelihood.

```
#### Functions needed for cross validation
# My data set
data(faithful)
X <- as.matrix(faithful)
n <- dim(X)[1]
```

```
################################################
# 5-fold Cross-validation

permutation <- sample(1:n, replace = FALSE)
K <- 5
# Uneven folds, but that is ok
test.index <- split(permutation, rep(1:K, length = n, each = n/K))

# Testing whether 2-4 classes are needed
potC <- 2:4
CV.errorLike <- numeric(length = length(potC))

# will run EM multiple times for each training data
# since EM convergence to local minima. Setting these
# reps = 7
reps <- 5
model.save <- list()
for(c in 1:length(potC))
{
  foo3 <- 0
  for(k in 1:K)
  {
    print(c(c,k))
    X.train <- X[-test.index[[k]], ]
    X.test <- X[test.index[[k]], ]

    for(r in 1:reps)
    {
      model.save[[r]] <- GLMMforC(X = X.train, C = potC[c])
    }
    # which ever run is the lowest negative log-like
    chosen.run <- which.min(sapply(model.save, function(t) t$log.like))
    model <- model.save[[chosen.run]]
    foo3 <- foo3 + log_like(X = X.test, pi.list = model$pi, mu.list
        =model$mu, Sigma.list = model$Sigma, C = potC[c])
  }
  CV.errorLike[c] <- foo3/n
```

```
}

CV.errorLike #Lowest value is for C = 3,
# [1] 4.217456 4.215619 4.232590

# Thus choose C = 3 classes.
```

When you run the above code you see there are multiple issues with using CV for model selection in this example

- **Time:** The cross-validation is time consuming particularly because we have to run the mode for multiple starting values
- **Final model:** The method does not give you the final estimates, which means you need to run the model again for the chosen C. However, it is possible when you run it again that you converge to a different solution!

1.2.3 AIC and BIC

An alternative model selection procedure is used, called the *Akaike Information Criterion*. The AIC is a function of the *negative* log-likelihood and a penalty term, penalizing the number of parameter the algorithm has to estimate. That is,

$$\text{AIC}(\hat{\theta} \mid x) = -2 \log l(\hat{\theta}|x) + 2K$$

where K is the number of parameters being estimated. $\text{AIC}(\hat{\theta} \mid x)$ is calculated by fitting the model on the *full dataset*. Since the negative log-likelihood will increase as you increase the number of clusters, the penalty term of $2K$ penalizes the usage of too many clusters.

In our 2-dimensional GMM example $K = 2 * C + (C - 1) + 3C = 6C - 1$.

Another similar information criterion is the *Bayesian Information Criterion* (BIC) defined as

$$\text{BIC}(\hat{\theta} \mid x) = -2 \log l(\hat{\theta}|x) + \log(n)K$$

where K is again the number of parameters being estimated. The BIC criterion increases the penalty term when the number of data are larger. This makes sense since when large number of data are available, we should be able to find simpler models. It

is well known that the BIC criterion is theoretically superior to the AIC criterion and even in this examples works better.

Note: Since we have the negative-log-likelihood, we want to *minimize* the AIC and BIC values.

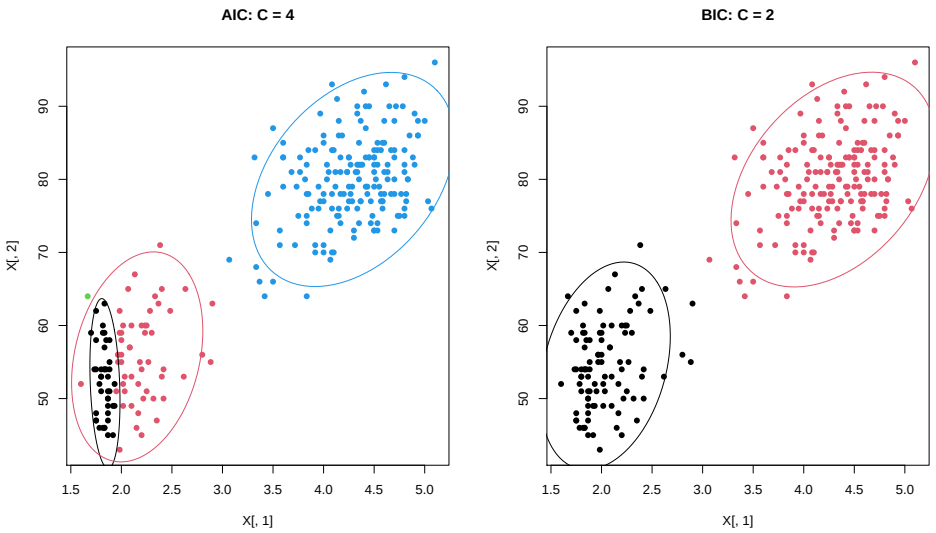
```
#######################################
##### Model selection via AIC #######
#######################################
aic <- function(X, pi.list, mu.list, Sigma.list, C)
{
  nlike <- log_like(X, pi.list, mu.list, Sigma.list, C)
  rtn <- 2*nlike + 2* (6*C - 1) # No. of params = 3*C - 1
  return(rtn)
}

bic <- function(X, pi.list, mu.list, Sigma.list, C)
{
  n <- dim(X)[1]
  nlike <- log_like(X, pi.list, mu.list, Sigma.list, C)
  rtn <- 2*nlike + log(n)* (6*C - 1) # No. of params = 3*C - 1
  return(rtn)
}

aicLike <- numeric(length = length(potC))
bicLike <- numeric(length = length(potC))
reps <- 5
model.save <- list()
model <- list()
for(c in 1:length(potC))
{
  print(c)
  for(r in 1:reps)
  {
    model.save[[r]] <- GLMMforC(X = X, C = potC[c])
  }

  chosen.run <- which.min(sapply(model.save, function(t) t$log.like))
  model[[c]] <- model.save[[chosen.run]]
```

```
  aicLike[c] <- aic(X = X, pi.list = model[[c]]$pi, mu.list =model[[c]]$mu,
      Sigma.list = model[[c]]$Sigma, C = potC[c])
  bicLike[c] <- bic(X = X, pi.list = model[[c]]$pi, mu.list =model[[c]]$mu,
      Sigma.list = model[[c]]$Sigma, C = potC[c])
}

aicLike # Lowest is C = 4 is the best!
# [1] 2282.528 2272.431 2255.077

bicLike # Lowest is C = 2
# [1] 2322.192 2333.730 2338.011
```

$C = 4$ gives the lowest AIC and $C = 2$ gives the lowest BIC. Note that since we've saved the model, we can now just use that model without rerunning anything. This is a very useful feature of model selection via AIC/BIC.

```
par(mfrow = c(1,2))
chosen <- which.min(aicLike)
allot <- apply(model[[chosen]]$Ep, 1, which.max) ## Final allotment of
    classification
plot(X[,1], X[,2], col = allot, pch = 16, main = "AIC: C = 4") # plot
    allotment

ell <- list()
for(c in 1:potC[[chosen]])
{
  ell[[c]] <- ellipse(model[[chosen]]$Sigma[[c]], centre =
      as.numeric(model[[chosen]]$mu[[c]]))
  lines(ell[[c]], col = c)
}

chosen <- which.min(bicLike)
allot <- apply(model[[chosen]]$Ep, 1, which.max) ## Final allotment of
    classification
plot(X[,1], X[,2], col = allot, pch = 16, main = "BIC: C = 2") # plot
    allotment

ell <- list()
```

```
for(c in 1:potC[[chosen]])
{
  ell[[c]] <- ellipse(model[[chosen]]$Sigma[[c]], centre =
      as.numeric(model[[chosen]]$mu[[c]]))
  lines(ell[[c]], col = c)
}
```



Using AIC, the model chooses $C = 4$, which is clearly not a good model since it has only one point in the fourth cluster. Clearly, the penalty added in AIC is not enough to ensure against over-fitting. Using BIC imposes extra penalty that helps against overfitting.

MTH 511a - 2021: L29 - Cross-validation for GMM

Instructor: Dootika Vats

1 Resampling Methods

1.1 Cross-validation

1.1.1 Choosing C in GMM

We have discussed before that for choosing C, the usual loss functions don't work since there is not response and covariate. However, we know that models that yield the highest log-likelihood or the lowest negative log-likelihood are preferred. Thus, the loss function is:

$$L(x, \hat{\theta}) = -\log f(x|\hat{\theta})\,.$$

We observe $X_1, \ldots, X_n$ from a mixture of normals and split the data up into testing and training CV sets. We fit the EM algorithm to estimate the MLE from $c = 2, \ldots, C$ classes over the training sets and estimate the prediction error.

Note that if use the negative log-likelihood on the full dataset to choose the number of classes C, we can keep reducing the loss by continually increasing C, until each data

point is a cluster in itself. Thus the CV algorithm is critical here.

Below are functions that calculate the negative log-likelihood and that implement the EM algorithm for multivariate GMM.

```
##########################################
## EM algorithm to fit mixture of Gaussians
## to multivariate data (old faithful)
##########################################
set.seed(12)



# calculate negative log-likelihood
# of mixture of multivariate normal
log_like <- function(X, pi.list, mu.list, Sigma.list, C)
{
  foo <- 0
  for(c in 1:C)
  {
    foo <- foo + pi.list[[c]]*dmvnorm(X, mean = mu.list[[c]], sigma =
        Sigma.list[[c]])
  }
  return(-sum(log(foo)))
}


# X is the data
# C is the number of clusters
GLMMforC <- function(X, C, tol = 1e-3, maxit = 1e3)
{
  n <- dim(X)[1]
```

```r
p <- dim(X)[2]
######## Starting values ###################
## pi are equally split over C
pi.list <- rep(1/C, C)

mu <- list()
Sigma <- list()

# The means for each C cannot be the same,
# since then the three distributions overlap
# Hence adding random noise to colMeans(X)
for(i in 1:C)
{
  mu[[i]] <- rnorm(p, sd = 3) + colMeans(X)
  Sigma[[i]] <- var(X)
}
# Choosing good starting values is important since
# The GMM likelihood is not concave, so the algorithm
# may converge to a local optima.

######## EM algorithm steps ###################

iter <- 0
diff <- 100
old.mu <- mu
old.Sigma <- Sigma
old.pi <- pi.list

Ep <- matrix(0, nrow = n, ncol = C) # gamma_{i,c}
```

```
save.loglike <- 0
while((diff > tol) && (iter < maxit) )
{
  iter <- iter + 1
  ## E step: find gammas
  for(c in 1:C)
  {
    Ep[ ,c] <- pi.list[c]*apply(X, 1, dmvnorm , mu[[c]], Sigma[[c]])
  }
  Ep <- Ep/rowSums(Ep)

  ### M-step
  pi.list <- colMeans(Ep)
  for(c in 1:C)
  {
    mu[[c]] <- colSums(Ep[ ,c] * X )/sum(Ep[,c])
  }

  for(c in 1:C)
  {
    foo <- 0
    for(i in 1:n)
    {
      foo <- foo + (X[i, ] - mu[[c]]) %*% t(X[i, ] - mu[[c]]) * Ep[i,c]
    }
    Sigma[[c]] <- foo/sum(Ep[,c])

    # Below is to ensure the estimator is positive definite
    # otherwise next iteration gamma_i,c,k cannot be calculated
```

```
    Sigma[[c]] <- Sigma[[c]] + diag(1e-5, p)
  }

  save.loglike <- c(save.loglike, log_like(X = X, pi.list = pi.list,
      mu.list = mu, Sigma.list = Sigma, C = C))
  # Difference in the log-likelihoods as the difference criterion
  diff <- abs(save.loglike[iter+1] - save.loglike[iter])

  old.mu <- mu
  old.Sigma <- Sigma
  old.pi <- pi.list
}

# Final allocation updates
for(c in 1:C)
{
  Ep[ ,c] <- pi.list[c]*apply(X, 1, dmvnorm , mu[[c]], Sigma[[c]])
}
Ep <- Ep/rowSums(Ep)

return(list("pi" = pi.list, "mu" = mu, "Sigma" = Sigma, "Ep" = Ep,
    "log.like" = tail(save.loglike,1)))
}
```

Next, we will do CV to choose the number of clusters in the `faithful` dataset. We choose between $2, 3, 4$ classes.

NOTE: Because EM for GLMM does not converge to a global maxima, for each cross-validation set, we run the algorithm multiple times from different starting values, and

choose the run that produced the lowest negative log-likelihood.

```
 #### Functions needed for cross validation
# My data set
data(faithful)
X <- as.matrix(faithful)
n <- dim(X)[1]


################################################
# 5-fold Cross-validation


permutation <- sample(1:n, replace = FALSE)
K <- 5
# Uneven folds, but that is ok
test.index <- split(permutation, rep(1:K, length = n, each = n/K))


# Testing whether 2-4 classes are needed
potC <- 2:4
CV.errorLike <- numeric(length = length(potC))


# will run EM multiple times for each training data
# since EM convergence to local minima. Setting these
# reps = 7
reps <- 5
model.save <- list()
for(c in 1:length(potC))
{
  foo3 <- 0
  for(k in 1:K)
  {
```

```
    print(c(c,k))
    X.train <- X[-test.index[[k]], ]
    X.test <- X[test.index[[k]], ]

    for(r in 1:reps)
    {
      model.save[[r]] <- GLMMforC(X = X.train, C = potC[c])
    }
    # which ever run is the lowest negative log-like
    chosen.run <- which.min(sapply(model.save, function(t) t$log.like))
    model <- model.save[[chosen.run]]
    foo3 <- foo3 + log_like(X = X.test, pi.list = model$pi, mu.list
        =model$mu, Sigma.list = model$Sigma, C = potC[c])
  }
  CV.errorLike[c] <- foo3/n
}

CV.errorLike #Lowest value is for C = 3,
# [1] 4.217456 4.215619 4.232590

# Thus choose C = 3 classes.
```

When you run the above code you see there are multiple issues with using CV for model selection in this example

- **Time:** The cross-validation is time consuming particularly because we have to run the mode for multiple starting values
- **Final model:** The method does not give you the final estimates, which means you need to run the model again for the chosen C. However, it is possible when

you run it again that you converge to a different solution!

1.1.2 AIC and BIC

An alternative model selection procedure is used, called the *Akaike Information Criterion*. The AIC is a function of the *negative* log-likelihood and a penalty term, penalizing the number of parameter the algorithm has to estimate. That is,

$$\text{AIC}(\hat{\theta} \mid x) = -2 \log l(\hat{\theta}|x) + 2K$$

where K is the number of parameters being estimated. $\text{AIC}(\hat{\theta} \mid x)$ is calculated by fitting the model on the *full dataset*. Since the negative log-likelihood will increase as you increase the number of clusters, the penalty term of $2K$ penalizes the usage of too many clusters.

In our 2-dimensional GMM example $K = 2 * C + (C - 1) + 3C = 6C - 1$.

Another similar information criterion is the *Bayesian Information Criterion* (BIC) defined as

$$\text{BIC}(\hat{\theta} \mid x) = -2 \log l(\hat{\theta}|x) + \log(n)K$$

where K is again the number of parameters being estimated. The BIC criterion increases the penalty term when the number of data are larger. This makes sense since when large number of data are available, we should be able to find simpler models. It is well known that the BIC criterion is theoretically superior to the AIC criterion and even in this examples works better.

Note: Since we have the negative-log-likelihood, we want to *minimize* the AIC and BIC values.

```
####################################
```

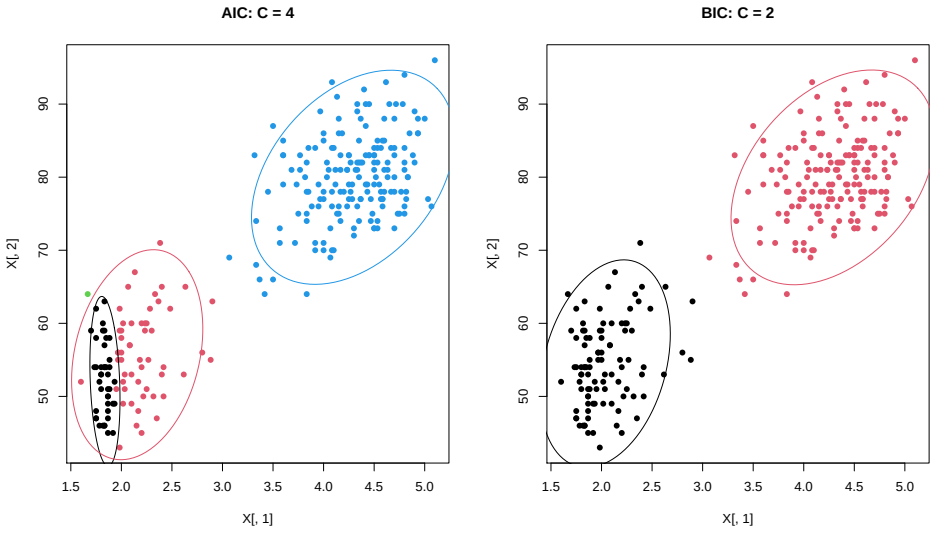
```
##### Model selection via AIC #######
#######################################
aic <- function(X, pi.list, mu.list, Sigma.list, C)
{
  nlike <- log_like(X, pi.list, mu.list, Sigma.list, C)
  rtn <- 2*nlike + 2* (6*C - 1) # No. of params = 3*C - 1
  return(rtn)
}

bic <- function(X, pi.list, mu.list, Sigma.list, C)
{
  n <- dim(X)[1]
  nlike <- log_like(X, pi.list, mu.list, Sigma.list, C)
  rtn <- 2*nlike + log(n)* (6*C - 1) # No. of params = 3*C - 1
  return(rtn)
}

aicLike <- numeric(length = length(potC))
bicLike <- numeric(length = length(potC))
reps <- 5
model.save <- list()
model <- list()
for(c in 1:length(potC))
{
  print(c)
  for(r in 1:reps)
  {
    model.save[[r]] <- GLMMforC(X = X, C = potC[c])
  }
```

```
  chosen.run <- which.min(sapply(model.save, function(t) t$log.like))
  model[[c]] <- model.save[[chosen.run]]
  aicLike[c] <- aic(X = X, pi.list = model[[c]]$pi, mu.list =model[[c]]$mu,
      Sigma.list = model[[c]]$Sigma, C = potC[c])
  bicLike[c] <- bic(X = X, pi.list = model[[c]]$pi, mu.list =model[[c]]$mu,
      Sigma.list = model[[c]]$Sigma, C = potC[c])
}

aicLike # Lowest is C = 4 is the best!
# [1] 2282.528 2272.431 2255.077

bicLike # Lowest is C = 2
# [1] 2322.192 2333.730 2338.011
```

$C = 4$ gives the lowest AIC and $C = 2$ gives the lowest BIC. Note that since we've saved the model, we can now just use that model without rerunning anything. This is a very useful feature of model selection via AIC/BIC.

```
par(mfrow = c(1,2))
chosen <- which.min(aicLike)
allot <- apply(model[[chosen]]$Ep, 1, which.max) ## Final allotment of
    classification
plot(X[,1], X[,2], col = allot, pch = 16, main = "AIC: C = 4") # plot
    allotment

ell <- list()
for(c in 1:potC[[chosen]])
{
```

```
  ell[[c]] <- ellipse(model[[chosen]]$Sigma[[c]], centre =
      as.numeric(model[[chosen]]$mu[[c]]))
  lines(ell[[c]], col = c)
}

chosen <- which.min(bicLike)
allot <- apply(model[[chosen]]$Ep, 1, which.max) ## Final allotment of
    classification
plot(X[,1], X[,2], col = allot, pch = 16, main = "BIC: C = 2") # plot
    allotment

ell <- list()
for(c in 1:potC[[chosen]])
{
  ell[[c]] <- ellipse(model[[chosen]]$Sigma[[c]], centre =
      as.numeric(model[[chosen]]$mu[[c]]))
  lines(ell[[c]], col = c)
}
```



Using AIC, the model chooses $C = 4$, which is clearly not a good model since it has only one point in the fourth cluster. Clearly, the penalty added in AIC is not enough to ensure against over-fitting. Using BIC imposes extra penalty that helps against overfitting.

MTH 511a - 2021: L30 - Bootstrapping

Instructor: Dootika Vats

1 Resampling Methods

1.1 Bootstrapping

We have discussed cross-validation, which we use to choose model tuning parameters. However, once the final model is fit, we would like to make *inference*. That is, we want to account for the variability of the final estimators obtained.

If our estimates are MLEs then we know that under certain important conditions, MLEs have asymptotic normality, that is

$$\sqrt{n}(\hat{\theta}_{\text{MLE}} - \theta) \xrightarrow{d} N(0, \sigma^2_{\text{MLE}}) \,,$$

where σ^2_{MLE} is the inverse Fisher information. Then, if we can estimate σ^2_{MLE}, we can construct asymptotically normal confidence intervals:

$$\hat{\theta}_{\text{MLE}} \pm z_{1-\alpha/2}\sqrt{\frac{\hat{\sigma}^2_{MLE}}{n}} \,.$$

We can also conduct hypothesis tests etc and go on to do regular statistical analysis. But sometimes we cannot use an asymptotic distribution:

1. when our estimates are not MLEs, like ridge and bridge regression
2. when the assumptions for asymptotic normality are not satisfied (I haven't shared these assumptions)
3. when n is not large enough for asymptotic normality to hold

We will try to approximate the distribution of $\hat{\theta}$ using *Boostrap*, and from there we will obtain a confidence intervals.

Suppose $\hat{\theta}$ is some estimator of θ from sample $X_1, \ldots, X_n \overset{iid}{\sim} F$. Then since $\hat{\theta}$ is random it has a sampling distribution G_n that is unknown. If asymptotic normality holds, then $G_n \approx N(\cdot, \cdot)$ for large enough n, but in general we may not know much about G_n. If we could obtain many similar datasets, we could obtain an estimate from each dataset:

$$\hat{\theta}_1, \ldots, \hat{\theta}_B \overset{iid}{\sim} G_n \,.$$

Once we have B realizations from G_n, we can easily estimate characteristics about G_n, like the overall mean, variance, quantiles, etc.

Thus, in order to learn things about the sampling distribution G_n, our goal is to draw more samples of such data. But this, of course is not easy in real-data scenarios. We could obtain more Monte Carlo datasets from F, but we typically do not know the true F.

In bootstrap, instead of obtaining typical Monte Carlo datasets, we will repeatedly "resample" from our current dataset. This would give us an approximate sample from our distribution G_n, and we could estimate characteristics of this distribution! This resampling using information from the current data is called *bootstrapping*. We

will study two popular bootstrap methods: *nonparameteric bootstrap* and *parametric bootstrap*.

1.1.1 Nonparametric Bootstrap

In nonparametric bootstrap, we resample data of size n from within $\{X_1, X_2, \ldots, X_n\}$ (with replacement) and obtain estimates of θ using these samples. That is

$$\begin{aligned}
\text{Bootstrap sample 1:} \qquad & X_{11}^*, X_{21}^*, \ldots, X_{n1}^* \Rightarrow \hat{\theta}_1^* \\
\text{Bootstrap sample 2:} \qquad & X_{12}^*, X_{22}^*, \ldots, X_{n2}^* \Rightarrow \hat{\theta}_2^* \\
& \vdots \\
\text{Bootstrap sample B:} \qquad & X_{1B}^*, X_{2B}^*, \ldots, X_{nB}^* \Rightarrow \hat{\theta}_B^* \,.
\end{aligned}$$

Each sample is called a bootstrap sample, and there are B bootstrap samples. Now, the idea is that $\hat{\theta}_1^*, \ldots \hat{\theta}_B^*$ are B approximate samples from the distribution of $\hat{\theta}, G_n$. That is

$$\hat{\theta}_1^*, \ldots \hat{\theta}_B^* \approx G_n \,.$$

If we want to know the variance of $\hat{\theta}$, then the bootstrap estimate of the variance is

$$\widehat{\mathrm{Var}(\hat{\theta})} = \frac{1}{B-1} \sum_{b=1}^{B} (\hat{\theta}_b^* - B^{-1} \sum_k \hat{\theta}_k^*)^2 \,.$$

Similarly, we may want to construct a $100(1-\alpha)\%$ confidence interval for $\hat{\theta}$. A $100(1-\alpha)\%$ confidence interval is the random interval (L, U) such that

$$\Pr((L, U) \text{ contains } \theta) = 1 - \alpha \qquad .$$

Note that here L and U are random and θ is fixed. We can find the confidence interval by looking at the quantiles of the distribution of $\hat{\theta}$, G_n. Since we have bootstrap samples from G_n, we can estimate these quantiles!. So if we order the bootstrap estimates

$$\hat{\theta}^*_{(1)} < \hat{\theta}^*_{(2)} < \cdots < \hat{\theta}^*_{(B)} \,,$$

and set L to be the $(\alpha/2)$th ordered statistic and U to be $(1-\alpha/2)$th order statistic, we get:

$$L = \hat{\theta}^*_{\lfloor \alpha/2 * B \rfloor} \quad \text{and } U = \quad \hat{\theta}^*_{\lfloor 1-\alpha/2 * B \rfloor} \,.$$

Then $\left(\hat{\theta}^*_{\lfloor \alpha/2 * B \rfloor}, \hat{\theta}^*_{\lfloor 1-\alpha/2 * B \rfloor}\right)$, is a $100(1-\alpha)\%$ bootstrap confidence interval.

Example 1 (Mean of Gamma$(a,1)$). Let $X_1, \ldots, X_n \overset{iid}{\sim}$ Gamma$(a,1)$. The mean of this distribution is $\theta = a$. Consider estimating θ with the sample mean

$$\hat{\theta} = \bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \sim G_n$$

Although a central limit theorem holds, so that $G_n \approx$ normally distribution for large n. However, we may not have many samples available in order to make confidence intervals. Thus, instead here we implement the nonparametric bootstrap:

$$\begin{gathered} X^*_{11}, X^*_{21}, \ldots, X^*_{n1} \Rightarrow \bar{X}^*_1 \\ X^*_{12}, X^*_{22}, \ldots, X^*_{n2} \Rightarrow \bar{X}^*_2 \\ \vdots \\ X^*_{1B}, X^*_{2B}, \ldots, X^*_{nB} \Rightarrow \bar{X}^*_B \\ . \end{gathered}$$

And find $\alpha/2$ and $1-\alpha/2$ sample quantiles from $\bar{X}^*_i, i = 1, \ldots, B$.

1.1.2 Parametric Bootstrap

Suppose $X_1, \ldots, X_n \sim F(\theta)$, where θ is a parameter we can estimate. Let $\hat{\theta}$ be a chosen estimator of θ. Instead of resampling within our data, in *parametric* bootstrap, we use our estimator of θ to obtain computer generated samples from $F(\hat{\theta})$:

$$\begin{aligned} X_{11}^*, X_{21}^*, \ldots, X_{n1}^* &\sim F(\hat{\theta}) \Rightarrow \hat{\theta}_1^* \\ X_{12}^*, X_{22}^*, \ldots, X_{n2}^* &\sim F(\hat{\theta}) \Rightarrow \hat{\theta}_2^* \\ &\vdots \\ X_{1B}^*, X_{2B}^*, \ldots, X_{nB}^* &\sim F(\hat{\theta}) \Rightarrow \hat{\theta}_B^* \end{aligned}$$

And again, we find the $\alpha/2$ and $1-\alpha/2$ quantiles of the $\hat{\theta}_i^*$s so that $\left(\hat{\theta}^*_{\lfloor \alpha/2 * B \rfloor}, \hat{\theta}^*_{\lfloor 1-\alpha/2 * B \rfloor}\right)$, is a $100(1-\alpha)\%$ bootstrap confidence interval.

Example 2 (Mean of Gamma$(a, 1)$). Let $X_1, \ldots, X_n \overset{iid}{\sim}$ Gamma$(a, 1)$. And let $\hat{\theta} = \bar{X}$ be the chosen estimator of a. A parametric bootstrap estimator does:

$$\begin{aligned} X_{11}^*, X_{21}^*, \ldots, X_{n1}^* &\sim \text{Gamma}(\bar{X}, 1) \Rightarrow \bar{X}_1^* \\ X_{12}^*, X_{22}^*, \ldots, X_{n2}^* &\sim \text{Gamma}(\bar{X}, 1) \Rightarrow \bar{X}_2^* \\ &\vdots \\ X_{1B}^*, X_{2B}^*, \ldots, X_{nB}^* &\sim \text{Gamma}(\bar{X}, 1) \Rightarrow \bar{X}_B^* \,. \end{aligned}$$

And find $\alpha/2$ and $1-\alpha/2$ sample quantiles from $\bar{X}_i^*, i = 1, \ldots, B$.

Let us implement this Gamma example in detail. Notice that a CLT holds here with asymptotic variance a, so that as $n \to \infty$.

$$\sqrt{n}(\bar{X} - a) \xrightarrow{d} N(0, a) \,.$$

So that an asymptotic 95% confidence interval is

$$\bar{X} \pm z_{.975}\sqrt{\frac{\bar{X}}{n}}\,.$$

We will make four comparisons:

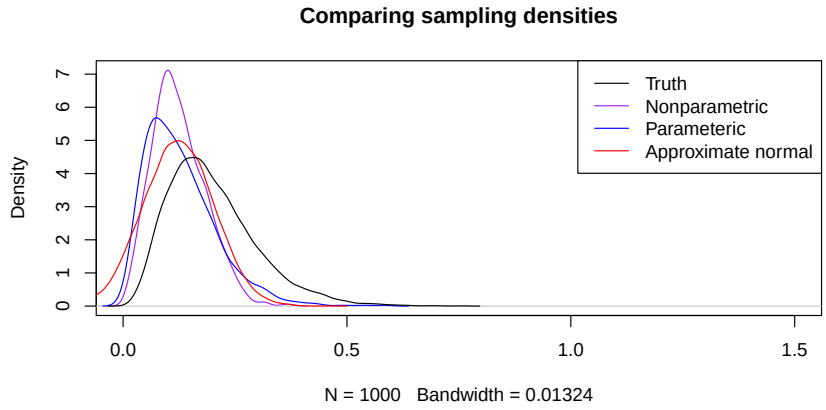
- The empirical distribution of nonparametric bootstrap estimates
- The empirical distribution of parametric bootstrap estimates
- The large sample normal distribution $N(a, a/n)$
- The true sampling distribution

```
##########################################
## Bootstrap for the Gamma distribution
##########################################
set.seed(10)
n <- 20
a <- .20
my.samp <- rgamma(n, shape = a, rate = 1)


barx <- mean(my.samp)


B <- 1e3 # number of bootstrap samples. 1e3 is standard.
boot.np <- numeric(length = B)
boot.p <- numeric(length = B)
for(b in 1:B)
{
  boot.samp.np <- sample(my.samp, replace = TRUE) # NP Bootstramp samples
  boot.np[b] <- mean(boot.samp.np) # NP Bootstrap estimator
```

```
  boot.samp.p <- rgamma(n, shape = barx, rate = 1) #parametric bootstrap
      samples
  boot.p[b] <- mean(boot.samp.p) # P bootstrap estimator
}

# 95% Bootstrap confidence interval
quantile(boot.np, probs = c(.025, .975)) # nonparameteric
#     2.5%     97.5%
#0.03058783 0.24917142

quantile(boot.p, probs = c(.025, .975)) #parametric
#     2.5%     97.5%
#0.02262155 0.31736283

# 95% asymptotic CI
c( barx - qnorm(.975)*sqrt(barx/n), barx + qnorm(.975)*sqrt(barx/n) )
#[1] -0.03029098 0.27848545

# Simulate repeated estimates to construct a 95% CI
true.samp <- numeric(length = 1e4)
for(i in 1:1e4)
{
  samp <- rgamma(n, shape = a, rate = 1)
  true.samp[i] <- mean(samp)
}
quantile(true.samp, probs = c(.025, .975))
#     2.5%     97.5%
#0.05446254 0.43878948
```

Since the sample size is low, all methods give different estimates. Certainly the CI based on the CLT is quite off since it proposed negative values in the interval, which is invalid. The parametric bootstrap is the closest to the truth. We can also compare their empirical densities against the truth.

```
plot(density(boot.np), col = "purple", xlim = c(0,1.5),
     main = "Comparing sampling densities")
lines(density(boot.p), col = "blue")
lines(density(rnorm(1e4, mean = barx, sd = sqrt(barx/n))), col = "red")
lines(density(true.samp))
legend("topright",lty = 1, legend = c("Truth", "Nonparametric",
    "Parameteric", "Approximate normal"), col = c("black", "purple", "blue",
    "red"))
```



Clearly the nonparametric estimator has lower than the real variance and this is the consequence of the really small sample size. The approximate normal interval assumes a symmetric sampling distribution, which is clearly not true.

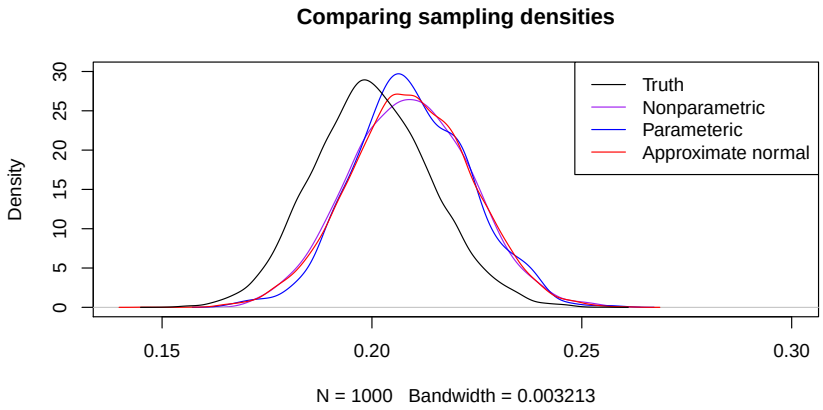
We can repeat the same thing for a larger n. Here, the asymptotic normal distribution coincides with the sampling distributions obtained via both bootstrap methods.

```r
n <- 1000
a <- .20
my.samp <- rgamma(n, shape = a, rate = 1)

barx <- mean(my.samp)

B <- 1e3 # number of bootstrap samples
boot.np <- numeric(length = B)
boot.p <- numeric(length = B)

for(b in 1:B)
{
  boot.samp.np <- sample(my.samp, replace = TRUE) # NP Bootstramp samples
  boot.np[b] <- mean(boot.samp.np) # NP Bootstrap estimator

  boot.samp.p <- rgamma(n, shape = barx, rate = 1) #parametric bootstrap
      samples
  boot.p[b] <- mean(boot.samp.p) # P bootstrap estimator
}

# 95% Bootstrap confidence interval
quantile(boot.np, probs = c(.025, .975)) # nonparameteric
#     2.5%    97.5%
#0.1812819 0.2370416

quantile(boot.p, probs = c(.025, .975)) #parametric
#     2.5%    97.5%
#0.1832509 0.2374146
```

```r
# 95% asymptotic CI
c( barx - qnorm(.975)*sqrt(barx/n), barx + qnorm(.975)*sqrt(barx/n) )
#[1] 0.1809254 0.2376330


# Simulate repeated estimates to construct a 95% CI
true.samp <- numeric(length = 1e4)
for(i in 1:1e4)
{
  samp <- rgamma(n, shape = a, rate = 1)
  true.samp[i] <- mean(samp)
}
quantile(true.samp, probs = c(.025, .975))
#     2.5%    97.5%
#0.1736390 0.2289925
```

All the estimated intervals are similar, however, they differ slightly from the true interval, which is natural, since the true interval is centered around the ground truth, and we are centered around $\bar{X}$.

```r
plot(density(boot.np), col = "purple", xlim = c(0,1.5),
     main = "Comparing sampling densities")
lines(density(boot.p), col = "blue")
lines(density(rnorm(1e4, mean = barx, sd = sqrt(barx/n))), col = "red")
lines(density(true.samp))
legend("topright",lty = 1, legend = c("Truth", "Nonparametric",
    "Parameteric", "Approximate normal"), col = c("black", "purple", "blue",
    "red"))
```



2 Questions to think about

- We set $B = 1000$ in our experiments. What would happen if we increase of decrease B?
- How will you used bootstrapping to obtain confidence intervals for bridge regression coefficients?
- Do we need bootstrapping to obtain confidence intervals for ridge regression estimates?
- Repeat the Gamma example to obtain the sampling distribution of the sample median minus sample mean. That is

$$\text{median}(X_1, \ldots, X_n) - \bar{X} \, .$$

We don't have a CLT here so we have to resort to only bootstrap.

MTH 511a - 2021: L31 - Stochastic Gradient Ascent

Instructor: Dootika Vats

1 Stochastic optimization methods

We go back to optimization this week. The reason we took a break from optimization is because we will focus on stochastic optimization methods, which will lead the discussion into other stochastic methods.

We will cover two topics:

1. Stochastic gradient ascent
2. Simulated annealing

Our goal is the same as before: for an objective function $f(\theta)$, our goal is to find

$$\theta^* = \arg\max_{\theta} f(\theta)\,.$$

1.1 Stochastic gradient ascent

Recall, in order to maximize the objective function the gradient ascent algorithm does the following update:

$$\theta_{(k+1)} = \theta_{(k)} + t\nabla f(\theta_{(k)}) \,,$$

where $\nabla f(\theta_{(k)})$ is the gradient vector. Now, since in many statistics problems, the objective function is the log-likelihood (for some density $\tilde{f}$),

$$f(\theta) = \sum_{i=1}^{n} \log \tilde{f}(\theta|x_i) = \frac{1}{n}\sum_{i=1}^{n} n \log \tilde{f}(\theta|x_i) \Rightarrow \nabla f(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla \left[n \log \tilde{f}(\theta|x_i) \right] \,.$$

That is, in order to implement a gradient ascent step, the gradient of the log-likelihood is calculated for the whole data. However, consider the following two situations

- the data size n and/or dimension of θ are prohibitively large so that calculating the full gradient multiple times is infeasible
- the data is not available at once! In many online data situations, the full data set is not available, but comes in sequentially. Then, the full data gradient vector is not available.

In such situations, when the full gradient vector is unavailable, our goal is to *estimate* the gradient. Suppose i_k is a randomly chosen index in $\{1, \ldots, n\}$. Then

$$\mathbb{E}\left[\underbrace{\nabla \left[n \log \tilde{f}(\theta|x_{i_k}) \right]}_{\text{Estimate of full gradient}} \right] = \frac{1}{n}\sum_{i=1}^{n} \left[n \nabla \log \tilde{f}(\theta|x_i) \right] \,.$$

Thus, $\nabla n \log \left[\tilde{f}(\theta|x_{i_k}) \right]$ is an unbiased estimator of the complete gradient, but uses *only one data point*. Replacing the complete gradient with this estimate yields the

stochastic gradient ascent update:

$$\theta_{(k+1)} = \theta_{(k)} + t\left[\nabla\left[n \log \tilde{f}(\theta_{(k)}|x_{i_k})\right]\right] ,$$

where i_k is a randomly chosen index. This randomness in choosing the index makes this a *stochastic algorithm*.

- advantage: it is much cheaper to implement since only one-data point is required for gradient evaluation
- disadvantage it may require larger k for convergence to the optimal solution
- disadvantage as k increases, $\theta_{(k+1)} \not\to \theta^*$. Rather, after some initial steps, $\theta_{(k+1)}$ oscillates around θ^*.

After K iterations, the final estimate of θ^* is

$$\hat{\theta}^* = \frac{1}{K}\sum_{k=1}^{K} \theta_{(k+1)} .$$

However, since each step involves estimating data gradient, variability in updates of $\theta_{(k)}$ is larger than using gradient ascent. To stabilize this behavior, often `mini-batch` stochastic gradient is used.

1.1.1 Mini-batch stochastic gradient ascent

Let I_k be a random subset of $\{1, \ldots, n\}$ of size b. Then, the mini-batch stochastic gradient ascent algorithm implements the following update:

$$\theta_{(k+1)} = \theta_{(k)} + t \left[\frac{1}{b} \sum_{i \in I_k} \nabla \left[n \log \tilde{f}(\theta_{(k)} | x_i) \right] \right] .$$

The mini-batch stochastic gradient estimate of θ^* after K updates is

$$\hat{\theta}^* = \frac{1}{K} \sum_{k=1}^{K} \theta_{(k)} .$$

There are not a lot of clear rules about terminating the algorithm in stochastic gradient. Typically, the number of iterations $K = n$, so that one full pass at the data is implemented.

1.1.2 Logistic regression

Recall the logistic regression setup where for a response Y and a covariate matrix X,

$$Y_i \sim \text{Bern} \left(\frac{e^{x_i^T \beta}}{1 + e^{x_i^T \beta}} \right) .$$

In order to find the MLE for β, we obtain the log-likelihood.

$$L(\beta | Y) = \prod_{i=1}^{n} (p_i)^{y_i} (1 - p_i)^{1 - y_i}$$

$$\Rightarrow n \log \tilde{f}(\beta) = -n \sum_{i=1}^{n} \log\left(1 + \exp(x_i^T \beta)\right) + n \sum_{i=1}^{n} y_i x_i^T \beta$$

Taking derivative:

$$\nabla \left[n \log \tilde{f}(\beta) \right] = n \sum_{i=1}^{n} x_i \left[y_i - \frac{e^{x_i^T \beta}}{1 + e^{x_i^T \beta}} \right].$$

As noted earlier, the target objective is concave, thus a global optima exists and the gradient ascent algorithm will converge to the MLE. We will implement the stochastic gradient ascent algorithm here.

```
################################################
## MLE for logistic regression
## Using stochastic gradient ascent
################################################


f.gradient <- function(y, X, beta)
{
  n <- dim(X)[1]
  beta <- matrix(beta, ncol = 1)
  pi <- exp(X %*% beta) / (1 + exp(X%*%beta))
  rtn <- colSums(X* as.numeric(y - pi))
  return(n*rtn)
}



################################################
## The following is a general function that
## implements the regular gradient ascent
## the stochastic gradient ascent and
```

```
## mini-batch stochastic gradient ascent
################################################
SGA <- function(y, X, batch.size = dim(X)[1], t = .1, max.iter = dim(X)[1],
    adapt = FALSE)
{
  p <- dim(X)[2]
  n <- dim(X)[1]

  # create the mini-batches
  permutation <- sample(1:n, replace = FALSE)
  K <- floor(n/batch.size)
  batch.index <- split(permutation, rep(1:K, length = n, each = n/K))

  # index for choosing the mini-batch
  count <- 1

  beta_k <- rep(0, p) # start at all 0s
  track.gradient <- matrix(0, nrow = max.iter, ncol = p)
  track.gradient[1,] <- f.gradient(y = y, X= X, beta = beta_k)

  # saving the running mean of the estimates of theta^*
  mean_beta <- rep(0,p)

  # tk: in case we want t_k
  tk <- t

  # ideally, we will have a while loop here, but
  # I have written this to always complete some max.iter steps
  for(iter in 1:max.iter)
```

```
 {
   count <- count+1


   if(adapt) tk <- t/(sqrt(iter)) # in case t_k
   if(count %% K == 0) count <- count%%K +1 # when all batches finish,
       restart the batches
   if(iter %% (max.iter/10) == 0) print(iter) #feedback


   # batch of data
   y.batch <- y[batch.index[[count]] ]
   X.batch <- matrix(X[batch.index[[count]], ], nrow = batch.size)


   # SGA step
   beta_k = beta_k + tk* f.gradient(y = y.batch, X = X.batch, beta =
       beta_k)/batch.size


   # saving overall estimates and running gradients for demonstration
   mean_beta <- (beta_k + mean_beta*(iter - 1))/(iter)
   if(batch.size == n)
   {
     est <- beta_k
   }else{
     est <- mean_beta
   }
   track.gradient[iter,] <- f.gradient(y = y, X = X, beta = est)/n
 }
 rtn <- list("iter" = iter, "est" = est, "grad" = track.gradient[1:iter,])
 return(rtn)
}
```

Next, I will generate data from the logistic regression model in order to demonstrate the performance of the stochastic gradient ascent algorithm.

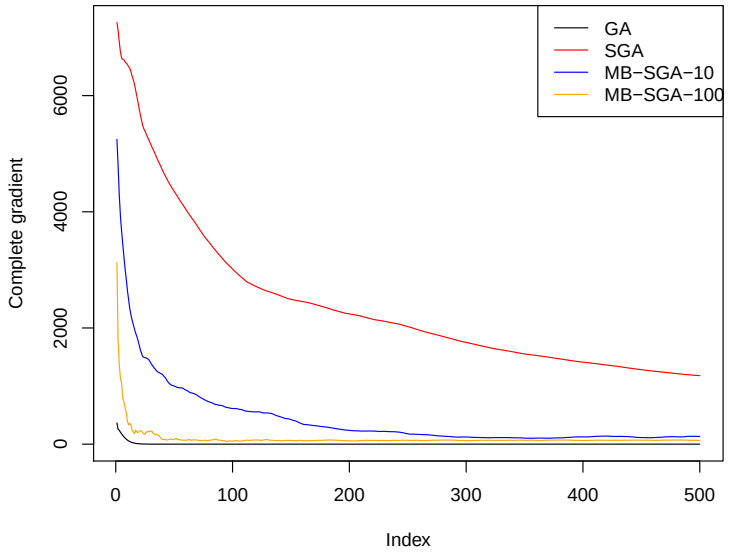
```
# Generating data for demonstration
set.seed(10)
p <- 5
n <- 1e4
X <- matrix(rnorm(n*(p-1)), nrow = n, ncol = p-1)
X <- cbind(1, X)
beta <- matrix(rnorm(p, 0, sd = 1), ncol = 1)
p <- exp(X %*% beta)/(1 + exp(X%*%beta))
y <- rbinom(n, size = 1, prob = p)
```

We implement the stochastic gradient ascent algorithm and keep a track of the running average of the estimate of θ^*, $\hat{\theta}_k^*$, and track the value of the complete gradient at that value $\|\nabla f(\hat{\theta}_k^*)\|$, the full gradient. A running plot of these should tend to 0 as k increases. We will implement the original gradient ascent, stochastic gradient ascent, and mini-batch gradient ascent algorithm.

I first run this for tuned values of t.

```
# Tuned value of t
ga <- SGA(y, X, batch.size = 1e4, t = .0015, max.iter = 1e3)
b1 <- SGA(y, X, batch.size = 1, t = .1, max.iter = ga$iter)
b10 <- SGA(y, X, batch.size = 10, t = .1, max.iter = ga$iter)
b100 <- SGA(y, X, batch.size = 100, t = .1, max.iter = ga$iter)

index <- 1:500
plot(apply(ga$grad[index,], 1, function(t) sum(abs(t))), type = 'l', ylim =
```

```
   c(0,max(apply(b1$grad[,], 1, function(t) sum(abs(t))))), ylab =
   "Complete gradient")
lines(apply(b1$grad[index,], 1, function(t) sum(abs(t))), col = "red")
lines(apply(b10$grad[index,], 1, function(t) sum(abs(t))), col = "blue")
lines(apply(b100$grad[index,], 1, function(t) sum(abs(t))), col = "orange")
legend("topright", col = c("black", "red", "blue", "orange"), legend =
   c("GA", "SGA", "MB-SGA-10", "MB-SGA-100"), lty = 1)
```

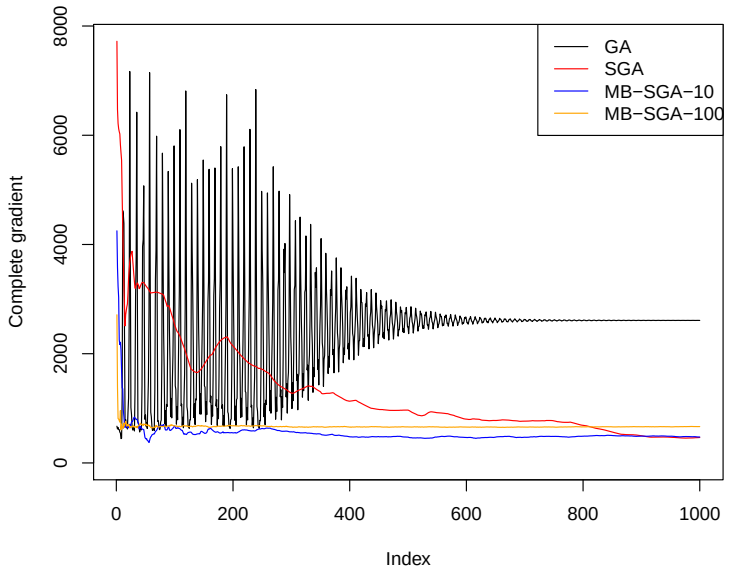


We see that the original stochastic gradient ascent algorithm is slow to converge although it is the cheapest. The mini-batches are far more stable.

Next, we repeat the same algorithm with different choices of t. These chosen choices of t are too large so that all the algorithm no longer perform well and essentially oscillate locally.

```
# all bad values of t
ga <- SGA(y, X, batch.size = n, t = .005, max.iter = 1e3)
```

```
b1 <- SGA(y, X, batch.size = 1, t = 1, max.iter = ga$iter)
b10 <- SGA(y, X, batch.size = 10, t = 1, max.iter = ga$iter)
b100 <- SGA(y, X, batch.size = 100, t = 1, max.iter = ga$iter)

index <- 1:1000
plot(apply(ga$grad[index,], 1, function(t) sum(abs(t))), type = 'l', ylim =
    c(0,max(apply(b1$grad[,], 1, function(t) sum(abs(t))))), ylab =
    "Complete gradient")
lines(apply(b1$grad[index,], 1, function(t) sum(abs(t))), col = "red")
lines(apply(b10$grad[index,], 1, function(t) sum(abs(t))), col = "blue")
lines(apply(b100$grad[index,], 1, function(t) sum(abs(t))), col = "orange")
legend("topright", col = c("black", "red", "blue", "orange"), legend =
    c("GA", "SGA", "MB-SGA-10", "MB-SGA-100"), lty = 1)
```

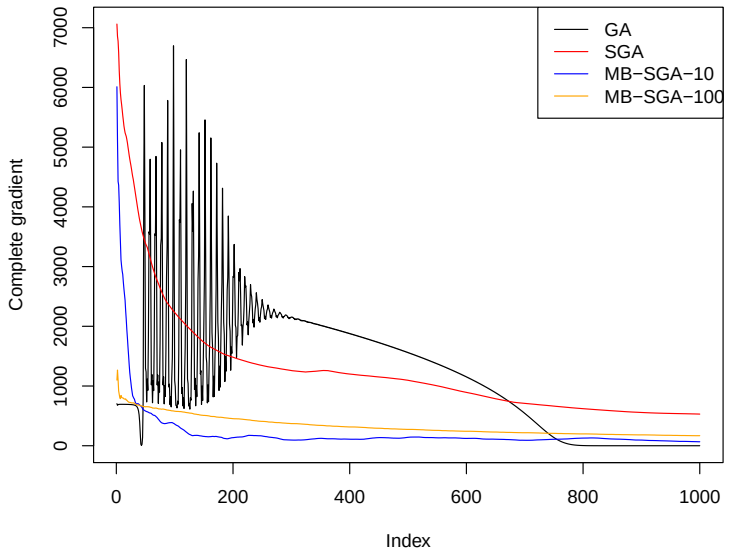


The original gradient ascent algorithm oscillates drastically. The stochastic versions seem to be converging away from 0 as well. Thus, the value of t is critical to imple-

menting the (stochastic) gradient ascent algorithms in a stable way.

Note that the oscillations occurs due to a large value of t. But which value of t which be large and which will be small is difficult to assess in the beginning. It is thus useful to choose a decreasing sequence t_k that reduces the step size and avoids long durations of getting stuck in an oscillation. Here we will use $t_k = t/\log(k)$.

```
ga <- SGA(y, X, batch.size = n, t = .05, max.iter = 1e3, adapt = TRUE)
b1 <- SGA(y, X, batch.size = 1, t = 1, max.iter = ga$iter, adapt = TRUE)
b10 <- SGA(y, X, batch.size = 10, t = 1, max.iter = ga$iter, adapt = TRUE)
b100 <- SGA(y, X, batch.size = 100, t = 1, max.iter = ga$iter, adapt = TRUE)

index <- 1:1000
plot(apply(ga$grad[index,], 1, function(t) sum(abs(t))), type = 'l', ylim =
   c(0,max(apply(b1$grad[,], 1, function(t) sum(abs(t))))), ylab =
   "Complete gradient")
lines(apply(b1$grad[index,], 1, function(t) sum(abs(t))), col = "red")
lines(apply(b10$grad[index,], 1, function(t) sum(abs(t))), col = "blue")
lines(apply(b100$grad[index,], 1, function(t) sum(abs(t))), col = "orange")
legend("topright", col = c("black", "red", "blue", "orange"), legend =
   c("GA", "SGA", "MB-SGA-10", "MB-SGA-100"), lty = 1)
```



Note here that although the algorithm begins oscillate, due to decreasing step-sizes, the algorithm escapes out of local oscillations.

2 Questions to think about

1. Can we change t_k adaptively as the algorithm goes along?

2. Why do we take the sample mean of the sequence of $\theta_(k)$ in SGA?

MTH 511a - 2021: L32: Simulated Annealing

Instructor: Dootika Vats

1 Stochastic optimization methods

Last lecture we went over the stochastic gradient ascent algorithm: the merit of this algorithm was its use in online sequential data and for large data set problem.

This lecture focuses on simulated annealing, an algorithm particularly useful for non-concave objective functions. Our goal is the same as before: for an objective function $f(\theta)$, our goal is to find

$$\theta^* = \arg\max_{\theta} f(\theta)\,.$$

1.1 Simulated annealing

Recall that when the objective function is non-concave, all of the methods we've discussed cannot escape out of a local maxima. This creates challenges in obtaining global maximas. This is where the method of *simulated annealing* has an advantage over other methods.

Consider an objective function $f(\theta)$ to maximize. Note that maximizing $f(\theta)$ is equivalent to maximizing $\exp(f(\theta))$. The idea in simulated annealing is that, instead of trying to find a maxima directly, we will obtain samples from the density

$$\pi(\theta) \propto \exp(f(\theta))\,.$$

Samples collected from $\pi(\theta)$ are likely to be from areas near the maximas. However, obtaining samples from $\pi(\theta)$ means there will be samples from low probability areas as well. So how do we force samples to come from areas near the maximas?

Consider for $T > 0$,

$$\frac{de^{f(\theta)/T}}{d\theta} = e^{f(\theta)/T}\frac{f'(\theta)}{T},$$

which has the same roots and direction as $f(\theta)$. Thus,

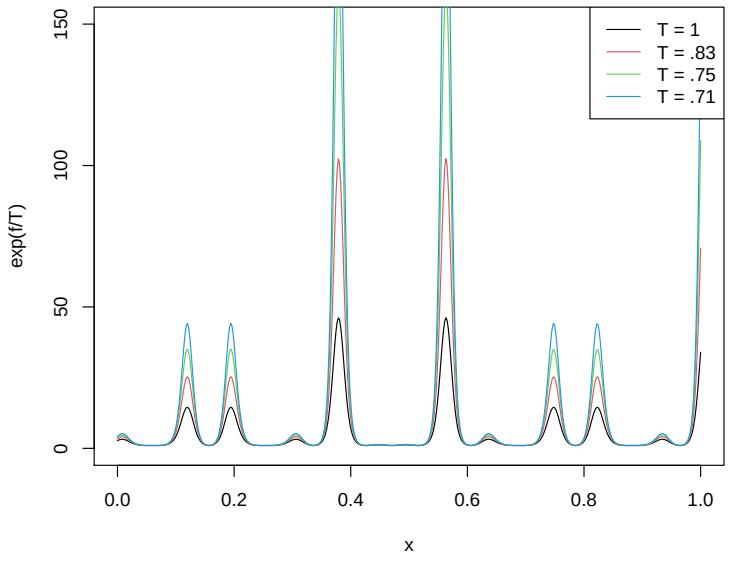
$$\arg\max_{\theta} f(\theta) = \arg\max_{\theta} e^{\{f(\theta)/T\}}.$$

For $0 < T < 1$, the objective function's modes are exaggerated there-by amplifying the maximas. This feature will help us in trying to "push" the sampling to areas of high-probability.

Example 1. Consider the following objective function

$$f(\theta) = [\cos(50\theta) + \sin(20\theta)]^2 I(0 < \theta < 1)$$

Below is a plot of $e^{f(\theta)/T}$ for various values of T.



In simulated annealing, this feature is utilized so that every subsequent sample is drawn

from an increasingly concentrated distribution. That is, at a time point k, a sample will be drawn from

$$\pi_{k,T}(\theta) \propto e^{f(\theta)/T_k},$$

where T_k is a decreasing sequence.

How do we generate these samples?

Certainly, we can try and use accept-reject or another Monte Carlo sampling method, but such methods cannot be implemented generally. Note that for any θ', θ

$$\frac{\pi_{k,T}(\theta')}{\pi_{k,T}(\theta)} = \exp\left\{\frac{f(\theta') - f(\theta)}{T_k}\right\}.$$

Let G be a proposal distribution with density $g(\theta'|\theta)$ so that $g(\theta'|\theta) = g(\theta|\theta')$. Such a proposal distribution is a symmetric proposal distribution. Further, given a value of θ, θ' is sampled using density $g(\cdot|\theta)$.

Algorithm 1 Simulated Annealing algorithm

1: For $k = 1, \ldots, N$, repeat the following:
2: Generate $\theta' \sim G(\cdot|\theta_k)$ and generate $U \sim U(0,1)$
3: Let $\alpha = \min\left\{1, \exp\left\{\frac{f(\theta') - f(\theta)}{T_k}\right\}\right\}$.
4: If $U < \alpha$, then let $\theta_{k+1} = \theta'$
5: Else $\theta_{k+1} = \theta_k$.
6: Update T_{k+1}
7: Store θ_{k+1} and $e^{f(\theta_{k+1})}$.
8: Return $\theta^* = \theta_{k^*}$ where k^* is such that $k^* = \arg\max_k e^{f(\theta_{k+1})}$

Thus, if the proposed value is such that $f(\theta') > f(\theta)$, then $\alpha = 1$ and the move is always accepted. The reason simulated annealing works is because when θ' is such that $f(\theta') < f(\theta)$, even then, the move is accepted with probability α.

Thus, there is always a chance to move out of local maximas.

Essentially, each θ_k is approximately distributed as $\pi_{k,T}$, and as $T_k \to 0$, $\pi_{k,T}$ puts more

and more mass on the maximas, thus, θ_k will typically be getting increasingly closer to θ^*.

- Typically, $G(\cdot|\theta)$ is $U(\theta - r, \theta + r)$ or $N(\theta, r)$ which are both valid symmetrical proposals. The parameter r dictates how far/close the proposed values will be.
- T_k is often called the *temperature* parameter. A common value of $T_k = d/\log(k)$ for some constant d.

Example 2 (continued...). We implement the simulated annealing algorithm for:

$$f(\theta) = [\cos(50\theta) + \sin(20\theta)]^2 \, I(0 < \theta < 1)$$

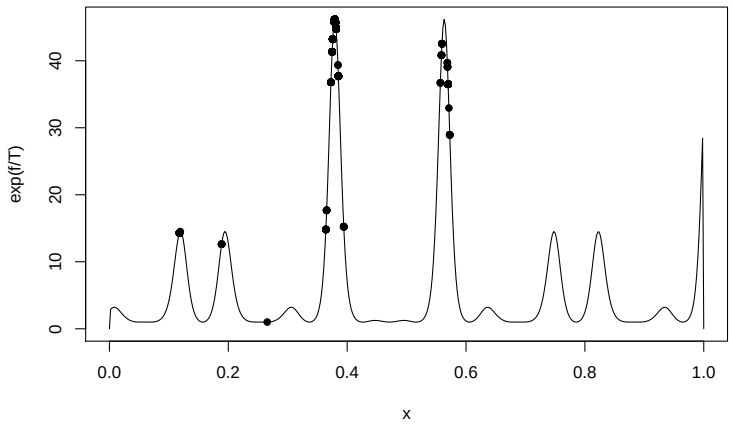
The true $\theta^* \approx .379$.

```
####################################
## Simulated Annealing
## Demonstrative example
####################################
fn <- function(x, T = 1)
{
  h <- ( cos(50*x) + sin(20*x) )^2
  exp(h/T)* (0 < x & x < 1)
}

simAn <- function(N = 10, r = .5)
{
  x <- numeric(length = N)
  x[1] <- runif(1)

  for(k in 2:N)
  {
    # U(x - r, x + r)
    a <- runif(1, x[k-1] - r, x[k-1] + r)
    T <- 1/(log(k))
    ratio <- fn(a,T)/fn(x[k-1], T)
    if( runif(1) < ratio)
    {
      x[k] <- a # accept
    } else{
```

```
      x[k] <- x[k-1] # reject, so stay
    }
  }
 return(x)
}
```

Below I implement the algorithm for 500 steps and return the estimate of θ^*. I also plot the values of θ obtained.

```
N <- 500
sim <- simAn(N = N)
sim[which.max(fn(sim))] # theta^*
[1] 0.3792136

x <- seq(0, 1, length = 5e2)
plot(x, fn(x), type = 'l')
points(sim, fn(sim), pch = 16, col = 1)
```

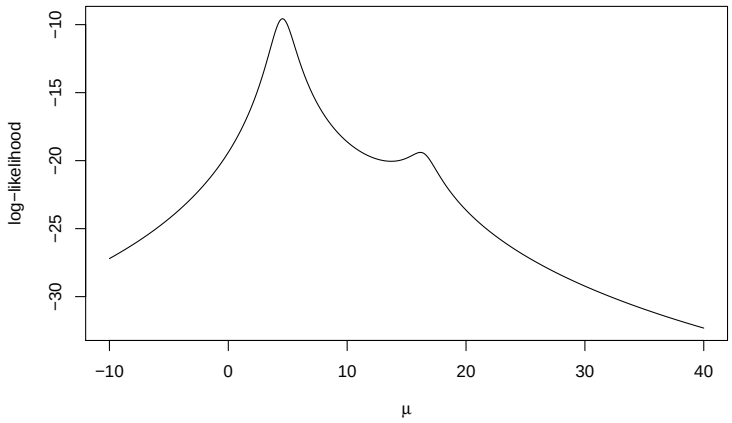


Example 3 (Location Cauchy). Recall the location Cauchy example discussed in Week 6 Lecture 15. The objective function is the log-likelihood of the location Cauchy distribution with mode at $\mu \in \mathbb{R}$. The goal is to find the MLE for μ.

$$f(x|\mu) = \frac{1}{\pi} \frac{1}{(1 + (x - \mu)^2)} .$$

The log-likelihood is

$$l(\mu) := \log L(\mu|X) = -n \log \pi - \sum_{t=1}^{n} \log(1 + (X_i - \mu)^2) \,.$$



Recall that for the dataset generated, the log-likelihood (above) was not concave and presented many local maxima. This caused Newton-Raphson to possible diverge/converge to minima/local maxima and caused the gradient ascent algorithm to converge to local maxima. We will implement the simulated annealing algorithm here with $G = U(\theta - r, \theta + r)$.

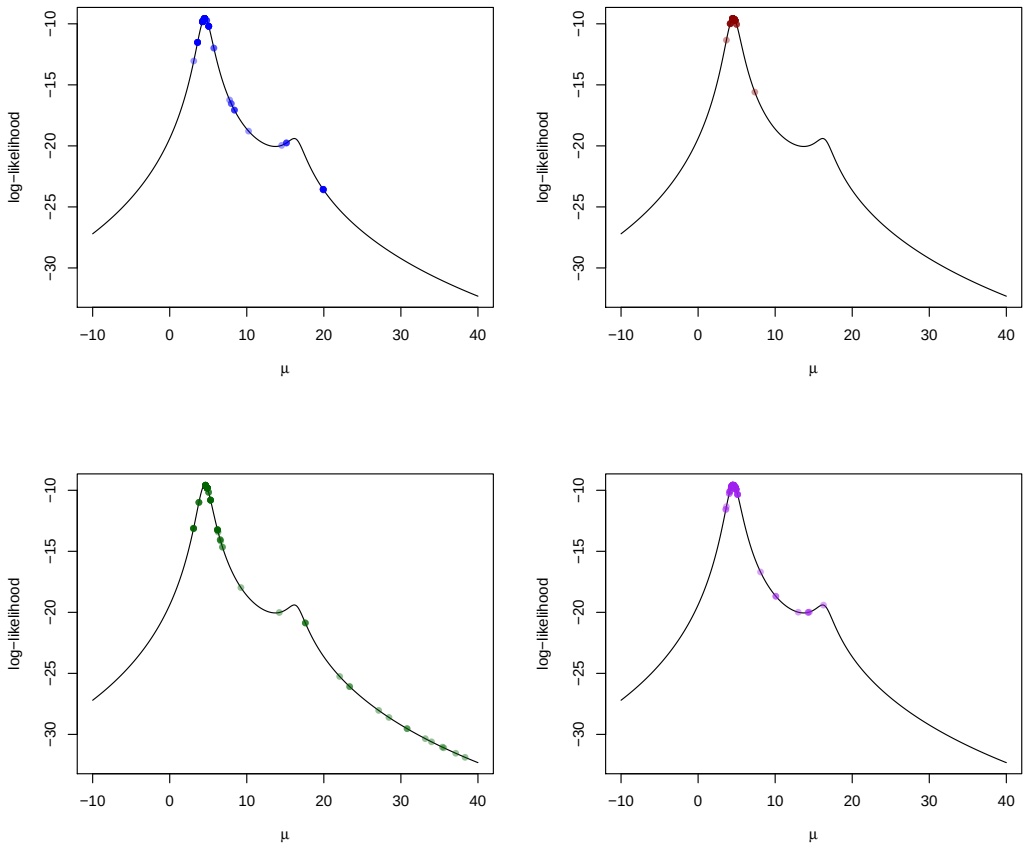
```
## Function calculates the exp(like/T)
log.like <- function(mu, X, T = 1)
{
  n <- length(X)
  rtn <- -n*log(pi) - sum( log(1 + (X - mu)^2) )
  return(exp(rtn/T))
}

# Simulated annealing algorithm
simAn <- function(N = 10, r = .5)
{
  x <- numeric(length = N)
  x[1] <- runif(1, min = -10, max = 40)
```

```
  fn.value <- numeric(length = N)

  fn.value[1] <- log.like(mu = x[1], X, T = 1)
  for(k in 2:N)
  {
    a <- runif(1, x[k-1] - r, x[k-1] + r)
    T <- 1/(1 + log(log(k)))
    ratio <- log.like(mu = a, X, T)/log.like(mu = x[k-1], X, T)
    if( runif(1) < ratio)
    {
      x[k] <- a
    } else{
      x[k] <- x[k-1]
    }
    fn.value[k] <- log.like(mu = x[k], X, T = 1)
  }
  x
  return(list("x" = x, "fn.value" = fn.value))
}
```

I run the algorithm for 100 steps from four randomly chosen starting points.

```
par(mfrow = c(2,2))

# Four different runs all converge.
sim <- simAn(N = 1e2, r = 5)
plot(mu.x, ll.est, type = 'l', ylab = "log-likelihood", xlab =
    expression(mu))
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("blue", alpha
    = .4))

sim <- simAn(N = 1e2, r = 5)
plot(mu.x, ll.est, type = 'l', ylab = "log-likelihood", xlab =
    expression(mu))
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("darkred",
    alpha = .4))

sim <- simAn(N = 1e2, r = 5)
```

```
plot(mu.x, ll.est, type = 'l', ylab = "log-likelihood", xlab =
    expression(mu))
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("darkgreen",
    alpha = .4))

sim <- simAn(N = 1e2, r = 5)
plot(mu.x, ll.est, type = 'l', ylab = "log-likelihood", xlab =
    expression(mu))
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("purple",
    alpha = .4))
```



Note the simulated annealing algorithm is able to escape out of local modes and head towards the global maxima. However, the above algorithm is implemented only after tuning r. Tuning r can be challenging.

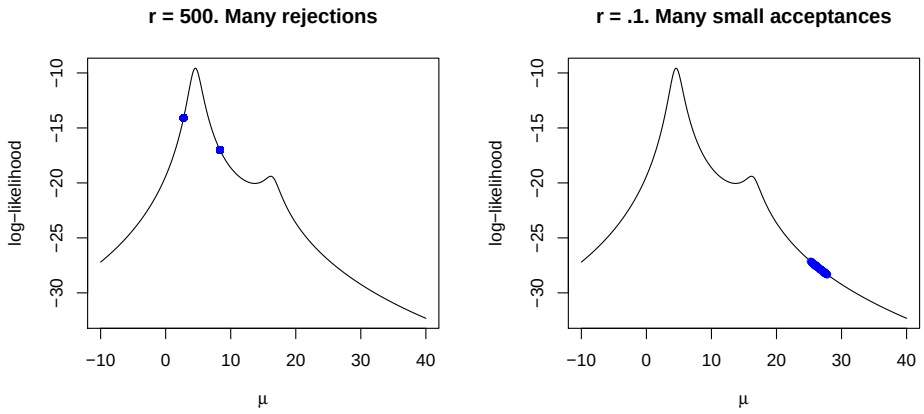
- Large r: values too far away are proposed where the objective function is very

low. These values will get rejected and the algorithm will not move.

- Small r: values too close are proposed where the change in the objective function is small. These values are often accepted, but the algorithm makes very tiny jumps.

Below are runs of the simulated annealing algorithm with r chosen to be too high (500) and too low (.1).

```
par(mfrow = c(1,2))
## Different values of r
# very large r
sim <- simAn(N = 1e3, r = 500)
plot(mu.x, ll.est, type = 'l', main = "r = 500. Many rejections", ylab =
    "log-likelihood", xlab = expression(mu))
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("blue", alpha
    = .2))

#very small r
plot(mu.x, ll.est, type = 'l', main = "r = .1. Many small acceptances",
    ylab = "log-likelihood", xlab = expression(mu))
sim <- simAn(N = 1e3, r = .1)
points(sim$x, log(sim$fn.value), pch = 16, col = adjustcolor("blue", alpha
    = .2))
```



2 Questions to think about

- How do you think this algorithm will scale in higher dimensions? Try implementing simulated annealing for a Lasso optimization problem.
- Is there any benefit to having $T > 1$?

MTH 511a - 2021: L33 - Recap

Instructor: Dootika Vats

1 A recap of data analysis techniques so far

After this lecture, we shift our focus to computation for Bayesian models, so before we go along there, this lecture is a quick recap of all the techniques which we've learned. We've covered three areas of topics to help us with analyzing data:

1. Estimation (optimization)
2. Model selection (cross-validation)
3. Model inference (bootstrapping)

Optimization in statistics is predominantly motivated from maximizing likelihoods or penalized likelihood functions. In the next week or so, we will also look into maximizing *posterior* distributions (but more about that later).

When obtaining closed-form expressions of parameters that maximize the (penalized) likelihood is challenging, numerical optimization techniques are used.

Step 0: Before you venture into any optimization method, it is crucial to *try* to check whether the objective function is concave. If it is concave, then you know a global maxima exists. If not concave, you know local maximas will exists.

- Non-stochastic Gradient-Based methods
  - Newton-Raphson
    - Requires explicitly calculating the Hessian and inverting the Hessian.
    - Can be expensive in large dimensions
    - For concave objective functions, converges the fastest and requires little to no tuning.
    - For non-concave objective functions, it may converge to a local maxima, converge to a local minima, or diverge.
    - Examples - logistic regression.
  - Gradient-Ascent
    - Requires only the gradient vector.
    - Can be expensive to calculate for large datasets
    - It may converge to a local maxima, or oscillate around a local maxima.
    - In case of oscillations, tuning the stepsize t to t_k is useful.
    - Examples - logistic (with careful tuning), Location Cauchy example (with good starting values)
- Non-stochastic non-gradient-based algorithms
  - MM algorithm
    - Useful when objective function can easily be lower bounded by a concave

function locally.

    * Guaranteed to converge to a local maxima

    * Examples - Bridge regression

  - EM algorithm

    * A specific case of the EM algorithm

    * Particularly useful for censored/missing data problems.

    * Immensely useful in clustering particularly using Gaussian mixture models.

- Stochastic optimization methods

  - Stochastic gradient ascent

    * Useful when original gradient ascent algorithm is expensive to implement

    * or when data is coming in sequentially (like in an online format)

    * Typically a good idea to use mini-batch stochastic gradient for stable answers.

    * Convergence properties are the same as gradient ascent

  - Simulated annealing

    * No gradient are required.

    * Most useful for non-concave target distributions since it can escape out of local maximas.

The above methods provide estimators based on certain fix model tuning parameters. This includes penalty term λ, bridge regression penalty term α, and the number of

clusters/classes in Gaussian mixture models, C.

In order to choose the these tuning parameters, we use model selection techniques:

- Cross-validation
  - Leave-one-out Cross-validation: expensive, but accurate
  - K-fold cross-validation: cheaper, but not as accurate.
  - effective in regression models
- Akaike Information Criterion (AIC) / Bayesian information criterion (BIC)
  - AIC/BIC is most effective for finding C in Gaussian mixture models.
  - Can also be used in regression problems
  - BIC is more robust than AIC

Note: None of these techniques can be used as black-box. For every implementation you must decide looking at the final prediction error/AIC/BIC values, which value of the tuning parameter may be more reasonable.

Once final estimates using model selection and optimization tools have been obtained, interest is in constructing confidence intervals around parameters of interest. For this we can use two Boostrapping methods:

- Nonparametric Bootstrap
  - Cheaper to implement and can always be implemented
  - Produces higher variance in resulting confidence intervals for low sample size settings
- Parametric Bootstrap
  - Can't always be implemented

– When it can, it typically works better than nonparametric methods

– It is more expensive to implement.

MTH 511a - 2021: L33-34: Bayesian Models

Instructor: Dootika Vats

1 Bayesian models

So far, in data analysis techniques, we have assumed that the data comes from some distribution. This distribution dictates the likelihood, and then having observed the data, the likelihood is maximized.

However, sometimes, we may already have some information about a parameter in the data distribution. We would like to take into account that information. Some examples:

- The mean of the batting average of cricket openers in ODIs. We know that opening batters are likely to have batting averages between 30 - 50.
- The mean GRE quantitative score of MTH students. It is likely that MTH students will have high GRE quant scores, so we should be able to account for this information.
- You want to estimate how much money Akshay Kumar's next movie will make. His latest few movies have been big successes and big failures and everything

in between. You have information that his movie can be highly variable in its revenue, and want to take account for that information.

1.1 Prior distribution

Suppose $X_1, \ldots, X_n \sim F_\theta$ is data from a distribution where θ is a defining parameter. So far in likelihood-based estimation techniques, we've obtained

$$L(\theta|X_1, \ldots, X_n) = \prod_{i=1}^{n} f(X_i \mid \theta)\,,$$

and then we maximized the likelihood. Thus, this likelihood becomes the key quantity of interest.

When we have some information on the parameter θ, we want to be able to include that information into our process. We will assume that θ, the parameter of interest, is random, in that, it has a distribution. The distribution assigned to θ is called the *prior distribution* and is meant to encapsulate the *prior* information about the parameter:

$$\theta \sim \pi(\theta)\,,$$

where $\pi(\theta)$ denotes the density of the prior distribution. For example:

- **Cricket:** $\theta \sim N(40, 20)$
- **GRE Quant:** $\theta \sim N(167, 1)$
- **Akshay Kumar:** $\theta \sim$ Gamma(40 lacs, 1).

1.2 Posterior distribution

Having observed the data $X_1, \ldots, X_n \sim F_\theta$, we now have more evidence that helps us understand θ. That is, we can update the prior information using the observed data. As in, we are interested in the distribution of θ *given* $X_1, \ldots, X_n$.

$$\pi(\theta|X_1, \ldots, X_n)\,.$$

This is called the *posterior distribution*. Since we know $\pi(\theta)$ and we know $f(X_1, \ldots, X_n|\theta)$, we can use Bayes' Theorem to the density of the posterior distribution:

$$\begin{aligned}
\pi(\theta \mid X_1, \ldots, X_n) &= \frac{f(\theta, X_1, \ldots, X_n)}{\int f(\theta, X_1, \ldots, X_n) d\theta} \\
&= \frac{f(X_1, \ldots, X_n \mid \theta)\, \pi(\theta)}{\int f(\theta, X_1, \ldots, X_n) d\theta} \\
&\propto f(X_1, \ldots, X_n \mid \theta)\, \pi(\theta)\,,
\end{aligned}$$

where note that the proportionality constant is the inverse of the marginal distribution of the data and is uniquely defined since the right hand side must integrate to 1.

The posterior distribution contains all relevant information about θ having accounted for the observed data. Note that, a key distinguishing feature of Bayesian models, from non-Bayesian (or *frequentist* models) is that the parameter θ is now random and has a distribution. This is a philosophical difference between the two ideologies. We will look at a few examples to understand this better.

Example 1 (Coin probability). Suppose $Y_1, \ldots, Y_n \sim \text{Bern}(p)$ are realizations from n series of coin flips of a coin that gives heads with probability p. Suppose you have no reason to believe that the coin is fair, and have no idea what is the probability of heads for the coin. You want to let the model know that p could be anything between $[0, 1]$ and in fact all values seem equally likely to you.

That is

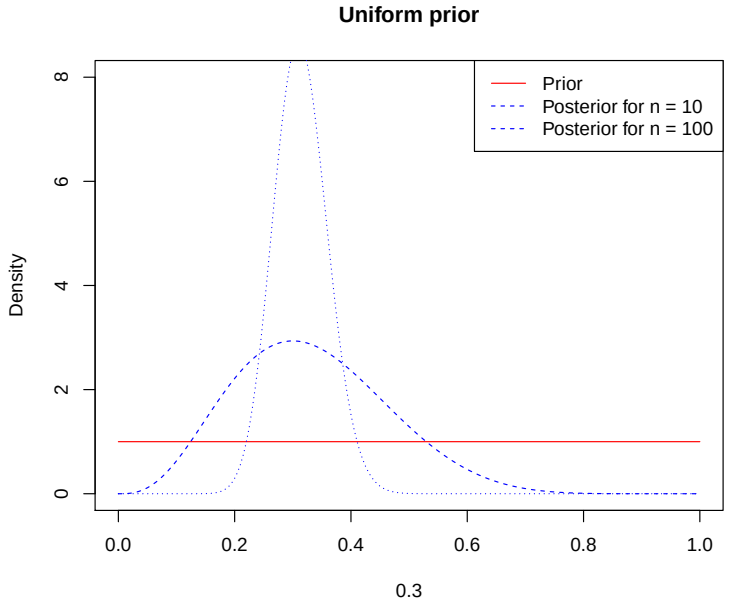
$$\textit{Prior distribution: } \theta \sim U[0,1]\,.$$

That is, the prior on p is $\pi(p) = 1$ for $0 \leq p \leq 1$. By Bayes' theorem, the posterior distribution is:

$$\begin{aligned}
\pi(p \mid y) &\propto \pi(p) \cdot \prod_{i=1}^{n} f(y_i \mid p) \\
&= 1\mathbb{I}(0 \leq p \leq 1) \cdot \prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i} \\
&= p^{\sum_{i=1}^{n} y_i}(1-p)^{n-\sum_{i=1}^{n} y_i}\, \mathbb{I}(0 \leq p \leq 1)\,,
\end{aligned}$$

Although we haven't found the proportionality constant (the marginal likelihood), we know that the above expression is that of a Beta distribution. Thus,

$$\textit{Posterior distribution: } p \mid y \sim \text{Beta}\left(\sum_{i=1}^{n} y_i + 1, n - \sum_{i=1}^{n} y_i + 1\right)$$



Now, suppose our prior distribution was different, and that we knew some values of p were more favorable than others. We can assume

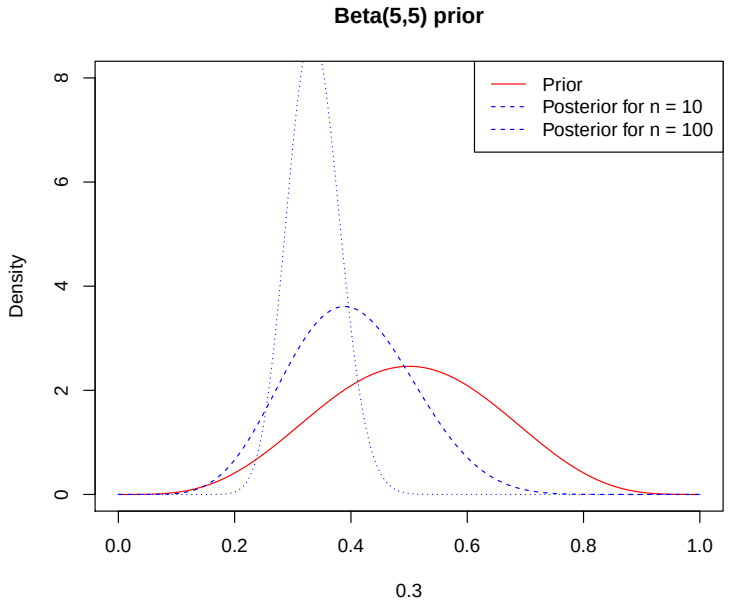
$$\textit{Prior distribution: } \theta \sim \text{Beta}(\alpha, \beta)\,,$$

for $\alpha, \beta > 0$. Note that for $\alpha = \beta = 1$, the prior is the $U[0,1]$ prior. For this prior, the posterior will be different, since

$$\begin{aligned}
\pi(p \mid y) &\propto \pi(p) \cdot \prod_{i=1}^{n} f(y_i \mid p) \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1}\mathbb{I}(0 \le p \le 1) \cdot \prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i} \\
&\propto p^{\sum_{i=1}^{n} y_i + \alpha - 1}(1-p)^{n - \sum_{i=1}^{n} y_i + \beta - 1}\,\mathbb{I}(0 \le p \le 1)\,.
\end{aligned}$$

Thus, the posterior distribution is

$$\textit{Posterior distribution: } p \mid y \sim \text{Beta}\left(\sum_{i=1}^{n} y_i + \alpha, n - \sum_{i=1}^{n} y_i + \beta\right)\,.$$



Question: Information about θ is in the form of a distribution. How do we summarize a full distribution:

1. *Maximum a posterior (MAP)*: the mode of the posterior distribution.
2. *Posterior mean*: the mean of the posterior distribution, $\mathrm{E}[\theta|y]$.

Amongst the two, posterior means are used more often that MAP. The above are point estimates and do not give any idea of the variability of the posterior distribution. Thus, we need a way to assess the variability. This is done via *credible intervals*. A 95% credible interval is $[L, U]$ such that

$$\int_L^U \pi(\theta|y)d\theta = .95\,.$$

Thus, one way to obtain $[L, U]$ is to set $L = .025^{th}$-quantile and $U = .975^{th}$-quantile. Similarly, an 80% quantile will be $L = .10^{th}$-quantile and $U = .90^{th}$-quantile.

Example 2 (Coin probability continued). The MAP estimator requires solving the following optimization problem:

$$\arg\max_p \left\{ p^{\sum_{i=1}^n y_i + \alpha - 1}(1-p)^{n - \sum_{i=1}^n y_i + \beta - 1}\, \mathbb{I}(0 \le p \le 1) \right\},$$

which, if it exists, is easy to show, that it happens at

$$\hat{p}_{\mathrm{MAP}} = \frac{\sum_{i=1}^n y_i + \alpha - 1}{n + \alpha + \beta - 2}$$

The posterior mean estimator is

$$\hat{p}_{\mathrm{PM}} = \frac{\sum_{i=1}^n y_i + \alpha}{n + \alpha + \beta}\,.$$

Notice also that

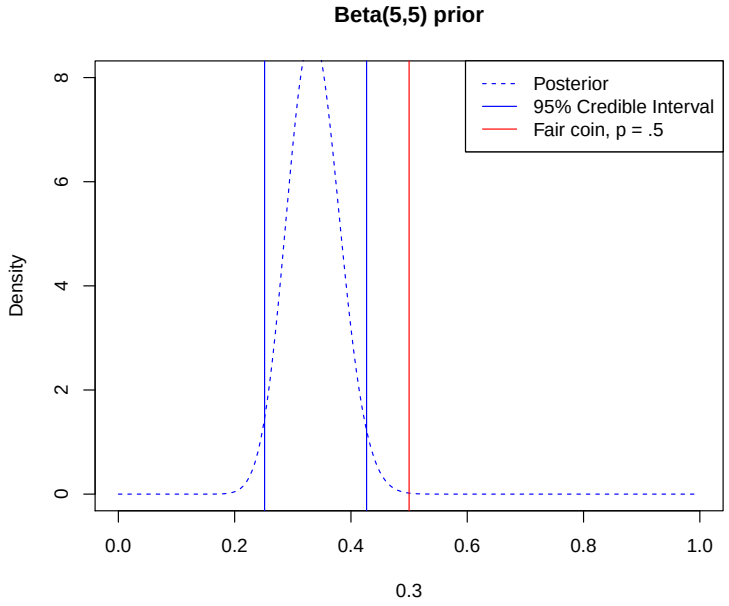
$$\begin{aligned}\hat{p}_{\text{PM}} &= \frac{\sum_{i=1}^{n} y_i}{n+\alpha+\beta} + \frac{\alpha}{n+\alpha+\beta} \\ &= \frac{n}{n+\alpha+\beta}\bar{y} + \frac{\alpha+\beta}{n+\alpha+\beta} \cdot \frac{\alpha}{\alpha+\beta}\,.\end{aligned}$$

Notice that the posterior mean is a weighted average between the prior mean and the data mean. As data increases ($n \to \infty$), $n/(n+\alpha+\beta) \to 1$ and thus,

$$\hat{p}_{PM} - \bar{y} \to 0\,.$$

That is, Bayesian modeling, in many cases (not in all!), finds a balance between information from the data and information from the prior, and once there is sufficient data, it essentially starts matching the inference made by only the data.

We can also visualize the 95% credible interval and a vertical line for the previous simulated data and we also present a vertical line for $p = .5$. Since this vertical line is outside the credible interval, we can be 95% certain that the coin is not fair!



Example 3 (Normal-normal example). Let $Y_1, Y_2, \ldots, Y_n \sim N(\theta, \sigma^2)$ for $\sigma^2 > 0$ known. Further, let's suppose the prior distribution is

$$\textit{Prior distribution: } \theta \sim N(m_0, s_0^2)\,.$$

The posterior distribution is

$$\textit{Posterior distribution: } \theta \sim \pi(\theta|y)\,.$$

In order to find this

$$\begin{aligned}
&\pi(\theta|y) \\
&\propto \pi(\theta) \cdot \prod_{i=1}^{n} f(y_i|\theta) \\
&\propto \exp\left\{-\frac{(\theta - m_0)^2}{2s_0^2}\right\} \exp\left\{-\sum_{i=1}^{n} \frac{(y_i - \theta)^2}{2\sigma^2}\right\} \\
&= \exp\left\{-\frac{1}{2}\left(\frac{\theta^2}{s_0^2} + \frac{m_0^2}{s_0^2} - \frac{2\theta m_0}{s_0^2} + \frac{\sum_{i=1}^n y_i^2}{\sigma^2} + \frac{n\theta^2}{\sigma^2} - \frac{2\theta \sum_{i=1}^n y_i}{\sigma^2}\right)\right\} \\
&\propto \exp\left\{-\frac{1}{2}\left(\theta^2\left(\frac{1}{s_0^2} + \frac{n}{\sigma^2}\right) - 2\theta\left(\frac{m_0}{s_0^2} + \frac{\sum_{i=1}^n y_i}{\sigma^2}\right)\right)\right\} \\
&\propto \exp\left\{-\frac{1}{2}\left(\frac{1}{s_0^2} + \frac{n}{\sigma^2}\right)\left(\theta^2 - 2\theta\left(\frac{m_0}{s_0^2} + \frac{n\bar{y}}{\sigma^2}\right)\left(\frac{1}{s_0^2} + \frac{n}{\sigma^2}\right)^{-1} + \left[\left(\frac{m_0}{s_0^2} + \frac{n\bar{y}}{\sigma^2}\right)\left(\frac{1}{s_0^2} + \frac{n}{\sigma^2}\right)^{-1}\right]^2\right)\right\}
\end{aligned}$$

So that

$$\theta|y \sim N\left(\left(\frac{m_0}{s_0^2} + \frac{n\bar{y}}{\sigma^2}\right)\left(\frac{1}{s_0^2} + \frac{n}{\sigma^2}\right)^{-1}, \left(\frac{1}{s_0^2} + \frac{n}{\sigma^2}\right)^{-1}\right)$$

How to choose a prior distribution:

- The support of the prior distribution defines the support of the posterior distribution. Thus, one way to restrict the family of distributions is to consider the restrictions on the parameter. For example, for the Bernoulli problem, the

prior on p must be between 0,1. If putting a prior on the variance of a normal distribution, the prior distribution must have positive support.

- Within a family of prior distributions, the specific choices depend on prior information available, like in the examples above.
- Sometimes (and this should not happen often), priors may be chosen so as to allow for simpler and computable posterior distributions. We will discuss these next time.

2 Questions to think about

- Show for yourself that the posterior mean in the normal example is a weighted average of the sample mean and the prior mean. What happens at $n \to \infty$?
- Find the posterior distribution for data distribution $\text{Exp}(\lambda)$ and prior $\lambda \sim \text{Gamma}(a, b)$.
- Generate data from Example 3 and pictorial see how the posterior distribution changes as n increases.
- When will MAP and posterior means coincide?

MTH 511a - 2021: L37 - Markov chain Monte Carlo

Instructor: Dootika Vats

It should be evident that accept-reject sampling is very difficult to implement in most practical Bayesian models. This makes it really difficult to estimate quantities of the posterior distribution without being able to get samples from it! It is here where Markov chain Monte Carlo (MCMC) will be credibly useful and powerful.

1 Markov chain Monte Carlo

We want to sample from a distribution $\pi(x)$ which we may or may not know completely. If iid samples (like using accept-reject) are not available or not possible, we often use Markov chain Monte Carlo (MCMC).

Let $\pi(x)$ be the target distribution defined on a space $\mathcal{X}$. We will choose a starting value $X_1 \sim \pi(x)$.

In MCMC, we sample a *Markov chain* $X_1, X_2, \ldots$ with $\pi(x)$ as the *stationary distribution*. The Markov chain is determined by a *Markov transition kernel*, where for a set

$A \subseteq \mathcal{X}$, $P(x_1, \cdot)$ describes how the next step in the Markov chain will be obtained:

$$P(x_1, A) := \Pr(X_2 \in A \mid X_1 = x_1)\,.$$

The kernel P satisfies π-stationarity; for a set $A \subseteq \mathcal{X}$

$$\int_{\mathcal{X}} P(x_1, A)\pi(x_1)dx_1 = \pi(A)\,.$$

That is, starting from a random draw from π and then moving according to $P(x_1, \cdot)$ is the same as drawing from π. In other words, if $X_1 \sim \pi$, then $X_2 \sim \pi$. In practice, it is difficult to sample $X_1 \sim \pi$, so a *reasonable starting value* is chosen.

In this way, we keep generating more and more samples. The sample obtained: $X_1, X_2, \ldots, X_T$ satisfies the following

- The samples are *all correlated*. So every new X carries far less information than an iid X
- Since we don't start from π, we actually do not sample exactly from π. In fact, the distribution of X_t converges to π. That is $X_\infty \sim \pi$. But usually, if X_1 is chosen well, we are very close to π reasonably quickly.

Since, $X_\infty \sim \pi$, a Markov chain law of large numbers holds, for any function g,

$$\frac{1}{T}\sum_{t=1}^{T} g(X_T) \xrightarrow{p} \mathrm{E}_\pi[g(X)] \quad \text{as } n \to \infty\,.$$

So even though, we have far from iid samples, we will still converge to the right quantity!! This means that we can try to use Markov chain Monte Carlo to generate samples from our posterior distributions.

Question is, how can we construct such a Markov chain?

MTH 511a - 2021: L35: Bayesian Models and Accept-Reject

Instructor: Dootika Vats

1 Bayesian models

Question: Information about θ is in the form of a distribution. How do we summarize a full distribution:

1. *Maximum a posterior (MAP)*: the mode of the posterior distribution.
2. *Posterior mean*: the mean of the posterior distribution, $\mathrm{E}[\theta|y]$.

Amongst the two, posterior means are used more often that MAP. The above are point estimates and do not give any idea of the variability of the posterior distribution. Thus, we need a way to assess the variability. This is done via *credible intervals*. A 95% credible interval is $[L, U]$ such that

$$\int_L^U \pi(\theta|y)d\theta = .95\,.$$

Thus, one way to obtain $[L, U]$ is to set $L = .025^{th}$-quantile and $U = .975^{th}$-quantile.

Similarly, an 80% quantile will be $L = .10^{th}$-quantile and $U = .90^{th}$-quantile.

Example 1 (Coin probability continued). The MAP estimator requires solving the following optimization problem:

$$\arg\max_p \left\{ p^{\sum_{i=1}^n y_i + \alpha - 1} (1-p)^{n - \sum_{i=1}^n y_i + \beta - 1} \, \mathbb{I}(0 \leq p \leq 1) \right\} ,$$

which, if it exists, is easy to show, that it happens at

$$\hat{p}_{\text{MAP}} = \frac{\sum_{i=1}^n y_i + \alpha - 1}{n + \alpha + \beta - 2}$$

The posterior mean estimator is

$$\hat{p}_{\text{PM}} = \frac{\sum_{i=1}^n y_i + \alpha}{n + \alpha + \beta} .$$

Notice also that

$$\begin{aligned} \hat{p}_{\text{PM}} &= \frac{\sum_{i=1}^n y_i}{n + \alpha + \beta} + \frac{\alpha}{n + \alpha + \beta} \\ &= \frac{n}{n + \alpha + \beta} \bar{y} + \frac{\alpha + \beta}{n + \alpha + \beta} \cdot \frac{\alpha}{\alpha + \beta} . \end{aligned}$$

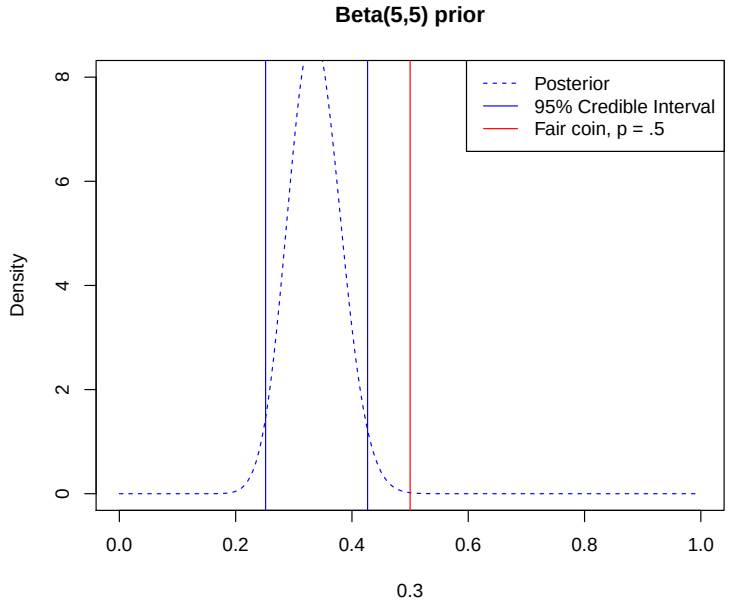
Notice that the posterior mean is a weighted average between the prior mean and the data mean. As data increases ($n \to \infty$), $n/(n + \alpha + \beta) \to 1$ and thus,

$$\hat{p}_{PM} - \bar{y} \to 0 .$$

That is, Bayesian modeling, in many cases (not in all!), finds a balance between information from the data and information from the prior, and once there is sufficient data, it essentially starts matching the inference made by only the data.

We can also visualize the 95% credible interval and a vertical line for the previous

simulated data and we also present a vertical line for $p = .5$. Since this vertical line is outside the credible interval, we can be 95% certain that the coin is not fair!



Example 2 (Normal-normal example). Let $Y_1, Y_2, \ldots, Y_n \sim N(\theta, \sigma^2)$ for $\sigma^2 > 0$ known. Further, let's suppose the prior distribution is

$$\textit{Prior distribution: } \theta \sim N(m_0, s_0^2)\,.$$

The posterior distribution is

$$\textit{Posterior distribution: } \theta \sim \pi(\theta|y)\,.$$

In order to find this

$$\begin{aligned}&\pi(\theta|y)\\ &\propto \pi(\theta) \cdot \prod_{i=1}^{n} f(y_i|\theta)\end{aligned}$$

$$
\begin{aligned}
&\propto \exp\left\{-\frac{(\theta-m_0)^2}{2s_0^2}\right\}\exp\left\{-\sum_{i=1}^{n}\frac{(y_i-\theta)^2}{2\sigma^2}\right\}\\
&= \exp\left\{-\frac{1}{2}\left(\frac{\theta^2}{s_0^2}+\frac{m_0^2}{s_0^2}-\frac{2\theta m_0}{s_0^2}+\frac{\sum_{i=1}^n y_i^2}{\sigma^2}+\frac{n\theta^2}{\sigma^2}-\frac{2\theta\sum_{i=1}^n y_i}{\sigma^2}\right)\right\}\\
&\propto \exp\left\{-\frac{1}{2}\left(\theta^2\left(\frac{1}{s_0^2}+\frac{n}{\sigma^2}\right)-2\theta\left(\frac{m_0}{s_0^2}+\frac{\sum_{i=1}^n y_i}{\sigma^2}\right)\right)\right\}\\
&\propto \exp\left\{-\frac{1}{2}\left(\frac{1}{s_0^2}+\frac{n}{\sigma^2}\right)\left(\theta^2-2\theta\left(\frac{m_0}{s_0^2}+\frac{n\bar{y}}{\sigma^2}\right)\left(\frac{1}{s_0^2}+\frac{n}{\sigma^2}\right)^{-1}+\left[\left(\frac{m_0}{s_0^2}+\frac{n\bar{y}}{\sigma^2}\right)\left(\frac{1}{s_0^2}+\frac{n}{\sigma^2}\right)^{-1}\right]^2\right)\right\}
\end{aligned}
$$

So that

$$
\theta|y \sim N\left(\left(\frac{m_0}{s_0^2}+\frac{n\bar{y}}{\sigma^2}\right)\left(\frac{1}{s_0^2}+\frac{n}{\sigma^2}\right)^{-1},\left(\frac{1}{s_0^2}+\frac{n}{\sigma^2}\right)^{-1}\right)
$$

How to choose a prior distribution:

- The support of the prior distribution defines the support of the posterior distribution. Thus, one way to restrict the family of distributions is to consider the restrictions on the parameter. For example, for the Bernoulli problem, the prior on p must be between 0,1. If putting a prior on the variance of a normal distribution, the prior distribution must have positive support.
- Within a family of prior distributions, the specific choices depend on prior information available, like in the examples above.
- Sometimes (and this should not happen often), priors may be chosen so as to allow for simpler and computable posterior distributions. We will discuss these next time.

2 Complicated Bayesian models

So far, we introduced the philosophy of Bayesian modeling and worked through two popular examples. In both examples, the posterior distributions were available in

closed-form after some algebra tricks.

However, in many situations, the posterior distribution is not easily obtained. Let's first see an example.

Example 3 (t-distribution likelihood). Suppose $Y_1, \ldots, Y_n|\mu \sim t_\nu(\mu)$, which is the t distribution with ν degrees of freedom and mean μ. Let's assume that ν is known.

Consider the prior $\mu \sim N(0, 1)$ on μ. The posterior distribution of μ is,

$$\begin{aligned}
\pi(\mu \mid y_1, \ldots, y_n) &\propto \pi(\mu) f(y|\mu) \\
&= e^{-\frac{\mu^2}{2}} \prod_{i=1}^{n} \left(\frac{\Gamma(\nu + 1/2)}{\sqrt{\nu\pi}\Gamma(\nu/2)} \left(1 + \frac{(y_i - \mu)^2}{\nu}\right)^{-\frac{\nu+1}{2}} \right) \\
&\propto e^{-\frac{\mu^2}{2}} \prod_{i=1}^{n} \left(1 + \frac{(y_i - \mu)^2}{\nu}\right)^{-\frac{\nu+1}{2}} .
\end{aligned}$$

This does not look like the functional form of any known density. We may be able to find the MAP estimate using the many optimization techniques that we've learned, however, if we want to find the quantiles and mean of this distribution, it will be further challenging.

The solution to this problem is sampling and Monte Carlo!

Instead of finding the closed form expression of the posterior distribution, we will obtain samples from this posterior distribution and *estimate* posterior quantities using Monte Carlo based estimators.

General sampling framework

We have a target density $\pi(\theta|y)$ whose expectation and quantiles are of interest. Notice that in the above example, we have a target density whose proportionality constant

we do not know. We are now in the situation where

$$\pi(\theta|y) = c\,\tilde{\pi}(\theta|y)\,,$$

where c is *unknown*. We have seen this before in importance sampling. Now we see when such instances occur!

We will use two methods to estimate quantities of interest.

- Accept-reject sampling
- Markov chain Monte Carlo (MCMC)

In both methods, we will sample values $X_1, \ldots, X_n$ from $\pi(\theta|y)$ and using sample mean and sample quantiles, to estimate the posterior mean and credible intervals.

You are already familiar with accept-reject sampling, however, you only known how to implement it when c is known. Now we will learn how to implement accept-reject when this constant is unknown.

2.1 Accept-Reject Sampling

Recall that, to implement accept-reject on a given target density, we need to find a proposal distribution G with density g such that

$$\sup_{x\in\mathcal{X}} \frac{\pi(x)}{g(x)} \leq M < \infty\,.$$

Since the upper bound, M, is subsequently used in the algorithm, M needs to be known constant. However, when

$$\pi(x) = c\ \tilde{\pi}(x)\,,$$

M cannot be known, since c is unknown. Turns out, accept-reject can be used to sample from an unnormalized distribution as well, however, we lose out on some benefits.

Suppose $g(x)$ is a proposal density such that $g(x) = r\tilde{g}(x)$, where r is a known or unknown constant. Suppose there exists $\tilde{M}$ such that

$$\sup_{x \in \mathcal{X}} \frac{\tilde{\pi}(x)}{\tilde{g}(x)} \leq \tilde{M} \,.$$

Notice that,

$$\begin{aligned} &\sup_{x \in \mathcal{X}} \frac{\tilde{\pi}(x)}{\tilde{g}(x)} \leq \tilde{M} \\ \Leftrightarrow &\sup_{x \in \mathcal{X}} \frac{m\tilde{\pi}(x)}{r\tilde{g}(x)} \leq \frac{m}{r}\tilde{M} \\ \Leftrightarrow &\sup_{x \in \mathcal{X}} \frac{\pi(x)}{g(x)} \leq \frac{m}{r}\tilde{M} =: M \end{aligned}$$

Thus, upper bounding $\tilde{\pi}/\tilde{g}$ by $\tilde{M}$ is the same as upper bounding π/g by the *unknown* M. That is, for $U \sim U[0,1]$,

$$U \leq \frac{\pi(x)}{Mg(x)} \Leftrightarrow U \leq \frac{\tilde{\pi}(x)}{\tilde{M}\tilde{g}(x)} \,.$$

So the accept-reject step in AR can be replaced. Essentially, we don't need to know the normalizing constants for both π and g.

Algorithm 1 Accept-Reject for unknown constants

1: Generate $Y \sim G$ and generate $U \sim U[0,1]$
2: If $U \leq \dfrac{\tilde{\pi}(y)}{\tilde{M}\tilde{g}(y)}$, then set $X = Y$
3: Else, go to step 1.

However, a significant drawback here is that, the probability of acceptance of any

proposal is still

$$\Pr\left(\text{Acceptance}\right) = \frac{1}{M}$$

and since M is unknown, this probability is unknown. Thus, there isn't any way to assess how efficient the algorithm will be.

Example 4 (Bayesian $t-$likelihood). Recall the posterior distribution

$$\pi(\mu|y) \propto e^{-\mu^2/2} \prod_{t=1}^{n} \left(1 + \frac{(y_i - \mu)^2}{\nu}\right)^{-(\nu+1)/2}$$

Consider the proposal distribution $N(0,1)$.

$$\begin{aligned}
\frac{\tilde{\pi}(\mu)}{\tilde{g}(\mu)} &= \frac{e^{-\mu^2/2} \prod_{t=1}^{n} \left(1 + \frac{(y_i - \mu)^2}{\nu}\right)^{-(\nu+1)/2}}{e^{-\mu^2/2}} \\
&= \prod_{t=1}^{n} \left(1 + \frac{(y_i - \mu)^2}{\nu}\right)^{-(\nu+1)/2} \\
&\leq 1 := \tilde{M}_1 \,.
\end{aligned}$$

This is not a tight bound, but an easily available bound. Nonetheless, we can implement an A-R algorithm, but it is not very efficient.

An alternative is to use $\hat{\mu}_{\text{MLE}}$ as an upper bound. This we known how to do using optimization techniques a similar Cauchy example.

$$\begin{aligned}
\frac{\tilde{\pi}(x)}{\tilde{g}(x)} &= \frac{e^{-\mu^2/2} \prod_{t=1}^{n} \left(1 + \frac{(y_i - \mu)^2}{\nu}\right)^{-(\nu+1)/2}}{e^{-\mu^2/2}} \\
&= \prod_{t=1}^{n} \left(1 + \frac{(y_i - \mu)^2}{\nu}\right)^{-(\nu+1)/2} \\
&\leq \prod_{t=1}^{n} \left(1 + \frac{(y_i - \hat{\mu}_{\text{MLE}})^2}{\nu}\right)^{-(\nu+1)/2} := \tilde{M}_2 \,.
\end{aligned}$$

This will be more efficient, but $\hat{\mu}_{\text{MLE}}$ needs to be found numerically. Notice that the bound gets worse when

- the number of data points, n increases
- if the true value of μ is far from 0.

```
##########################################
## Accept-reject to sample from posterior distrbituon
# from t likelihood and normal priors
# Code takes some time to run
##########################################
library(MASS)
set.seed(10)


#Log posterior
log.post <- function(y, mu, nu)
{
  -mu^2/2 - (nu + 1)/2 *sum( log( 1 + (y-mu)^2/nu ) )
}


# Accept-reject for a given bound M
AR_tmodel <- function(N = 1e2, M = 1)
{
  count <- 0
  attempts <- 0
  samp <- numeric(length = N)
  while(count < N)
  {
    attempts <- attempts + 1
    prop <- rnorm(1)
```

```
    ratio <- log.post(y = y, mu = prop, nu = nu) + prop^2/2 - log(M)
    if(exp(ratio) > 1) print(exp(ratio)) # Making sure M is correct
    if(runif(1) < exp(ratio))
    {
      count <- count + 1
      samp[count] <- prop
      if(count%% (N) == 0) print(paste("Accepted = ",count, ", Accept Prob.
          = ", count/attempts))
    }
  }
  return(samp)
}
```

We will now generate data from a t_3 distribution with the true $\mu = 0$ and generate 10000 iid samples from the posterior using the AR with $\tilde{M}_1$ and $\tilde{M}_2$.

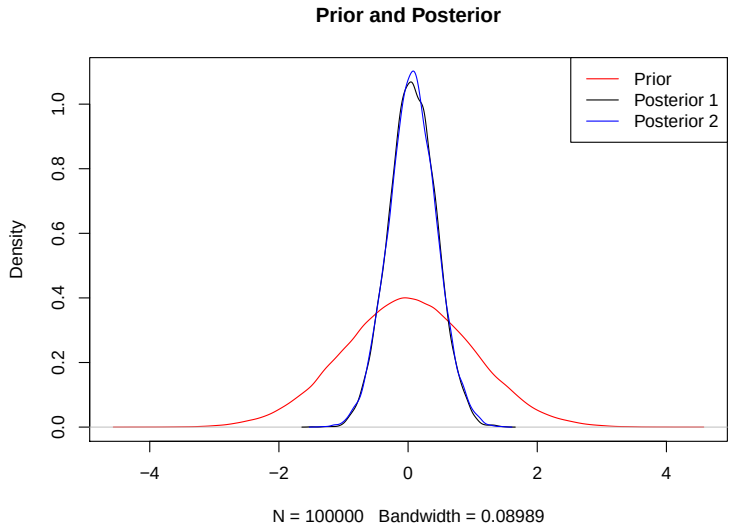
```
# Generate the data set.
# Small n and mu close to zero (the prior)
nu <- 3
n <- 10
mu <- 0
y <- mu + rt(n, df = nu) # Generate the data

mle <- fitdistr(y, "t", df = 3)$estimate[1] # Find the MLE to construct the
    tighter upper bound for pi(x)/g(x)
M2 <- prod( (1 + (y - mle)^2/nu ) )^(-(nu+1)/2) + 1e-5 # adding a little to
    remove numerical approximation errors
```

```
# Run the A-R sampler using the two different upper bounds
system.time(out1 <- AR_tmodel(N = 1e4, M = 1) ) # About 50 seconds
#[1] "Accepted = 10000 , Accept Prob. = 0.00146156500580753"
system.time(out2 <- AR_tmodel(N = 1e4, M = M2) ) # About .11 seconds
#[1] "Accepted = 10000 , Accept Prob. = 0.37267543696195"
```

Although the AR algorithms are different, both methods with M_1 and M_2 will yield iid samples from the same posterior. We can check the posterior means and quantiles (for credible intervals) and draw the posterior density estimate plot:

```
# Posterior mean estimates
c(mean(out1), mean(out2))
#[1] 0.06775302 0.06930398


# 95% credible interval
quantile(out1, c(.025, .975))
#      2.5%     97.5%
#-0.6487182 0.7868883

quantile(out2, c(.025, .975))
#      2.5%     97.5%
#-0.6534764 0.8166141



# Compare the performance
plot(density(rnorm(1e5)), col = "red", type = 'l', ylim = c(0,1.1), main =
    "Prior and Posterior")#prior
lines(density(out1)) # samples posterior 2
lines(density(out2), col = "blue") # sampled posterior 2
```

```
legend("topright", legend = c("Prior", "Posterior 1", "Posterior 2"), col =
    c("red", "black", "blue"), lty = 1)
```



We repeat the same but now I increase my data size from $n = 10$ to $n = 20$. This small change will dramatically decrease the acceptance probability for both M_1 and M_2.

```
# Increase n
n <- 20
y <- mu + rt(n, df = nu) # Generate the data

mle <- fitdistr(y, "t", df = 3)$estimate[1] # Find the MLE to construct the
    tighter upper bound for pi(x)/g(x)
M2 <- prod( (1 + (y - mle)^2/nu ) )^(-(nu+1)/2) + 1e-6



# Run the A-R sampler using the two different upper bounds
system.time(out1 <- AR_tmodel(N = 1e1, M = 1) ) # too slow for even 10
    samples
```

```
# [1] "Accepted = 10 , Accept Prob. = 2.39168192161119e-07"
system.time(out2 <- AR_tmodel(N = 1e4, M = M2) ) # About .11 seconds
#[1] "Accepted = 10000 , Accept Prob. = 0.128182121157741"
```

3 Questions to think about

- Show for yourself that the posterior mean in the normal example is a weighted average of the sample mean and the prior mean. What happens at $n \to \infty$?
- Find the posterior distribution for data distribution $\text{Exp}(\lambda)$ and prior $\lambda \sim \text{Gamma}(a, b)$.
- Generate data from Example 3 and pictorial see how the posterior distribution changes as n increases.
- When will MAP and posterior means coincide?
- Repeat the last simulation with the true data generated from $\mu = 5$. How does the AR change? How does the posterior change?
- How do you think AR will fare for higher dimensional problems?

MTH 511a - 2021: L38 - Markov chain Monte Carlo Continued

Instructor: Dootika Vats

It should be evident that accept-reject sampling is very difficult to implement in most practical Bayesian models. This makes it really difficult to estimate quantities of the posterior distribution without being able to get samples from it! It is here where Markov chain Monte Carlo (MCMC) will be credibly useful and powerful.

1 Markov chain Monte Carlo

We want to sample from a distribution $\pi(x)$ which we may or may not know completely. If iid samples (like using accept-reject) are not available or not possible, we often use Markov chain Monte Carlo (MCMC).

Let $\pi(x)$ be the target distribution defined on a space $\mathcal{X}$. We will choose a starting value $X_1 \sim \pi(x)$.

In MCMC, we sample a *Markov chain* $X_1, X_2, \ldots$ with $\pi(x)$ as the *stationary distribution*. The Markov chain is determined by a *Markov transition kernel*, where for a set

$A \subseteq \mathcal{X}$, $P(x_1, \cdot)$ describes how the next step in the Markov chain will be obtained:

$$P(x_1, A) := \Pr(X_2 \in A \mid X_1 = x_1)\,.$$

The kernel P satisfies π-stationarity; for a set $A \subseteq \mathcal{X}$

$$\int_{\mathcal{X}} P(x_1, A)\pi(x_1)dx_1 = \pi(A)\,.$$

That is, starting from a random draw from π and then moving according to $P(x_1, \cdot)$ is the same as drawing from π. In other words, if $X_1 \sim \pi$, then $X_2 \sim \pi$. In practice, it is difficult to sample $X_1 \sim \pi$, so a *reasonable starting value* is chosen.

In this way, we keep generating more and more samples. The sample obtained: $X_1, X_2, \ldots, X_T$ satisfies the following

- The samples are *all correlated*. So every new X carries far less information than an iid X
- Since we don't start from π, we actually do not sample exactly from π. In fact, the distribution of X_t converges to π. That is $X_\infty \sim \pi$. But usually, if X_1 is chosen well, we are very close to π reasonably quickly.

Since, $X_\infty \sim \pi$, a Markov chain law of large numbers holds, for any function g,

$$\frac{1}{T}\sum_{t=1}^{T} g(X_T) \xrightarrow{p} \mathrm{E}_\pi[g(X)] \quad \text{as } n \to \infty\,.$$

So even though, we have far from iid samples, we will still converge to the right quantity!! This means that we can try to use Markov chain Monte Carlo to generate samples from our posterior distributions.

Question is, how can we construct such a Markov chain?

2 Metropolis-Hastings

The Metropolis-Hastings algorithm is the most common one to construct such a Markov chain. And is similar to accept-reject in principle, although very different. The Metropolis-Hastings algorithm functions as following.

To construct a Markov chain with target distribution π and given that the current sample drawn is x, choose a proposal distribution Q_x with density $q(y|x)$.

1. Choose a starting value x_1 that is in $\mathcal{X}$. For any iterate t do the following
2. Draw a proposed value $y^* \sim q(y^*|x_t)$.
3. Calculate
$$\alpha(x_t, y^*) = \min\left\{1, \frac{\pi(y^*)\, q(x_t|y^*)}{\pi(x_t)\, q(y^*|x_t)}\right\}$$
4. Draw $U \sim U[0,1]$. If $U \leq \alpha(x_t, y^*)$. Set $x_{t+1} = y^*$.
5. Else, set $x_{t+1} = x_t$.

Remember that if we reject a value, then our next step is our previous step. The proposal distribution is usually taken so that it is symmetric in both x and y. That is, $q(y|x) = q(x|y)$. This means that the MH ratio is

$$\alpha(x_t, y^*) = \min\left\{1, \frac{\pi(y^*)}{\pi(x_t)}\right\}$$

Example 1 (Sampling from a circle). Recall our example of sampling from a circle and the problem set problem of sampling from a p-dimensional sphere. As the dimension of the sphere increases, the acceptance probability of the iid accept-reject algorithm reduces (exponentially). Although demonstrating this in p-dimensions will be difficult, we will demonstrate this for sampling from a circle.

The target density is

$$f(x, y) = \frac{1}{\pi} I(x^2 + y^2 < 1) \,.$$

We consider the proposal distribution to a small box around around our current step. That is, given a step (x_t, y_t) , the proposal distribution is $U[x_t - h, x_t + h] \times [y_t - h, y_t + h]$. The density of the proposal distribution is:

$$q(w, z|x_t, y_t) = \frac{1}{4h} I(x_t - h < w < x_t + h) I(y_t - h < z < y_t + h) \,.$$

Since the proposal distribution is symmetric, to implement the MH algorithm, we need the ratio:

$$\frac{f(w, z)}{f(x_t, y_t)} = I(w^2 + z^2 < 1) \leq 1$$

So, the MH algorithm here can be implemented in the following way:

1. Set (X_0, Y_0) to be any coordinate inside the circle.
2. Generate $(W, Z) \sim U[x_t - h, x_t + h] \times [y_t - h, y_t + h]$.
3. If $I(W^2 + Z^2 < 1)$, then set $(X_{t+1}, Y_{t+1}) = (W, Z)$
4. Else set $(X_{t+1}, Y_{t+1}) = (X_t, Y_t)$

Run simulation separately to show animation from this.

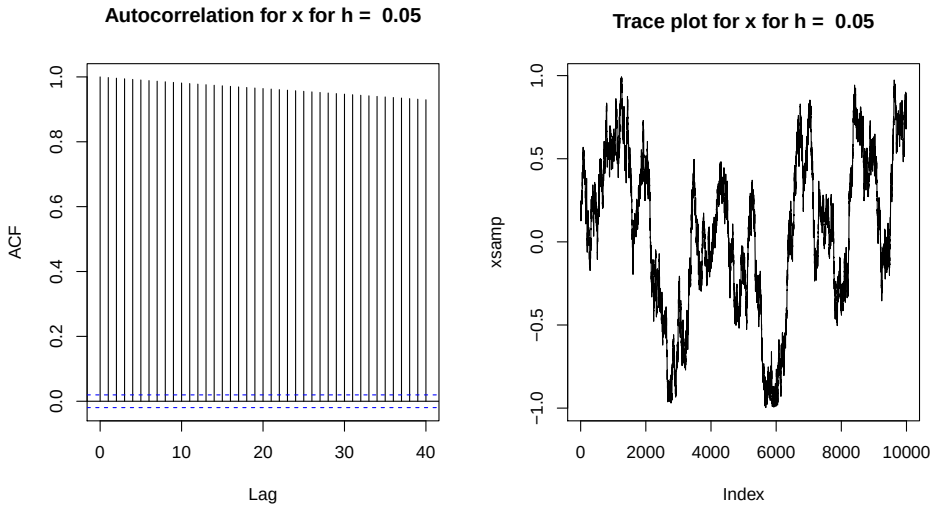
From the above example it is clear that the quality of the MCMC sampler is determined

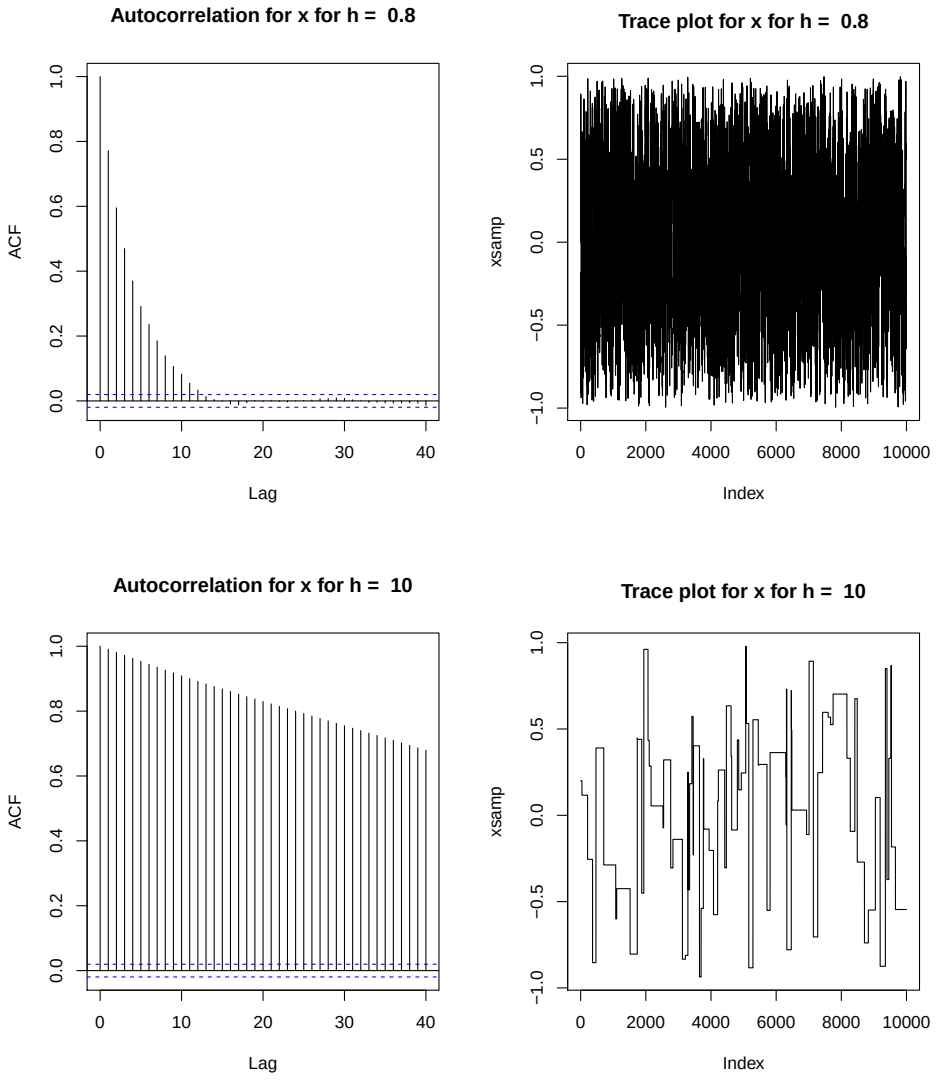
by the choice of h.

- If h is too large, then we will propose values potentially too far away, which in principle would be good, but this means a lot of the values will be rejected. That means we will stay where we are quite often, increasing the correlation in the sample.
- If h is too small, then we will make tiny moves implying we may accept a lot of them, but the samples obtained will be quite dependent.

Two visual aides to assess the performance of the sampler are

- Autocorrelation plots (ACF plots): these tell us upto how many steps (on average) is there correlation in the Markov chain. The higher the ACF correlation, the worst it in.
- Trace plot: Trace plots indicate how well the Markov chain moves. We want to see a white noise (random behavior).





Clearly the performance of the MCMC sampler can be highly impacted by the variance in the proposal distribution. A good rule of thumb is:

Aim for 45% acceptance probability for 1-dimensional target distributions and 23% acceptance probability for high-dimensional target distributions (> 5). Anything in between you can choose between 23-45%.

3 Questions to think about

- What would happen if you choose the starting point to be outside the support of the target distribution?
- In general, how can you choose a starting values?
- Although a law of large numbers holds, do you think a central limit theorem can also hold here?
- Implement the MCMC algorithm for the p-dimensional sphere problem and compare with the efficiency of the iid accept-reject sampler.

MTH 511a - 2021: L39 - Metropolis-Hastings examples

Instructor: Dootika Vats

1 Metropolis-Hastings continued

In this video, we will continue discussing the Metropolis-Hastings algorithm via a few examples.

Example 1 (Normal distribution). Suppose our target distribution is the standard normal distribution with density

$$\pi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \propto e^{-x^2/2}.$$

We do not need to know the proportionality constant, since that gets canceled out in the MH ratio. We want to first choose a proposal distribution.

Let's choose proposal distribution be a a Unif$[x-h, x+h]$ for some window h. So that

$$q(y|x) = \frac{1}{2h} I(x - h < y < x + h).$$

This is a symmetric proposal since $q(x|y) = q(y|x)$ for particular values of x and y.

Next, we need to choose a starting value: we know that the center of a normal distribution is 0, so let's start there and let's set $h = 1$.

1. Set $x_1 = 0$.
2. Draw a proposed value $y^* \sim U[x_t - 1, x_t + 1]$.
3. Calculate

$$\alpha(x_t, y^*) = \min\left\{1, \frac{\pi(y^*)}{\pi(x_t)}\right\} = \min\left\{1, \exp\left\{-\frac{y^{*2}}{2} - \frac{x_t^2}{2}\right\}\right\}$$

4. Draw $U \sim U[0, 1]$. If $U < \alpha(x_t, y^*)$. Set $x_{t+1} = y^*$.
5. Else, set $x_{t+1} = x_t$.
6. Stop when $t = T$

Notice that we set the starting value to be 0 in the above example, since the mode/mean of the target distribution is 0. Thus, 0 is an area of high probability and a reasonable starting point for the Markov chain. Choosing starting values for Bayesian problems can be tricky, but the following can be helpful:

- Start from the MLE or the method of moments estimator of the parameter.
- If you believe the prior distribution is chosen well, then you can start from the prior distribution.

Example 2 (Continuing t Bayesian example). The resulting Bayesian posterior distribution is

$$\pi(\mu|y) = c\, e^{-\frac{\mu^2}{2}} \prod_{i=1}^{n} \left(1 + \frac{(y_i - \mu)^2}{\nu}\right)^{-\frac{\nu+1}{2}},$$

where the c is unknown.

We will use a Normal distribution proposal, centered at the current value. So $q(y|x)$ will be the density of the distribution $N(x, h)$ where h is similar to stepsize. Note that this is also a symmetric density.

We can choose a starting value for μ. μ is the mean of the t likelihood. So a good $x_1 = n^{-1}\sum y_i$. We set $h = 2$ for now.

1. Set $x_1 = n^{-1}\sum y_i$.
2. Draw a proposed value $y^* \sim N(x_t, 2)$.
3. Calculate

$$\alpha(x_t, y^*) = \min\left\{1, \frac{\pi(y^*)}{\pi(x_t)}\right\} = \min\left\{1, \frac{e^{-\frac{y^{*2}}{2}}\prod_{i=1}^n\left(1+\frac{(y_i-y^*)^2}{\nu}\right)^{-\frac{\nu+1}{2}}}{e^{-\frac{x_t^2}{2}}\prod_{i=1}^n\left(1+\frac{(y_i-x_t)^2}{\nu}\right)^{-\frac{\nu+1}{2}}}\right\}$$

4. Draw $U \sim U[0,1]$. If $U < \alpha(x_t, y^*)$. Set $x_{t+1} = y^*$.
5. Else, set $x_{t+1} = x_t$.
6. Stop when $t = T$

```
##########################################
## MH samples from posterior distrbituon
# from t likelihood and normal prior
##########################################
set.seed(10)
log.post <- function(y, mu, nu)
{
  -mu^2/2 - (nu + 1)/2 *sum( log( 1 + (y-mu)^2/nu ) )
}
```

```r
# Generate the data set.
# No need for small n and mu close to zero (the prior)
nu <- 3
n <- 50
mu <- 4
y <- mu + rt(n, df = nu) # Generate the data
```

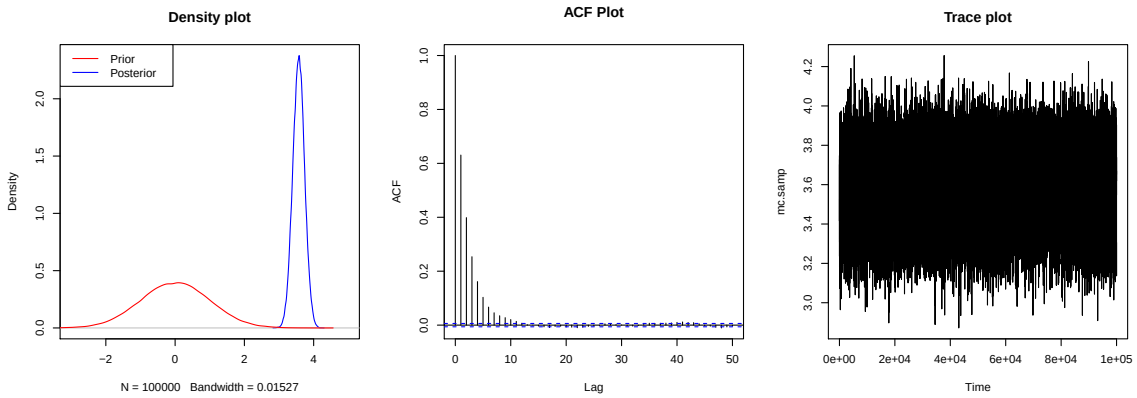
Unlike accept-reject, the MCMC algorithm will not be limited by the size of the data. Thus, we can generate more data. Now we implement the Metropolis-Hastings algorithm above.

```r

# Now compare to MCMC for the same target distribution
T <- 1e5
mc.samp <- numeric(length = T)
mc.samp[1] <- mean(y)
acc <- 0
foo <- proc.time()
for(t in 2:T)
{
  prop <- rnorm(1, mean = mc.samp[t-1], sd = .5)
  ratio <- log.post(y = y, mu = prop, nu = nu) - log.post(y = y, mu =
      mc.samp[t-1], nu = nu)

  if(runif(1) < exp(ratio))
  {
    mc.samp[t] <- prop
    acc <- acc + 1
  }else{
```

```
    mc.samp[t] <- mc.samp[t-1]
  }
}
proc.time() - foo
#   user  system elapsed
#  0.786   0.018   0.805

# Acceptance probability
# One dimensional problem, so the acceptance is good.
print(acc/T)
#[1] 0.4453
```

It is always a good idea to first look at some diagnostic plots, like the density plot, the acf plot, and the trace plot.

```
par(mfrow = c(1,3))
plot(density(mc.samp), col = "blue", xlim = c(-3,5), main = "Density plot")
lines(density(rnorm(1e5)), col = "red")
legend("topleft", col = c("red", "blue"), legend = c("Prior", "Posterior"),
    lty = 1)
acf(mc.samp, main = "ACF Plot")
plot.ts(mc.samp, main = "Trace plot")
```



From the plots above, we learn the following:

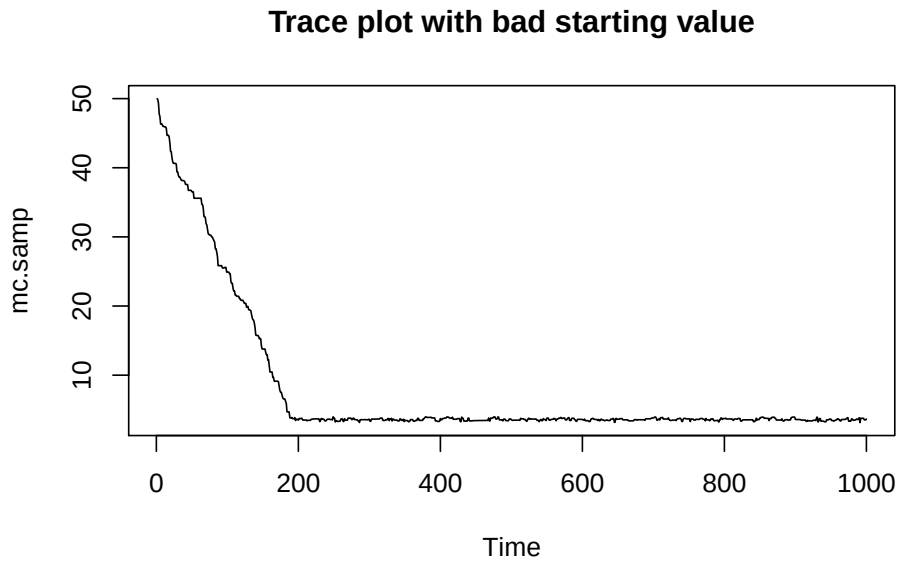
- The estimated posterior density plot is fairly smooth, indicating a reasonable quality of sampling. This makes sense since we generated $T = 10^5$ samples, which is a large number of samples for this problem.
- Our acceptance probability was 44% which was well-chosen and the ACF plots indicate reasonably low autocorrelation. Thus our samples are of a reasonable quality.
- The trace plot looks like white noise with no discernible patterns. This is what we are looking for.

We conclude that the sampler performs reasonably well and note that it is fast as well.

```
# Since the sampler looks good, we can now
# estimate the posterior mean and quantiles
mean(mc.samp)
# [1] 3.568152
quantile(mc.samp, c(.025, .975))
#     2.5%    97.5%
#3.235619 3.902179
```

For the same problem, if we start from a bad starting value, the Markov chain will look first tend towards an area of high probability.

```
# a short run again to indicate what happens
# with a bad starting value
# But choose a BAD starting values
T <- 1e3
mc.samp <- numeric(length = T)
mc.samp[1] <- 50 ## bad starting value
acc <- 0
foo <- proc.time()
for(t in 2:T)
{
  prop <- rnorm(1, mean = mc.samp[t-1], sd = .5)
  ratio <- log.post(y = y, mu = prop, nu = nu) - log.post(y = y, mu =
      mc.samp[t-1], nu = nu)

  if(runif(1) < exp(ratio))
  {
    mc.samp[t] <- prop
    acc <- acc + 1
  }else{
    mc.samp[t] <- mc.samp[t-1]
  }
}
par(mfrow = c(1,1))
plot.ts(mc.samp, main = "Trace plot with bad starting value")
```



There are multiple questions to still answer in MCMC that we will not be able to get to here. For example:

- Even though, we have non-iid samples, how do we still have a law of large numbers:
$$\frac{1}{T}\sum_{t=1}^{T} X_t \overset{a.s.}{\to} \mathrm{E}_\pi[X]\,.$$
- Does a Markov chain version of a central limit theorem hold? If yes, what is the variance in this CLT?
- Note that other posterior quantities pertaining to the target distribution can be found. For example, the variance of the target distribution can be estimated by
$$\frac{1}{T-1}\sum_{t=1}^{T}(X_t-\bar{X})^2 \overset{a.s.}{\to} \mathrm{Var}_\pi(X)\,.$$
Similarly target quantiles can be computed by sample quantiles.
- How many samples are needed in MCMC. That is, what is T? For now, you

should obtain enough samples so that the marginal density plots are relatively smooth.

- Tuning the MCMC sampler can take a lot of work. There are some other rules, guidelines, that are outside the scope of the course.
- Aside from Metropolis-Hastings algorithm, there are multiple other MCMC algorithms that we will not get to discuss here.

2 Bayesian logistic regression

Consider a Bayesian logistic regression model. For $i = 1, \ldots, n$, let

$$x_i = (1, x_{i2}, \ldots, x_{i(p-1)})^T$$

be the vector of covariates for the ith observation and $\beta \in \mathbb{R}^p$ be the corresponding vector of regression coefficients. Suppose response y_i is a realization of Y_i with

$$Y_i | x_i, \beta \sim \text{Bern}\,(p_i) \quad \text{where} \quad p_i = \frac{\exp(x_i^T \beta)}{1 + \exp(x_i^T \beta)}\,.$$

Since this is a Bayesian model, we also assume that β has the following prior distribution

$$\beta \sim N_p(0,\, I_p)\,.$$

Our goal is to find the posterior distribution and report the posterior mean and credible intervals of β. In order to do this, the first thing we do is write down the posterior distribution.

$$\pi(\beta|y) \propto \pi(\beta) \prod_{i=1}^{n} f(y_i|\beta)$$

$$\propto e^{-\beta^T\beta/2}\prod_{i=1}^{n}(p_i)^{y_i}\,(1-p_i)^{1-y_i}$$

The posterior is p dimensional, so here we need to sample from a p dimensional distribution. Our proposal distribution will be

$$q(\beta^* \mid \beta, \sigma^2) = \prod_{k=1}^{p} q(\beta^*|\beta)\,.$$

So each component is given it's own proposal value, independent of each other. We will use all normal distributions, with different step sizes $h_1, \ldots, h_p$. So we propose from

$$N_p\left(\beta_t\ ,\ \begin{bmatrix} h_1 & \ldots & 0 & 0 \\ 0 & h_2 & 0 & 0 \\ \vdots & & \ddots & 0 \\ 0 & 0 & 0 & h_p \end{bmatrix}\right)$$

Note that this is a symmetric proposal, so the MH ratio is simplifies. Since we already know the MLE of the logistic regression model, we can start from the MLE solution!

```
##########################################
## Bayesian logistic regression
## with MH implementation
##########################################
# log posterior
logf <- function(beta)
{
  one.minus.yx <- (1 - y)*X
  -sum(beta^2)/2 - sum(log(1 + exp(-X%*%beta))) - sum(one.minus.yx%*%beta)
```

```
}


bayes_logit_mh <- function(y, X, N = 1e4, prop.sd = .35)
{
  p <- dim(X)[2]
  one.minus.yx <- (1 - y)*X

  # starting value is the MLE
  foo <- glm(y ~X - 1, family = binomial("logit"))$coef
  beta <- as.matrix(foo, ncol = 1)
  beta.mat <- matrix(0, nrow = N, ncol = p)
  beta.mat[1, ] <- as.numeric(beta)
  accept <- 0

  for(i in 2:N)
  {
    #symmetric density
    prop <- rnorm(p, mean = beta, sd = prop.sd)

    # log of the MH ratio
    log.rat <- logf(prop) - logf(beta)
    if(log(runif(1)) < log.rat)
    {
      beta <- prop
      accept <- accept + 1
    }
    beta.mat[i, ] <- beta
  }
  print(paste("Acceptance Prob = ", accept/N))
```

```
  return(beta.mat)
}
```

When we run the above function, it automatically prints the acceptance probability. We now load the dataset.

```
titanic <- read.csv("https://dvats.github.io/assets/titanic.csv")


y <- titanic[,1]
X <- as.matrix(titanic[, -1])
```

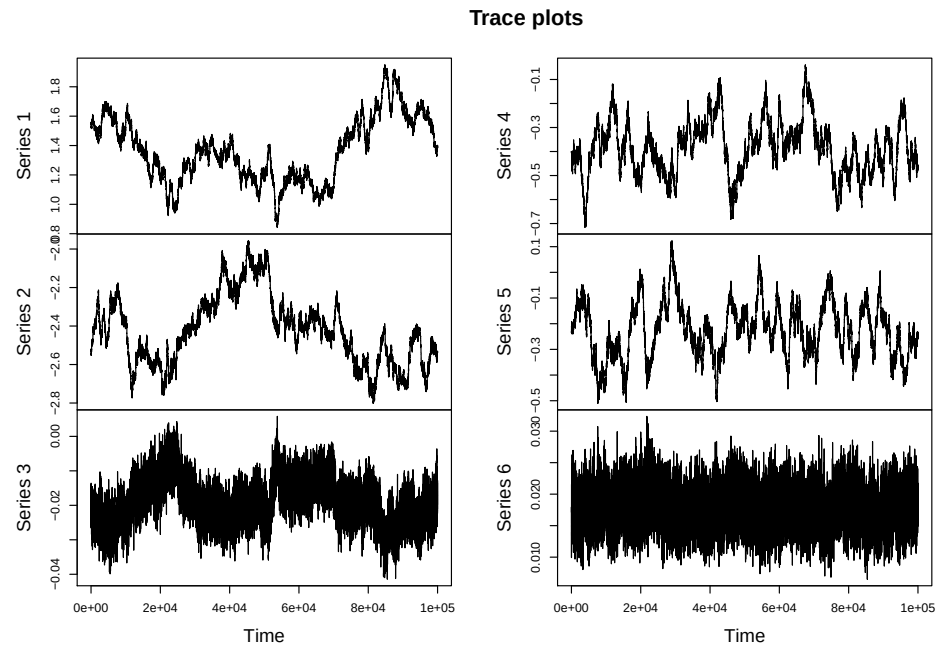
First we will try to find a proposal variance h that words reasonably well to give about 23% acceptance. We will do this by running the sampler for short (10^3) runs. First we let all proposal variance to be the same.

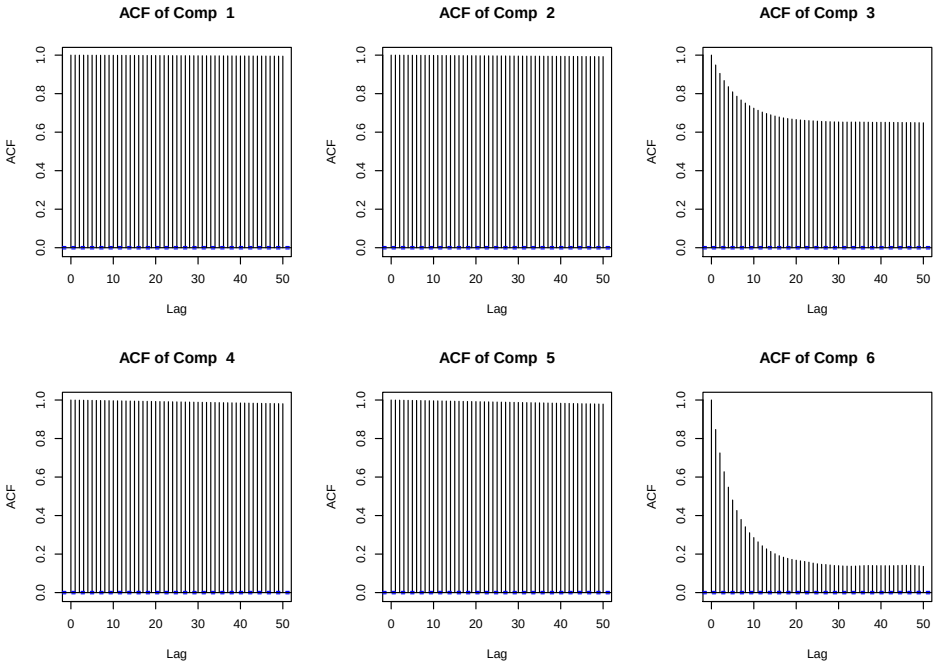
```
## acceptance is too low. we want 23%
## so decrease proposal variance
chain <- bayes_logit_mh(y = y, X = X, N = 1e3, prop.sd = .35)
#[1] "Acceptance Prob = 0"


# still too low
chain <- bayes_logit_mh(y = y, X = X, N = 1e3, prop.sd = .1)
#[1] "Acceptance Prob = 0"


# now its better
chain <- bayes_logit_mh(y = y, X = X, N = 1e3, prop.sd = .0065)
#[1] "Acceptance Prob = 0.218"
```

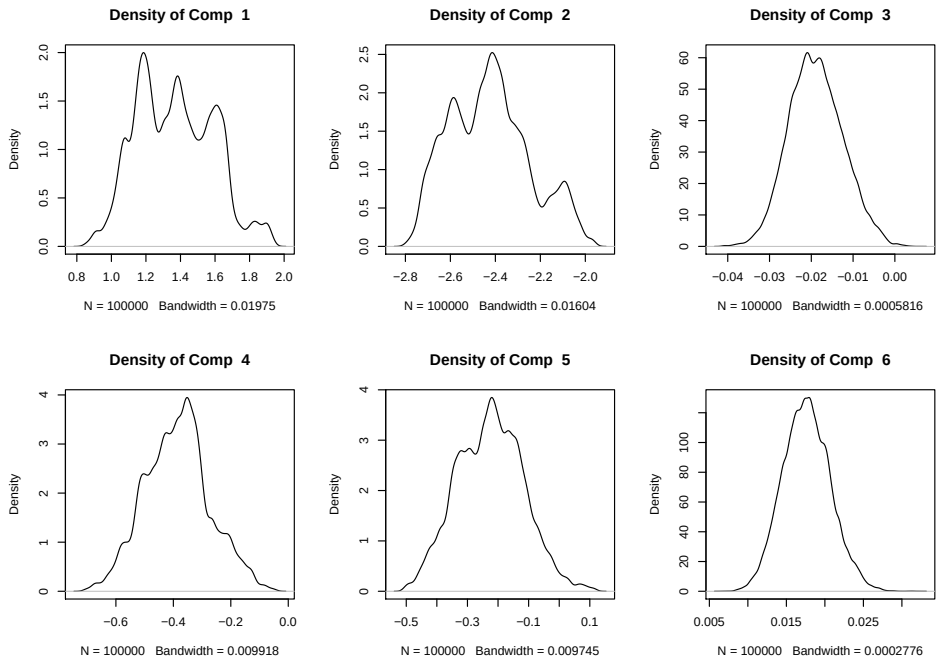
Notice our first two runs did not work well since our acceptance rate was too low. This means we were proposing large jumps and it would be better to take smaller jumps

to increase acceptance. When we reduced the proposal sd to .0065 we got a decent acceptance rate. Now we will run this same run for longer (10^5) and print diagonstics.

```
# will now run the chain much longer for 10^5
# takes a few seconds
chain <- bayes_logit_mh(y = y, X = X, N = 1e5, prop.sd = .0065)


# all trace plots
plot.ts(chain)


par(mfrow = c(2,3))
# all ACF plots
for(i in 1:dim(chain)[2])
{
  acf(chain[,i], main = paste("ACF of Comp ", i))
}


# all density plots plots
for(i in 1:dim(chain)[2])
{
  plot(density(chain[,i]), main = paste("Density of Comp ", i))
}
```


Trace plots
Series 1
Series 2
Series 3
Series 4
Series 5
Series 6
Time
Time

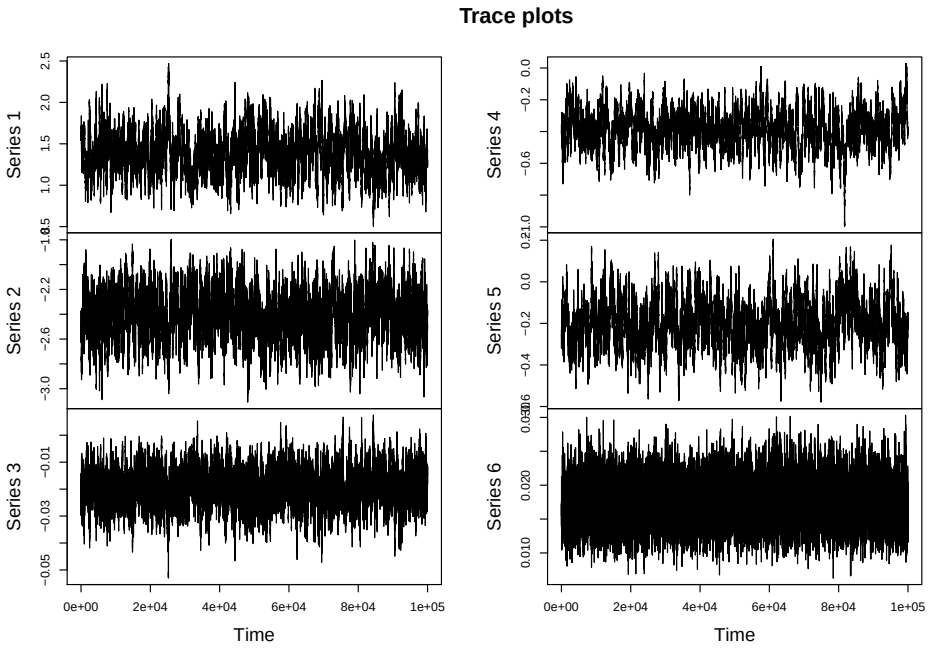

ACF of Comp 1
ACF of Comp 2
ACF of Comp 3
ACF of Comp 4
ACF of Comp 5
ACF of Comp 6
ACF
Lag

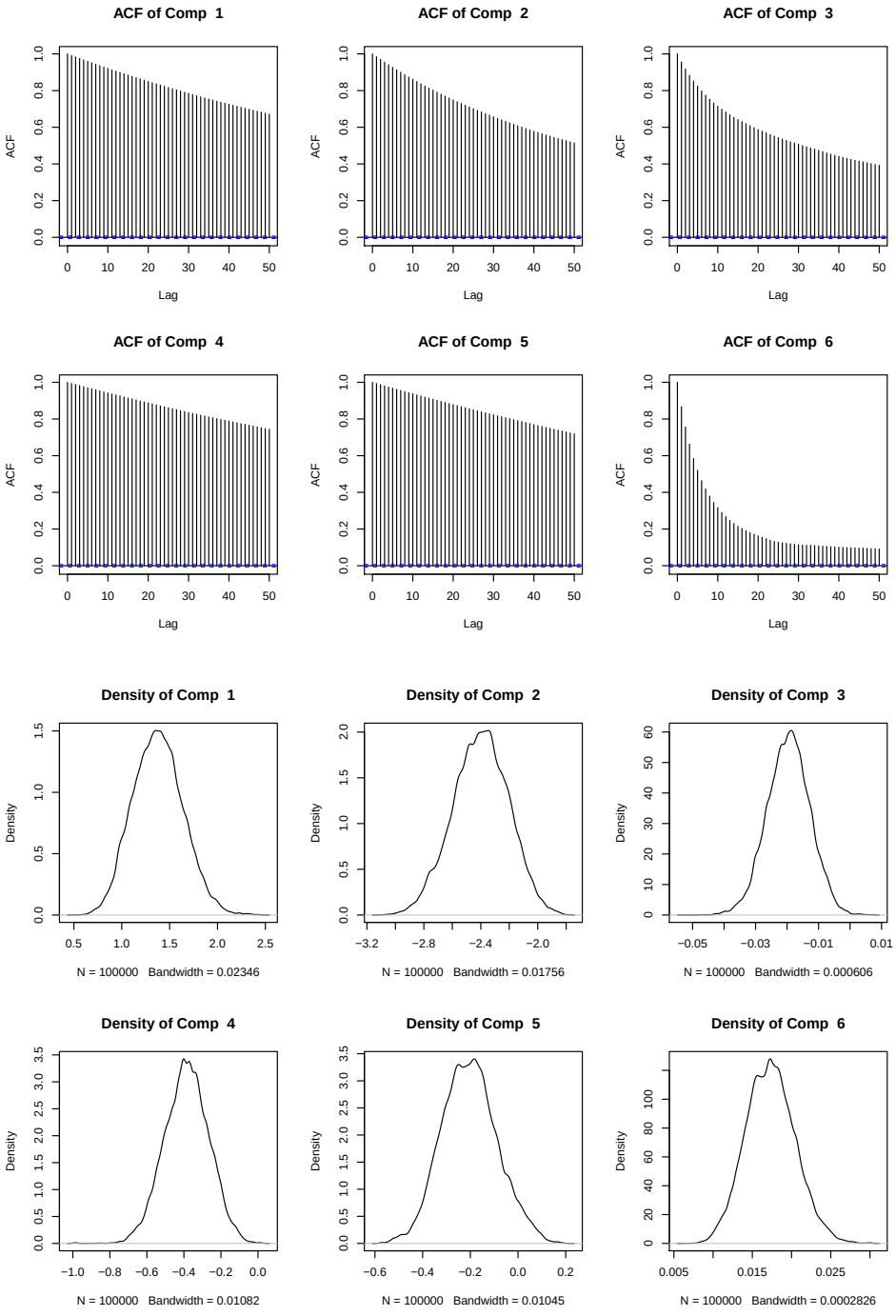


What we see is that although components 3 and 6 are well estimated with sample size 10^5, the other four are very poorly moving. This is because we choose the same proposal variance for each component which is not ideal here. We will now give each component a different proposal variance and run the sampler again for 10^5 steps.

```
# we see above that some components are ok, but 4 components are
# moving very slowly. This is because we are using the same proposal
# variance for each component, which is not adequate here.
# Below now I use different proposal variances for different
#components.


chain <- bayes_logit_mh(y = y, X = X, N = 1e5, prop.sd = c(.08, .08, .0065,
    .03, .03, .0065))
```

```r
# all trace plots
plot.ts(chain, main = "Trace plots")


par(mfrow = c(2,3))
# all ACF plots
for(i in 1:dim(chain)[2])
{
  acf(chain[,i], main = paste("ACF of Comp ", i))
}


# all density plots plots
for(i in 1:dim(chain)[2])
{
  plot(density(chain[,i]), main = paste("Density of Comp ", i))
}
```





The estimated density plots, acfs, and trace plots are much better!

Thus, we see that MCMC, although powerful, can be difficult to tune. However, once you can make it work, it works reasonable well.

3 Questions to think about

- Implement the MCMC sampler for the normal target distribution example. Change proposal variance h to see how the sampler changes.
- How would you choose the proposal distribution when the target is higher-dimensional?
- When starting from bad starting values, like the in the plot above, what do you think we should do having observed that trace plot?
- Try and implement MCMC for the Bayesian regression model.
- Obtain posterior mean and quantiles for the above implemented example. How do the final estimates compare to the MLE estimates?