Contents

1 Simulation studies

`http://192.168.50.201/statcomp/resample/simul1.html`

1.1 Introduction

The world around us is full of chance, and there are many random experiments that are of great importance to us. A gambler, for instance, is interested in guessing the outcome of the next roll of die in a casino, an economist wants to predict how the stock market is going to behave tomorrow, a geologist is puzzled about how the shape of a river would change over the next decade. In all these cases we are interested in knowing the outcome of some activity, but in spite our best effort we cannot perfectly predict the outcome.

Man has devised various techniques to deal with chance. These may be classified into roughly three categories:

1. **Wait and see:** Here we patiently wait until the end of the activity. It is the most frequently used method among laymen. Most shopkeepers in India, for example, learn about the random market trends through direct experience. Similarly, many gamblers try to design winning strategies just by trial-and-error. This approach is very simple, and sometimes quite effective. However, learning by just trial-and-error may prove too expensive. Suppose that you go to a casino and wonder if playing a certain game will be a good idea or not. If the entry fee is Rs. 50 per game, attempting 100 games to "collect data" will cost you Rs.5000.

2. **Solve probability models:** Often we can gather information about a random experiment without doing the experiment ourselves. For example, if a careful scrutiny of a die does not reveal any difference between the faces of a die, one may assume that the die is fair. Such information can be expressed as a probability model, which is a mathematical formulation of our knowledge about the random experiment. Then we can get information about the average outcome of a random experiment by just mathematically working with the model. This method has the advantage that we do not have to incur any actual loss. However, if the model is very complicated, then it may be difficult to manipulate it even with a computer.

3. **Simulate probability models:** Here also we start with a model. But instead of dealing with it mathematically, we make the computer to perform (or *simulate*) the random experiment following the model. This is possible because computers can generate random numbers[1]. Now it is just like the "wait and see" strategy, except that we do not have to wait long, since computers can perform many repitions of a random experiemtn very fast. Also, no real monetary loss is involved.

1.2 Concept of simulation

Computers can do addition, subtraction, matrix manipulation and other mathematical processing. Computers can also generate "random numbers". Simulation is the method to get benefit out of this ability of the computer.

[1] Actually computers generate only *pseudo-random numbers*. These are not random but only behave like random numbers. However, in this course we shall disregard the difference.

1.2.1 Coin tosses

Consider the simplest random experiment: a coin toss. In real life it is usually difficult to procure a coin with a given probability of head. Also if we want to perform a large number of tosses, it is definitely a torture on the fingers. But simulating a coin toss is quite simple using a computer.

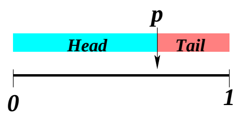
Like any other simulation that we shall do in this course, coin toss starts with a $Unif(0,1)$ random variable. Such a random variable may be obtained by either

```
X = ranuni(-1);
```
SAS

or

```
X = unifrnd(0,1)
```
MATLAB

For a $Unif(0,1)$ random variable X, we have $P(X < p) = p$ for any p in $[0,1]$. This is the main idea behind coin toss simulation. We shall instruct the computer to output "head" if $X < p$, and "tail" otherwise.



Coin toss simulation

The following code is a SAS module to simulate a coin with probability of head equal to 0.3. Write this in some file (say, `toss.sas`).

```
start toss;
  x = ranuni(-1);
  if(x< 0.3) then
    result = 'head';
  else
    result = 'tail';
finish;
```
SAS

Now submit the commands

```
%include 'toss.sas';
run toss; print(result);
run toss; print(result);
run toss; print(result);
```
SAS

In Matlab write the following code in a file called `toss.m`

```
function result=toss
x = unifrnd(0,1);
if x < 0.3
  result = 'head';
else
  result = 'tail';
end
```
MATLAB

Now issue the `toss` command a number of times. Notice how the outcome changes randomly.

Exercise: Modify the above program so that it takes a probability p as its input, and tosses a coin with p as the probability of head. Thus, `toss(0.2)` will toss a coin with probability of head 0.2. ♡

Example: Toss 100 times a coin with probability of head 0.6. Count the number of heads you get.

SAS

```
x = J(1000,1,'    ');
do i = 1 to 100;
  x[i] = toss(0.2);
end;
heads = (x='head');
nHead = heads[+];
print(nHead);
```

Here `heads` is a 0-1 array (1=head, 0=tail). The expression `heads[+]` denotes the sum of all these 0's and 1's. Notice how we have used summation for counting.

In Matlab you can write:

MATLAB

```
x = zeros(1,100);
for i=1:100
  x(i) = toss(0.6);
end
heads = (x=='head');
nHead = sum(heads);
```

♡

Example: Let us check the Laws of Large Numbers using simulation. Toss a coin with probability 0.4 of head 1000 times. What is the proportion of heads among the first 100 tosses? First store the results of the tosses in an array called `x`. Now you can find the required proportion as

SAS

```
heads = (x='head');
print(mean(x[1:100]));
```

In Matlab:

MATLAB

```
heads = (x=='head');
mean(heads(1:100))
```

What does the following code do?

SAS

```
y = cusum(heads)/(1:1000);
x = 1:1000;
%lineplot(y*x);
```

Here is its Matlab counterpart.

MATLAB

```
props = cumsum(x)/(1:1000);

plot(props);
```

♡

Exercise: Keep on tossing a coin with probability 0.7 of head until you get the first head. Let X be the number of tosses required. Then it is known that X has a *Geometric*(0.7) distribution. Check this using simulation as follows.

SAS

```
x = 0;
y = toss(0.7);
do while(y^='head');
  y = toss(0.1);
  x = x + 1;
end;
print(x);
```

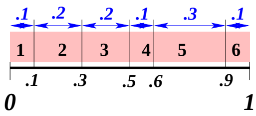
MATLAB

```
X = 0;
while(~(toss(0.7)=='head')) X = X + 1;
```

Carry this out 500 times. What is the maximum value for X that you have obtained? Make a histogram of the 500 values. ♡

1.2.2 Rolling a die

Example: Next let us simulate a roll of a die. A roll of a die produces 6 possible outputs: 1,2,...,6. We want to simulate a die with probabilities: 0.1 , 0.2, 0.2, 0.1, 0.3, 0.1. This may be achieved by partitionaing the (0,1) interval into 6 subintervals as follows:



Die simulation

The numbers 0.1, 0.3 etc are obtained by adding the given probabilities one by one (or *cumulatively*).

SAS

```
prob = cusum({0.1 , 0.2, 0.2, 0.1, 0.3, 0.1});
print(prob);
```

Now we proceed just like simulating a coin. We first generate a $Unif(0,1)$ random variable X and check in which of the 6 subintervals it lands in. If you are writing a module to simulate the die, then this checking may be done like this:

SAS

```
...
  do i = 1 to 5;
    if(X < prob[i]) then return(i);
  end;

  return(6);
...
```

In Matlab you may compute the cumulative sum as

MATLAB

```
prob = cumsum([0.1 , 0.2, 0.2 0.1, 0.3, 0.1])
```

Checking where X lands may be done like

MATLAB

```
...
for i=1:5
  if(X < prob[i])
    result = i;
    return;
  end
end

result = 6;
...
```

♡

Exercise: Follow the above guidelines to write a module in SAS (or function in Matlab) called `die(p)` to simulate a die with probabilities given by the array `p`. Roll your die 100 times. Make a histogram of the outcomes. ♡

Exercise: Simulate a die with probabilities 0.1, 0.1, 0.1, 0.1, 0.1, 0.5. Roll it 1000 times. Compute the number of cases where an even number immediately follows an odd number. For instance, among the numbers

$$1, 3, 2, 4, 3, 5, 6$$

we have exactly 2 such cases : 3,2 and 5,6.

To check whether a number is even or odd you have to divide the number by 2, and check if the remainder is 0 or 1.

SAS

```
parity = mod(x,2);
```

In Matlab

MATLAB

```
parity = x%2;
```

♡

Example: Simulating a fair die is quite easy. In fact, it is easy to simulate from the $DiscreteUnif\{1,..,n\}$ for any n. The technique is to start by generating it a $Unif(0,1)$ random variable X. Then nX is uniformly distributed over $(0,n)$. So it is equally likely to fall in any of the n intervals $(i-1,i)$ for $i = 1,...,n$. Here is the code for simulating a fair die with n faces. It returns a r by c matrix, each entry being an independent roll of a fair n-sided die.

SAS

```
start randuni(r,c,n);
  U = ranuni(J(r,c,-1));
  return(floor(n*U)+1);
finish;
```

In Matlab you need only a single function: `unidrnd`.

MATLAB

```
unidrnd(6)
```

♡

Exercise: Use `randuni` in SAS or `unidrnd` in Matlab to simulate a fair coin. Your program should return 1 for head and 0 for tail. ♡

Exercise: How would you simulate from $DiscreteUnif\{2,-3,5,4\}$? ♡

Exercise: Here is a casino version of Ludo. It consists of a 10 by 10 board with the numbers 1 to 100 written on the cells. The following cells are coloured red.

17, 23, 24, 35, 42, 56, 64, 99.

You have a counter that is moved by rolls of a fair die. Whenever you land on a coloured cell, the casino dealer pays you an amount equal to the number in that cell. For each roll of the die you pay the dealer Rs 5. Play the game 10 times. What is the avergage gain?

Here are some hints to get you started using SAS. First make an array for the red cells:

SAS

```
red = {17, 23, 24, 35, 42, 56, 64, 99};
```

If `x` is the current position of the counter then you will need to check if `x` is red or not. You can do this as

SAS

```
if(any(x=red)) then ...
```

Also, you may use a `do while` loop to move your counter:

SAS

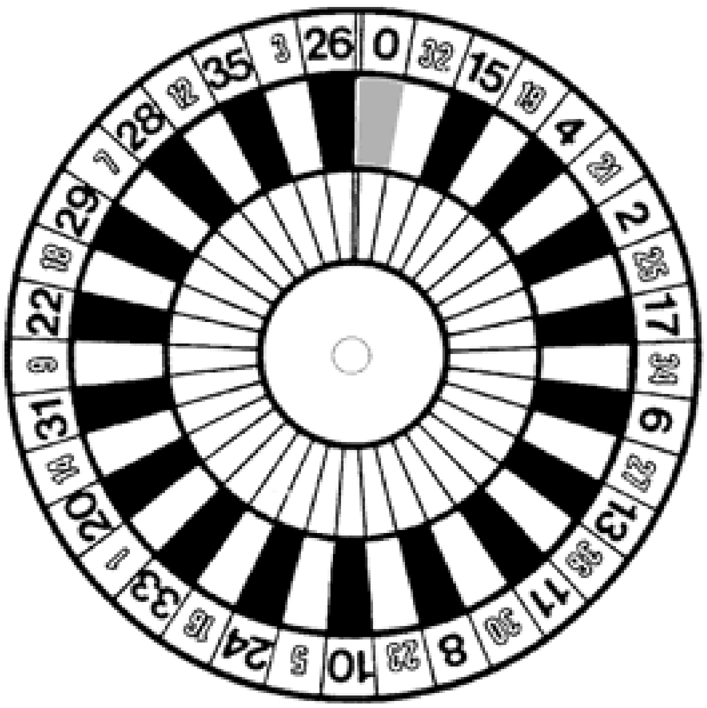
```
x = 0;
do while(x < 100);
  x = x + randuni(6);
  ...
end;
```

♡

1.2.3 Roulette

The central gambling attraction of most casinos is the game of Roulette. Most movies show you a Roulette game when they show a casino.

The main randomising device in a Roulette game is a big colourful wheel with numbered slots around it. Here is how a Roulette wheel looks like.



Roulette wheel

It has the numbers 0 to 36 written around the rim in the following order (clockwise)

> 0, 32, 15, 19, 4, 21, 2, 25, 17, 34, 6, 27, 13, 36, 11, 30, 8, 23, 10, 5, 24, 16, 33, 1, 20, 14, 31, 9, 22, 18, 29, 7, 28, 12, 35, 3, 26.

The numbers are written in three different colours: black, red and green.

> Red: 1, 3, 5, 7, 9, 12, 14, 16, 18, 19, 21, 23, 25, 27, 30, 32, 34, 36
> Black: 2, 4, 6, 8, 10, 11, 13, 15, 17, 20, 22, 24, 26, 28, 29, 31, 33, 35
> Green: 0

The casino dealer spins the wheel and drops a little ball on the wheel. When the wheel comes to rest the ball drops into one of the slots (in a fair Roulette wheel each slot is equally likely.) The player bets for some set of numbers. (The player has to pay an entry fee of Re 1 for that.) If the ball lands among a chosen set of numbers then the dealer pays money to the player, otherwise the player loses the entry fee.

When the player pays the entree fee he is given a token which he places on a special table (called *Roulette table*) to indicate his chosen set. The table is marked with the same numbers as on the wheel.

If the player places his token on a number (as in A in the picture) then he is choosing the singleton set consising of that number alone. If he places the token between two numbers (as in B) then he is betting on the set consisting of those two numbers. A list of some sample bets are given below.

Example	**The bet set**	**Dealer pays on win**
A	3	36
B	8, 9	18
C	13, 14 15	12
D	17, 18, 20, 21	9
E	0, 1, 2, 3	9
F	28, ..., 33	6
G	the entire second column	3
H	1,...,12	3
J	last two columns	1.5
K	13,..,36	3

We shall simulate this game assuming that the wheel is fair. Then the wheel can be simulated by

SAS

```
x = randuni(37)-1;
```

Each bet specifies a set of numbers. If the ball stops at any number in the set then you win. Consider the set of red numbers:

SAS

```
red = {1,3,5,7,9,12,14,16,18, 19,21,23,25,27,30,32,34,36};
```

Is `x` in the set `red`? This is found by the `any` function:

SAS

```
if(any(x=red)) then
  print 'You have won';
else
  print 'You have lost';
```

Essentially the same code works in Matlab, too:

MATLAB

```
x = unidrnd(37)-1;
red = [1,3,5,7,9,12,14,16,18, 19,21,23,25,27,30,32,34,36];
if any(red==x)
  fprintf('You have won');
else
  fprintf('You have lost');
end
```

Exercise: Write a neat version of simulated Roulette that will allow the user to play the game. It should print pretty messages. ♡

1.2.4 Playing cards

Playing cards have a long history as gambling tools. Here we shall learn how to simulate them. An ordinary deck of cards contains 52 cards (excluding the two jokers). Each card has a suit as well as a denomination. There are 4 suits: spade, club, diamond, heart, and 13 denominations: 2,...,9,10, Jack, Queen, King, Ace. Let us make two arrays for these:

SAS

```
suit = {'spades', 'clubs', 'diamond', 'heart'};
denom = {'2', '3', '4', '5', '6', '7', '8', '9', '10',
         'Jack', 'Queen', 'King', 'Ace'};
```

Example: Many magic tricks and gambling require one to pick a card at random from a deck. Here is one way of simulating it: First pick the suit randomly:

SAS

```
s = randuni(4);
```

Then pick the denomination:

SAS

```
d = randuni(13);
```

So the entire code is

SAS

```
start card;
suit = {'spades', 'clubs', 'diamond', 'heart'};
denom = {'2', '3', '4', '5', '6', '7', '8', '9', '10',
         'Jack', 'Queen', 'King', 'Ace'};

s = randuni(4);
d = randuni(13);

print(denom[d] || 'of' || suit[s]);
```

♡

Exercise: Can you write a program that achieves the same goal but needs only a single random number generation? ♡

While writing programs it is often easier to think of the 52 cards as labelled by the numbers 1,2,...,52. Thus, card 1 is "2 of spades", card 2 is "3 of spades" and so on until "ace of hearts" which is card 52. The following code converts the number `k` into suit `s` and denomination `d`.

SAS

```
s = mod(k-1,4) +1;
d = mod(k-1,13) +1;
```

In Matlab,

MATLAB

```
s = (k-1)%4 +1;
d = (k-1)%13 +1;
```

Now suppose that you are to pick 10 cards at random from a deck. In statistical terminology you are drawing an SRSWOR of size 10 from a population of size 52. Let us start by creating the population and initialising the sample to an array of length `k` containing all -1's.

SAS

```
popln = 1:52;
samp = J(1,10,-1);
```

Now you have to enter a loop of size `k`. In each pass you will pick a card from the remaining deck. In the i-th pass, you have $52-(i-1)$ cards. Pick one of these at random and remove it from the population:

SAS

```
x = randuni(52-(i-1));
samp[i] = popln[x];
popln = remove(popln,x);
```

The Matlab code corresponding the last line above is

MATLAB

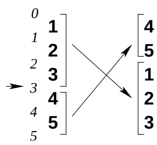
```
popln = popln[~x];
```

Exercise: Write a code to draw an SRSWOR of size k from a population of size n. ♡

Many card games start by shuffling the deck. An ideal shuffle orders the cards randomly, so that each of 52! possible orders are equally likely. This is very easy to simulate using SRSWOR. (How?) Matlab users may also use the `randperm` function.

Exercise: Write a code to draw an SRSWR of size k from a population of size n. ♡

When we, human beings, shuffle a deck by hand, the shuffling is always imperfect. It is possible to simulate these imperfect methods in the computer. The simplest method is called "cutting" the deck. Here we cut the deck at some random point chosen somewhere around the middle of the deck, and put the lower part on top of the upper part. The probability model for this as follows.



Cut shuffle

Suppose that our original deck is 1 2 3 4 5, with 1 at top and 5 at bottom. It can be cut at 6 points that are shown as 0,1,...,5 in the diagram. Human beings tend to choose the cut point near the center. To simulate this we generate the cut point X as a $Binomial(5, 0.5)$ random variable. This splits the deck into two parts: the top X cards and bottom $n-X$ cards. Now we swap the two parts. The diagram shows the case $X = 3$.

To shuffle a deck of 52 cards the binomial cut point is generated by

SAS

```
X = ranbin(-1,52,0.5);
```

Suppose that `where` turns out to be 33. Then the swapping may be done like

SAS

```
deck[(X+1):52] || deck[(1:X)];
```

Here deck is an array of length 52 storing the original permutation of the cards. Notice the `||` operator. It concatenates two matrices *horizontally.* The Matlab code is similar, using `binornd` to generate binomial random numbers and `[]` for matrix concatenation.

Exercise: Write a program that simulates the cut method for a deck of n cards. ♡

As any experienced card player will tell you, the cut method is a very bad way to shuffle cards. A much more reliable way is the *rifle* shuffle (also known as *Faro* shuffle or *dovetail* shuffle.)

Here is a well-known mathematical model of rifle shufle. First split the deck into two parts binomially just as in the cut method. Take the top half in your left hand, and the other half in your right. Now you have to release the cards randomly from both the hands. Mathemtically this is done as follows. If at any stage there are a cards in your left hand and b cards in your right, then the next card comes from the left hand with probability $a/(a+b)$ and from the right with probability $b/(a+b)$.

Exercise: Write a program to simulate rifle shuffle for n cards. Here are some hints. During rifle shuffle you are working with three piles of cards: the pile in your left hand, the pile in your right hand, and the pile growing on the table. Let us call them `left` , `right` and `table`, respectively. Also, you will need three counters `a,b` and `k` to store the sizes of these three piles. Intially `k` is 0. At the the end of the shuffle it will equal n. Let us see how you drop a card from your left hand. The code is

SAS

```
table[n-k+1] = left[a];
left = left[1:(a-1)];
k = k + 1;
a = a - 1;
```

Similarly for dropping cards from the right hand. ♡

1.2.5 Capture-recapture method

This is a popular method to estimate the number of fish in a lake:

> Suppose that there are N fish in a lake. N is unknown. We want to estimate N. In the capture-recapture method we start by capturing some number of fish (say m.) They are tagged (eg, specially marked with some water-resistant colour) and released back into the lake. After a week or so again some number of fish (say n) are caught. Some of these will be from tagged from last time, while the rest will be new. Let the number of tagged fish among these n be k. Then we estimate the total number(N) of fish in the lake by solving
>
> $$k/n = m/N$$
>
> One motivation[2] behind the equation is that assuming the m tagged fish have mixed well among the other fish, the proportion of tagged fish in the sample (k/n) should equal the proprotion of tagged fish in the population (k/N). Solving the equation we get
>
> $$N = mn/k.$$

We want to simulate this procedure. Suppose that $N = 1000$. Denote the set of fish by an array:

SAS

```
fish = 1:1000;
```

We shall draw a SRSWOR of size $m = 100$ from it.

SAS

```
capture = SRSWOR(1000,100);
```

Now we shall tag all the fish in `capture`. We shall do so by making them negative.

SAS

```
fish[capture] = -fish[capture];
```

Now we shall release the fish and let them mix up again. Then we shall draw a recapture of size $n = 200$.

SAS

```
recapture = SRSWOR(1000,200);
```

Let us count how many of the recaptured fish are tagged (ie, negative):

SAS

```
tagged = (fish[recapture] < 0);
k = tagged[+];
```

Exercise: Write a code called `caprec(N,m,n)` using the above idea. It should return the estimate of N. ♡

2 Continuous simulation

http://192.168.50.201/statcomp/resample/simul2.html

In the last lecture we have done some discrete simulations. Here we shall simulate from continuous distributions.

2.1 Poisson Process

Consider customers coming to a shop at random times. We start our clock at time $t = 0$. The first customer arrives at time X_1 which is distributed as $Expo(\lambda)$(i.e., exponential with mean $1/\lambda$.) The next customer arrives X_2 time *after* that, i.e., at time $t = X_1 + X_2$, where X_2 is an $Expo(\lambda)$ random variable independent of X_1. Similarly, the third customer arrives another independent $Expo(\lambda)$ random variable later. We call this arrival process as *Poissson Process with parameter* λ or $PP(\lambda)$ for short. Simulating a $PP(\lambda)$ is basically generating iid $Expo(\lambda)$ random variables and adding them one by one.

Let $N(t)$ denote the number of customers arrived on or before time t. We want to plot $N(t)$ against t upto time $Tmax$. The following SAS code generates $Expo(\lambda)$ random variables:

SAS

```
X = ranexp(-1)/lambda;
```

We use this inside a `do while` loop until we reach $t = Tmax$.

SAS

```
start PP(lambda, Tmax);
  arrivals = {};
  now = 0;
  do while( t < Tmax );
    gap = ranexp(-1)/lambda;
    now = now + gap;
    arrivals = arrivals || now;
  end;
  return(arrivals);
finish;
```

Now plot `1,2,...,ncol(arrivals)` against `arrivals`. Similar code works for Matlab, except that Matlab generates $Expo(\lambda)$ random variables by

MATLAB

```
X = exprnd(1/lambda);
```

[2] One may also apply MLE for Hypergeometric distribution.

2.2 Simulated cell culture

Biologists often culture cells in the laboratory to study how the cells multiply. Cancerous cells, for instance, have a different birth rate than normal cells, and this may help to detect the cancerous cells.

Suppose that we start with just one cell in the culture dish. After a random wait period (t_1) this cell will split into two cells. Then each of the two cells will wait for a random amount of time before splitting up into two cells. The two cells will behave in an iid manner (i.e., the wait periods will be iid random variables.) And this process goes on and on. For simplicity we do not allow cells to die.

A commonly used mathematical model for the wait time is $Expo(\alpha)$ We sometimes refer to this situation by saying that the split rate[3] of cells is α. If there are n cells at some given time (T), then each of them will wait for a random amount time distributed as iid $Expo(\alpha)$. In other words, if the i-th cell waits for X_i amount of time, then

$$X_1, ..., X_n \text{ are iid } Expo(\alpha).$$

So the first birth will occur after

$$min(X_i)$$

amount of time. By a property of exponential distribution, this follows $Expo(n\alpha)$ distribution.

Exercise: Prove this property of the Exponential distribution. ♡

So the following code does the necessary simulation.

SAS

```
start growCells(Tmax, alpha);
count = 1;
T = 0;
do while(T[count]<Tmax);
  gap = ranexp(-1)/(count*alpha);
  T = T || (T[count]+gap);
  count = count + 1;
end;
return(T);
finish;
```

Here's the Matlab version.

MATLAB

```
function T = growCells(Tmax, alpha)
count = 1;
T(1) = 0;
while T(count)<Tmax
  gap = exprnd(1/(count*alpha);
  T(count+1) = T(count) + gap;
  count = count + 1;
end
```

Exercise: Suppose that cells have birth rate α and death rate β. This means that the moment a cell is born Nature generates two independent random variables for the cell: lifetime $X \sim Expo(\alpha)$ and split time $Y \sim Expo(\beta)$. The cell dies after time X and if $X < Y$, and splits after time Y if $X > Y$. Notice that

[3] In the literature it is often called "birth rate". But "split" is a better term, because after a split the parent cell ceases to exist as a separate entity, only the two children remain.

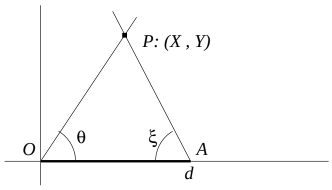
once it splits, the two new cells behave independently, and so X is not used in this case. Simulate this up to time $Tmax$. Plot $N(t)$ against t, where $N(t)$ is the number of living cells at time t. Try different values of α and β. ♡

Exercise: There are two types of cells, normal and mutant. The mutant cells are potentially cancerous and have different split rates. When a normal cell splits, it may either produce two normal cells (with probability p), or one normal and one mutant cell(with probability $1-p$). When a mutant cell splits it produces two mutant cells. The splitting rate for normal cells is α and for mutant cells is β. Simulate this up to time $Tmax$. Plot $N(t)$ against t, where $N(t)$ is the number of normal cells at time t. Try different values of α and β. ♡

Exercise: Again consider the cell culture situation where the cells do not die. But now the cells split after a wait period that is distributed as $|Z|$, where Z is a $N(0,1)$ random variable. Start with a single cell and simulate until you have 1000 cells. ♡

2.3 Land surveys

In most land surveys, it is easier to measure angles than distances. So people first measure angles and then use trigonometry to compute distances. Consider the following situation.

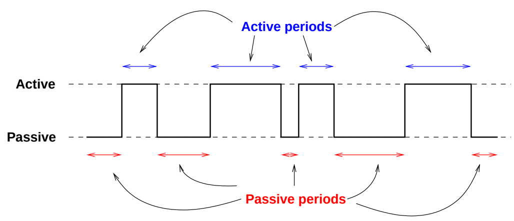


Survey

The distance d is known exactly. We want to find the coordinates of the point P. For this we have measured the two angles θ andξ. Now, owing to mechanical imprecision the angles could not be measured exactly. Our measured angles are independently distributed uniformly over a range of 0.2 radians centered around the true value.

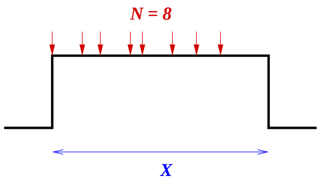
Exercise: If the true values of θ and ξ are 1.2 radians and 1.4 radians, respectively, then simulate the measurement process 100 times and make a scatter plot of the 100 estimated locations of P. Take $d = 50$ metres. ♡

Exercise: Consider a particle detector on which particles impinge following $PP(1)$. The detector can be in one of two states: active and inactive. It starts in the inactive state, and goes into the active state when a particle hits it. After 1 sec it becomes inactive again, unless there are further particles in the meantime to keep it active longer. The durations of the active and inactive period are recorded.



Active and inactive periods

Conside the duration X of an active period. Let N be the number of particles during this period. Clearly, $X \geq 1$. If $X = 1$ then $N = 1$. In fact, $X \leq N + 1$, since each particle accounts for at most 1 sec of active period.



Multiple particles during an active period

A scientist wants to estimate N from X. Help him by simulating the process to get 100 pairs (X_i, N_i) and make a scatter plot. ♡

2.4 Mixture distributions

Sometimes we have to simulate from mixture distributions with densities like

$$0.4\phi(x; 3, 1) + 0.6\phi(x; 2, 4).$$

Here $\phi(x; \mu, \sigma^2)$ denotes $N(\mu, \sigma^2)$ density. First observe that this *is* indeed a density. In fact, any convex combination[4] of densities is a density. The method to simulate from a mixture is to first consider the coefficients as a discrete probability vector. In this case the probability vector is $(0.4, 0.6)$. Toss a coin with probability 0.4. If you get a head, then generate X from $N(3, 1)$, else from $N(2, 4)$. Then X has the required mixture distribution.

3 Computation by simulation

`http://192.168.50.201/statcomp/resample/simul3.html`

So far we have learned how to simulate random experiments. Now it is time to learn how to compute variaour things based on simulation:

1. moments
2. medians and other quantiles
3. probabilities

Any computation based on simulation is approximate.

3.1 Computation of moments using simulation

To compute moments you have to use approapriate averages of the simulated data. If you want to compute $E(X^k)$ using simulation, then first simulate N observations from X (N large) and then average the k-th powers of these N numbers.

Example: Consider once again the capture-recapture method. We want to find the expected value of our estimate of the total number of fish in the lake. We shall take the total number of fish to be $N = 1000$, and use capture size $m = 50$ and recapture size $n = 100$. Carry out the process 500 times, say, and average all the estimates.

SAS

```
tot = 0;
do i = 1 to 500;
  tot = tot + caprec(1000,50,100);
end;
print(tot/500);
```

[4] A convex combination is a liear combination where the coefficients are nonnegative and add up to 1.

♡

Exercise: Using the same N, m, n estimate the variance of the estimator. ♡

Example: Let X be number of tosses you need until (and including) you get the pattern HHT for the first time. Assume that $P(H) = 0.3$. We want to find $E(X)$. The following code will simulate the random experiment once.

SAS

```
beforeLast = toss(0.3);
last = toss(0.3);
current = toss(0.3);
X = 3;
do while(^(beforeLast==1 && last==1 && current==0));
  beforeLast = last;
  last = current;
  current = toss(0.3);
  X = X + 1;
end;
```

Now perform the experiment a large number of times and average. ♡

Exercise: Estimate the variance of X in the above example. The matlab function for computing variance is `var`. ♡

Exercise: Let Y be the number of rolls of a fair die required until (and including) the first time you get the pattern 34 ot or 61. Estimate $E(Y)$ and $E(Y^3)$ using simulation. ♡

3.2 Computation of median and quantiles using simulation

To compute the p-th quantile of a random variable X, first simulate N observations from X, where N is large. Then use the appropriate sample quantile.

Example: Let us find out the 0.95-th quantile of $N(0, 1)$ using simulation. For this purpose we shall start by generating, say, 1000 samples from $N(0, 1)$. Then we shall take their 0.95-th sample quantile.

SAS

```
x = rannor(J(1000,1,-1));
x = sort(x);
print(x[950]);
```

MATLAB

```
x = normrnd(0,1,1000,1);
prctile(x,95)
```

♡

Exercise: Use simulation to compute the interquantile range of $Beta(3, 4)$ distribution. ♡

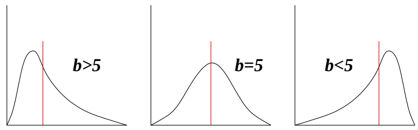
Exercise: Compute the interquantile range of X, where X is the modulus of the largest eigenvalue of a 3 by 3 matrix with iid $N(0, 1)$ entries. The following command gives you the absolute eigenvalues in decreasing order.

MATLAB

```
abs(eig(normrnd(0,1,3,3)))
```

♡

Exercise: I have a sample of size 101 from $Beta(5, b)$ distribution for $b > 1$. Such a density looks like one of the following:



Different forms of beta density

The vertical lines show the median of the distribution. Our aim is to test

$$H_0 : b = 5 \text{ against } H_1 : b < 5.$$

For this purpose we use the sample median as our test statistic. We shall reject H_0 if the sample median exceeds a suitable critical value, c. We shall use 90% level of significance. Use simulation to compute c. ♡

3.3 Computation of probabilities using simulation

To compute probability of some event A by simulation we should simulate some large number, N, of cases and count the number of cases for which A has ocurred. If this number is n, then the probability is approximately n/N.

Example: Recall the "cut" method of shuffling that we have discussed earlier. It was remarked then that it is not a good method of randomising the cards. We shall explore this now using simulation. consider some card, say the 23-rd card from the top. In the intial (unshuffled) deck it was followed by the 24-th card. What is the chance that the 23-rd card is followed by the 24-th card in a well-shuffled deck? It is 1/51 (why?). Let us find the probability that the 24-th card follows the 23-rd card in deck after 8 cuts.

SAS

```
count = 0;
do simul = 1 to 1000;
  deck = 1:52;

  do i = 1 to 8;
    deck = cut(deck);
  end;

  do j = 1 to 51;
    if(deck[j]=23 & deck[j+1]=24) then count = count + 1;
  end;
end;

print(count/1000);
```

Almost the same code works for Matlab. ♡

Exercise: Use simulation to compute the probability that the 24-th card follows the 23-rd card after 8 rifle shuffles. ♡

3.3.1 A magic trick

Claude Shannon invented the following card trick based on rifle shuffle.

> The magician produces a deck of 52 cards. The audience rifle shuffles it once behind his back. Then they remember the top card of the shuffled deck, and insert this card at any random place inside the deck. After this they again rifle shuffle the deck twice and hand back the deck to the magician. The magician goes through the deck and brings out the card remembered by the audience!

The trick lies in the fact that a rifle shuffle produces two increasing subsequences in a deck. To understand this let the original deck be consists of 10 cards:

1 2 3 4 5 6 7 8 9 10

If we split the deck between 6 and 7, say, then a typical rifling will produce

1 2 3 **7** 4 **8** **9** 5 6 **10**.

The two increasing subsequences are 1,2,3,4,5,6 and 7,8,9,10. The idea is to start counting from 1 until you find a k such that $k+1$ comes *before* it. Then $1, ..., k$ is a subsequence. The next subsequence starts $k+1$ onwards. It should not be difficult to see that n rifle shuffles can produce at most 2^n subsequences. If n is small it is very unlikely that any of the subsequences is of length just 1. Now when the audience inserted the top card inside the deck a subsequence of length 1 was formed (except, say, for the unlilely event that the card happens to get pefectly aligned with some existing subsequence.) Thus the magician just goes through the deck to locate the subsequences, and picks up the card that forms a singleton subsequnece.

This trick is thus a probabilisitc trick and there is a chance that it might fail to work (e.g., if the card gets aligned with some existing subsequence.) We want to find the probability that the trick works.

Exercise: Compute by simulation the probability that Shannon's trick as described above will succeed. For this we shall assume that when the audience inserted the top cards inside the deck, it was inserted after X cards from the top where $X \sim Binomial(51, 0.5)$. [Hint: You may find it useful to write a function/module that takes a deck (i.e., a permutation of 1,...,52) as its input and produces an array `x` as its output where the the i-th rising subsequence consists of cards `x[i],...,x[i+1]-1`.] ♡

3.4 Sampling distribution

Suppose that you have a sample $X_1, ..., X_n$ iid F, where F is some distribution. You have computed a random quantity $T(X, F)$ that is based on the sample as well as (possibly) on the underlying distribution. Then $T(X, F))$ is a random variable, whose distribution is called its *sampling distribution.*

Example: Let $\mathbf{X} = X_1, ..., X_n$ be our sample, and let $S(\mathbf{X})$ be a statsitc to estimate the mean $\theta(F)$ of F. Then its bias is defined as

$$E(S(\mathbf{X}) - \theta(F)),$$

which is the mean of the sampling distribution of

$$T(\mathbf{X}, F) = S(\mathbf{X}) - \theta(F).$$

Similarly, the second raw moment of this sampling distribution gives the M.S.E. of $S(\mathbf{X})$. ♡

Sampling distribution is, in a sense, the most important object in statistics. We shall denote it by the formula $LAW(T(\mathbf{X}, F), F)$. Notice that F is present twice in this formula. The first occurence tells you that T itself may involve F. The second occurence tells you that $\mathbf{X}$ is an iid sample from F.

If we are able to compute quantities like mean, variance and quantiles of $LAW(T(\mathbf{X}, F), F)$, then we can solve many statistical problems. We have already seen the case of bias and MSE. Some other examples are confidence intervals $T(\mathbf{X}, F)$ is a pivotal quantity, and the end points of the confidence interval are quantiles of $LAW(T(\mathbf{X}, F), F)$. In testing problems, $T(\mathbf{X}, F)$ is the test statistic and the cut-off points are quantiles of $LAW(T(\mathbf{X},F),F)$, where F is the null distribution.

If F is known and easy to simulate, we can simulate from it to compute $LAW(T(\mathbf{X},F),F)$. But for most real problems F is unknown. In such a situation we generalise the method of simulation to **resampling techniques**. We shall discuss three important resampling techniques in this course:

1. Bootstrapping
2. Cross validation
3. Jackknifing

In all these cases, we somehow avoid fresh simulation from F and re-use part of the orginal data in place of freshly simulated data.

4 Bootstrap

`http://192.168.50.201/statcomp/resample/boot1.html`

4.1 Introduction

Set up: We have data $\mathbf{X}$ from unknown distribution F. We want to know about $LAW(T(\mathbf{X},F),F)$. We do not have enough information about F to be able to simulate from F.
Example: Suppose that $X_1,...,X_{101}$ are iid with common distribution F. Let $\theta(F)$ be the mean of F. We want to estimate $\theta(F)$ by the median, M, of X_i's. The M.S.E. of this estimator,

$$E(M(\mathbf{X})-\theta(F))^2,$$

is the second moment of $LAW(T(\mathbf{X},F),F)$, where

$$T(\mathbf{X},F)=M(\mathbf{X})-\theta(F).$$

♡

Exercise: Show that if $F=N(0,1)$, the density of $LAW(T(\mathbf{X},F),F)$ is

$$\text{constant } (\Phi(x)(1-\Phi(x)))^{50}\phi(x)$$

over the entire real line. Work out the value of the constant. ♡

4.2 The bootstrap idea

Note that if you know F then you can easily get an idea about this distribution, because then you can compute $\theta(F)$, and get fresh simulations of $\mathbf{X}$. However, this is never going to happen in practice because if you already *know* F, why on earth would you ever think of *estimating* its mean? However, bootstrapping is a method that lets you have the advantage of knowing F, even when you actually do not know it. Here's the idea:

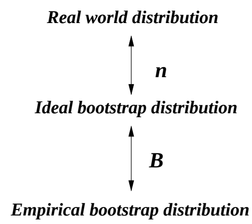
Try to approximate F by some distribution F^* that

1. is close to F
2. is *completely* known
3. can be easily simulated from.

Use F^* as a proxy for F. This is a two-stage concept: First, use $LAW(T(\mathbf{X}, F^*), F^*)$ as an estimate of $LAW(T(\mathbf{X}, F), F)$. Notice that *both* the occurences of F have been replaced by F^*. In particular, if your aim was to find mean/median/variance etc of $LAW(T(\mathbf{X}, F), F)$ then use the mean /median/variance etc of $LAW(T(\mathbf{X}, F^*), F^*)$ as its estimate. $LAW(T(\mathbf{X}, F), F)$ is called the **real world distribution**, while $LAW(T(\mathbf{X}, F^*), F^*)$ is called the **ideal bootstrap distribution**. Its mean/median/variance etc are called the ideal bootstrap mean/median/variance etc. Observe that the ideal bootstrap distribution is a function of the data, and hence is completely known, at least in principle. However, in practice it may be quite complicated to work with. In this case, we simulate fresh data from F^* to approximate the ideal bootstrap distributuion by the empirical distribution of

$$T(X_1^*, F^*), T(X_2^*, F^*), ..., T(X_B^*, F^*),$$

where each X_b^* is an iid sample of size n drawn from F^*. This distribution is called the **empirical bootstrap distribution**. We shall denote it by $EMP\{T(X_b^*, F^*); b = 1...B\}$. Its mean/median/variance etc are called empirical bootstrap mean/median/variance etc.



Two layers in bootstrap

Notice that there are two sample sizes involved here: the original sample size n, and the bootstrap replication number: B. Each of the B bootstrap replicates is of size n; so we are generating Bn random numbers in all. If n is large then the ideal bootstrap distribution is close to the real distribution. If B is large then the empirical bootstrap distribution is close to the ideal bootstrap distribution. Out of these two sample sizes, n is already fixed by the available original sample. B can be increased by changing the bootstrap program. It may be tempting to increase B too much. But it is fruitless since, if n is not large the gap between real world distribution and ideal bootstrap distribution cannot be decreased by merely increasing B.

You do not need the empirical bootstrap distribution if the ideal bootstrap distribution is easy to work with.

4.3 Parametric bootstrap

Suppose that $\mathbf{X}$ consists of $X_1, ..., X_{200}$ iid $F = N(\mu, \sigma^2)$, μ and σ^2 being unknown. Here $n = 200$. We want to estimate $\mu = \mu(F)$ by the 5% trimmed mean, $M(\mathbf{X})$, which is the average of the central 190 X_i's. We want to compute the M.S.E., which is the second (raw) moment of

$$LAW(M(\mathbf{X}) - \mu(F), F).$$

Here $T(\mathbf{X}, F) = M(\mathbf{X}) - \mu(F)$. We cannot simulate from F, because the parameters are unknown.

Parametric bootstrap suggests estimating μ, σ^2 from the data, by by $\bar{X}, S^2$, say. Plug these in place of the parameters to get our $F^* = N(\bar{X}, S^2)$. This F^* is completely known and is easy to simulate from.

Replacing *both* occurences of F by F^* we get the ideal bootstrap distribution

$$LAW(M(\mathbf{X}) - \mu(F^*), F^*) = LAW(M(\mathbf{X}) - \bar{X}, F^*).$$

We need to compute its second (raw) moment, but because of the presence of $M(\mathbf{X})$ this is not simple. So we shall need to work with the empirical bootstrap distribution. Let us choose $B = 100$ (arbitrarily). Draw 100 samples (each of size $n = 200$) from F^*, that is $N(\bar{X}, S^2)$. Call these as

$$X_1^*, ..., X_{100}^*.$$

For each of these compute the 5% trimmed mean and subtract the *original sample mean* to get 100 numbers:

$$M(X_1^*) - \bar{X}, ..., M(X_{100}^*) - \bar{X}.$$

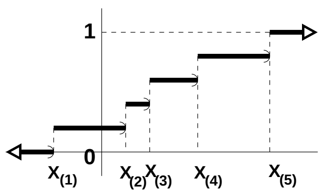
The emprical distribution of these 100 numbers is the emprical bootstrap distribution. So the required estimate is just the average of the squares of these 100 numbers.

4.4 Nonparametric bootstrap

This is by far the most common form of bootstrap. Most people mean only this when they talk about bootstrap.

Let us consider the simplest case of nonparametric bootstrap. Here $\mathbf{X}$ consists of $X_1, ..., X_n$ iid from some unknown F. We know nothing about F. One popular choice of F^* here is F_n, the empirical distribution function, defined as follows.

$$F_n(x) = (\text{Number of } X_i's \leq x)/n$$



Empirical distribution function

Let us check if F_n satisfies the three conditions above.

Close to F ? The following results are well known about the proximity of F_n to F.

Theorem

For any number c, $F_n(c) \to F(c)$ a.e.

Theorem

For any number c,

$$F_n(c) \sim AN(F(c), F(c)(1 - F(c))/n).$$

Here AN means "asymptotically normal."

Theorem

(Glivenko-Cantelli)

$$sup\{|F_n(c) - F(c)| : c \text{ real}\} \to 0 \text{ a.e.}$$

The first two are easy to prove. The last is proved in any standard text.

Here is a S-plus session to overlay the $N(0,1)$ cdf on the empirical function based on 100 iid samples. Do it and see how close the two functions are.

SAS

```
x = do(-3,3,.1);
y = cdf('normal',x);
%lineplot(y*x);

data = rannor(J(100,1,-1));
call sort(data,1);
h = do(0.01,1,0.01);
%stairplot(h*data);
```

Completely known ? F_n is completely known, since it is just a function of the data.

Easy to simulate ? Also it is easy to simulate from, since it is just a discrete distribution with finitely many mass points. It should be easy to see that simulating from F_n is same as choosing a X_i at random. Thus, simulating k iid samples from F_n is same as choosing a simple random sample *with* replacement (SRSWR) from X_i's. In SAS use the following code. Assume that the original X_i's are stored in an array `x` of length `n`.

SAS

```
start resample(x);
  n = nrow(x);
  indices = randuni(n,1,n);
  xstar = x[indices];
  return(xstar);
finish;
```

In Matlab you may use

MATLAB

```
bootData = x(unidrnd(n,1,n));
```

4.5 How to do bootstrap using F_n?

Here the ideal bootstrap distribution is $LAW(T(\mathbf{X}, F_n), F_n)$. To understand this distribution first notice that F_n is a discrete distribution with n mass points, the masses being $1/n$ each. (We shall disregard the possibility that some sample may be repeated.) The number of samples of size n that can be drawn from this is also finite. In fact, there are n^n such equally likely samples (though many of them are identical, since the order of observations in the sample does not matter.) If we list all these n^n samples and compute $T(\mathbf{X}, F_n)$ for all these, then you get the ideal bootstrap distribution $LAW(T(\mathbf{X}, F_n), F_n)$.

Exercise: Suppose that your original sample is

$$2.3, 4.2, 1.0, 3.0, 5.5, 3.4, 2.1, 5.0.$$

Compute $LAW(T(\mathbf{X}, F_n), F_n)$ where $T(\mathbf{X}, F) = \text{mean}(\mathbf{X})$ - mean(F). You will need to write a computer program to enumerate all the possible resamples. ♡

Example: Let us solve the example about estimating mean using median at the beginning of this section. We shall take $B = 1000$ (arbitrarily). Generate B samples

$$X_1^*, ..., X_B^*,$$

from F_n. Each X_b^* is of size $n = 101$. Compute $T(X_b^*, \theta(F_n))$ for $b = 1, ..., B$. The estmated M.S.E. is then given by the average of the squares of these B numbers.

Do the following SAS session to get an idea about how well $LAW(T(\mathbf{X}, F), F)$ is approximated by the empirical bootstrap distribution. For this purpose we shall take $F = N(0, 1)$.

SAS

```
/* First the density of LAW(T(X,F),F) */
x = do(-1,1,0.1);
temp = cdf('normal',x);
y = pdf('beta',temp,51,51);

%lineplot(y*x);


/* Now the empirical bootstrap histogram */

B = 1000;
n = 101;

x = rannor(J(n,1,-1)); /* Here's our original data */
avg = mean(data);

do i = 1 to B;
  xstar = resample(x);
  T(i) = median(xstar) - avg;
end;


%hist(T);

print(mean(T##2));    /* Print the M.S.E. */
```

And here is the Matlab version.

MATLAB

```
% First the density of LAW(T(X,F),F)

x = -1:0.1:1;
temp = normcdf(x);
y = (temp.*(1-temp)).^50.*normpdf(x)/beta(51,51);
plot(x,y);

% Now the empirical bootstrap histogram

B = 1000;
n = 101;

data = normrnd(0,1,1,n); %Here's our original data
avg = mean(data);

do i = 1 to B;
  indices = floor(n*ranuni({[n]-1}))+1;
  bootData = x[indices];
  T(i) = median(bootData) - avg;
end;

figure;
hist(T);
```

```
mean(T.^2)    % Print the M.S.E.
```

♡

4.6 Bias and standard error

While the above method can be used to estimate any quantity that is a function of $LAW(T(\mathbf{X}, F), F)$, two special cases occur far more frequently than others. These are estimations of bias and standard error of an estimator.

Suppose that we have a parameter $\theta(F)$ that we are estimating with some estimator $T(\mathbf{X})$. T is free of F in this case. To estimate bias of $T(\mathbf{X})$ by bootstrapping generate B resamples from $\mathbf{X}$:

$$X_b^*, b = 1, ..., B,$$

each X_b^* being of size n, the original sample size. Then the bootstrap estimator of bias is the avergage of

$$T(X_b^*) - \theta(F_n), b = 1, ..., B.$$

The bootstrap estimator of the variance of T is the sample variance of

$$T(X_b^*), b = 1, ..., B.$$

Here we use B (and not $B-1$) in the denominator. The square root of the variance is the bootstrap estimator of the standard error of T.
Exercise: Suppose that our data consist of

$$X_1, ..., X_{200} \text{ iid } F,$$

where F has mean θ. Both F and θ are unknown. We want to estimate θ using 5% trimmed mean. This estimator is the mean of the central 192 samples (after throwing away the smallest 4 X_i's and the largest 4 X_i's.)

Apply the bootstrapping technique to estimate the bias and standard error of this estimator. Take $B = 1000$. ♡
Exercise: Let V_{resamp} denote the bootstrap estimator of variance given above. Let V_{ideal} denote the variance of $LAW(T(\mathbf{X}), F)$. Show that

$$E(V_{boot}|\mathbf{X}) = V_{ideal}.$$

Here $\mathbf{X}$ denotes the original sample. ♡
Exercise: Let V_{resamp} and V_{ideal} be as above. Show that

$$Var(V_{boot}) \geq Var(V_{ideal}).$$

♡
Exercise: Given a data set consisting of n values all of which are distinct, what is the total number of distinct bootstrap resamples possible? Compute this number for $n = 15$. ♡
Exercise: Given a data set consisting of n distinct values

$$x_1, ..., x_n,$$

what is the probability of drawing a bootstrap resample consisting of n_i copies of x_i for $i = 1, ..., n$, where the sum of n_i's is 1? ♡

5 Bootstrap Confidence intervals

`http://192.168.50.201/statcomp/resample/boot2.html`

5.1 Introduction

There are a number of approaches for finding confidence intervals using bootstrap. We shall discuss three approaches which are listed below in increasing order of sophistication.

1. Bootstrap-t
2. Percentile confidence intervals
3. Bias-Corrected and Accelerated (BCa) confidence intervals

But first let us review the concept of confidence intervals.

5.2 Confidence intervals

Suppose we have data

$$X_1, ..., X_n \text{ iid } F_\theta,$$

where θ is the parameter of interest. We want to get a confidence interval for θ of level $1 - 2\alpha$, for some given $\alpha < 0.5$. Typically, $\alpha = 0.05$ or 0.025, corresponding to 90% and 95% confidence intervals.

Defn

> A $1 - 2\alpha$ level (2-sided) confidence interval is a pair of statistic-s L and U such that
>
> $$P_\theta(L < \theta \text{ and } U > \theta) = 1 - 2\alpha \text{ for all } \theta.$$
>
> The subscript θ in P_θ is just to remind us that the probability is computed under F_θ.

We may even take $L = -\infty$ or $U = +\infty$. These will correspond to one-sided confidence intervals.

There are, in general, infinitely many choices for the (L, U) pair. For instance, you can move L a bit to the right, and adjust U accordingly.

5.3 Bootstrap-t

This method is applicable when we have an estimator $\hat{\theta}$ of θ and an estimator S for the standard error of $\hat{\theta}$. [5] Thus we have two statistic-s $\hat{\theta}$ and S. Let

$$T(\mathbf{X}, F) = (\hat{\theta}(\mathbf{X}) - \theta(F =))/S(\mathbf{X}).$$

Then the **ideal** bootstrap-t confidence interval of level $1 - 2\alpha$ is

$$(\hat{\theta} + t_1 S(\mathbf{X}), \hat{\theta} + t_2 S(\mathbf{X})),$$

where t_1 and t_2 are the α-th and $(1 - \alpha)$-th quantiles of $LAW(T(\mathbf{X}, F_n), F_n)$.

These quantiles are difficult to compute. So we use **empirical** bootstrap-t confidence intervals as an approximation. For this we generate bootstrap resamples $X_1^*, ..., X_B^*$. For each of these B resamples compute T to get B numbers:

$$T_b^* = T(X_b^*, F_n). \text{ for } b = 1, ..., B.$$

[5] It is possible to apply the method even when such a formula is not available, but it requires a nested bootstrap loop (see later).

Let t_1^* and t_2^* be the appropriate sample quantiles of these B numbers. Then the empirical bootstrap-t confidence interval is then

$$(\hat{\theta}(\mathbf{X}) + t_1^* S(\mathbf{X}), \hat{\theta}(\mathbf{X}) + t_2^* S(\mathbf{X})).$$

Example: Let us obtain 95% confidence interval for the population mean based on a sample of size 100. Here θ is population mean. We shall take $\hat{\theta} = \bar{X}$. An estimator of its standard error is

$$\sqrt{\frac{\sum (X_i - \bar{X})^2/(n-1)}{n}}$$

which we shall use as S. Then T is the usual t-statistic. However, since we are not assuming normality, it need not have t-distribution, and so we cannot use the t-table to find the cut-off points. In bootstrap-t we generate, say, $B = 500$ replicates. Each replicate is a resample of size 100 from the original sample. Compute Z for each of the replicates. Then find t_1 and t_2 as 0.025-th and 0.975-th quantiles of these 500 numbers.

SAS

```
data = ranexp(J(100,1,-1))/5;
thetaHat = mean(data);

Z = J(500,1,0);  /* Declare Z as a 500 x 1 matrix with all zero entries.
do b = 1 to 500;
  bootData = resample(data);
  S = sqrt(var(bootData)/100);
  T = mean(bootData);
  Z[b] = (T-thetaHat)/S;
end;
L = quantile(Z,0.025);
R = quantile(Z,0.975);
print(L,R);
```

♡

The usual t-test requires the fact that

$$\frac{\sqrt{n}(\bar{X} - \mu)}{\sqrt{\sum (X_i - \bar{X})^2/(n-1)}}$$

has a distribution free of μ and σ^2. But in absence of normality this need not be case any more. However, still assumption the bootstrap-t confidence interval is of $1 - 2\alpha$ level (approximately). To see this argue as follows.

In the usual t-test the t-cutoff points are fixed constants not depending on the data. This is possible because the distribution of the pivotal quantity

$$\frac{\sqrt{n}(\bar{X} - \mu)}{\sqrt{\sum (X_i - \bar{X})^2/(n-1)}}$$

is parameter-free. In bootstrap-t this distribution depends on μ and σ^2. To adjust for this we let the cut-off points t_1 and t_2 depend on the data. If the underlying distribution is skewed, for instance, then the data will be more "one-sided", causing t_1 and t_2 to move asymetrically maintaining the confidence level at $1 - 2\alpha$.

Exercise: The following numbers are the weights (in grams) of 10 bottles output by a bottling plant:

100.3, 99.8, 103.4, 102.4, 100.5, 93.4, 98.4, 101.3, 101.2, 99.4.

Assume that these are iid with unknown mean μ. Compute the bootstrap-t confidence interval of μ at 90% level. Use sample mean to estimate μ. ♡

5.3.1 When S is not known

In this case you cannot apply the above procedure directly. But still you may estimate the standard error using bootstrapping. The idea is as follows. For each bootstrap resample $\mathbf{X}_b$ you generate N further bootstrap resamples

$$\mathbf{X}_{b1}^{**}, ..., \mathbf{X}_{bN}^{**}.$$

by resampling $\mathbf{X}_b$. Based on these you compute S.

Exercise: Do the last exercise, but using sample median instead of mean. Use resample of resamples to estimate standard error. ♡

5.4 Plug-in estimators

Before going into the next method we need to understand the so-called "plug-in" estimator of a parameter.

What is the definition of a parameter? Suppose we have some data $\mathbf{X}$ with joint distribution F. Then any (real-valued) function of F is called a parameter. This definition may strike you as odd, because we are used to considering parameters as unknown constants. But there is no contradiction of terms here as we see below.

Example: If our dataset consists of

$$X_1, ..., X_n \text{ iid } F$$

then $\mu = E(X_1)$ is actually a function of F, because if we know F then μ is also known. We express this as

$$\mu = \mu(F).$$

♡

Defn

> Suppose that we have n iid data with common distribution F. We also have a parameter $\theta = \theta(F)$. Consider the empirical distribution F_n, which is also a distribution function. Then the quantity
>
> $$\hat{\theta}_n = \theta(F_n)$$
>
> is called the **plug-in estimator** of θ.

Observe that since F_n is a function of the data, so is $\hat{\theta}_n$. Many well-known estimators are actually plug-in estimators.

Example: Suppose that θ is the mean. So for a distribution F,

$$\theta(F) = \text{mean of } F.$$

Hence $\hat{\theta}_n$ = mean of F_n = sample mean. ♡

Exercise: What is the plug-in estimator of variance? ♡

5.5 Percentile interval

This is somewhat like bootstrap-t, except that it does not require us to know or estimate S. But here we must work with the plugin estimator $\hat{\theta}$. Let r_1 and r_2 be the α-th and $(1-\alpha)$-th quantiles of $LAW(\hat{\theta}, F_n)$. Then the **ideal** bootstrap percentile confidence interval of level $1-2\alpha$ is

$$(r_1, r_2).$$

To avoid computing the quantiles, we can use **empirical** bootstrap percentile conficence interval as follows. We start by generating B bootstrap resamples and compute $\hat{\theta}$ for each of them:

$$\hat{\theta}_1^*, ..., \hat{\theta}_B^*$$

Let r_1^* and r_2^* be the appropriate sample quantiles of these. Then the empirical bootstrap percentile confidence interval is (r_1^*, r_2^*).

5.5.1 When does the percentile method work nicely?

Suppose first that $\hat{\theta}$ is $N(\theta, \sigma)$, where σ is known. Then, of course, the best $1 - 2\alpha$ confidence interval for θ is

$$(\hat{\theta} - z\sigma, \hat{\theta} + z\sigma),$$

where $\Phi(z) = 1 - \alpha$. Now suppose that $\hat{\theta}$ is not normal, but it is possible to transform it to normal by some monotone transform, g. In other words,

$$g(\hat{\theta}) \sim N(g(\theta), \sigma),$$

for some known σ. Then a nice confidence interval for θ would be

$$(g^{-1}(\hat{\theta} - z\sigma), g^{-1}(\hat{\theta} + z\sigma)).$$

Apparently, to use this you need to know such a transform g. Efron and Tibshirani have shown (details in the main textbook) that if such a g exists, then the ideal percentile confidence interval (which does not require explicit knowledge about g) is close to the above nice interval.
Exercise: Show this. Assume that the orginal sample size is large enough, so that the sample quantiles are good approximations to population quantiles. ♡
Example: Suppose we have iid bivariate data (X_i, Y_i) from some distribution F. Consider obtaining a 90% confidence interval for θ, the population correlation coefficient.

We shall apply bootstrap percentile method. First generate the original data from bivariate normal with correlation 0.5, means 0 and variances 1. Now draw 1000 bootstrap resamples and compute the correlation coefficients. Finally, take the 5% and 95% sample quantiles of these 1000 correlations.

Notice that the bootstrap method did not use the fact that our data was actually bivariate normal. However, for bivariate normal there is a result that approximately

$$g(\hat{\theta}) \sim N(g(\theta), 1/(n-3)),$$

where g is Fisher's z-transform

$$g(x) = 0.5log((1+x)/(1-x)).$$

So the best interval would be

$$(g^{-1}(\hat{\theta} - z\sigma), g^{-1}(\hat{\theta} + z\sigma)),$$

where where $\Phi(z) = 95\%$ and $\sigma^2 = 1/(n-3)$. ♡
Exercise: Compare this with the bootstrap percentile confidence interval. ♡
Exercise: Show that the ideal bootstrap percentile confidence interval is monotone-transformation-preserving. This means that if g is any monomtone increasing transform, and (L, U) is a bootstrap percentile confidence interval of some level for some parameter θ, then the bootstrap percentile confidence interval of $g(\theta)$ is $(g(L), g(U))$. ♡

5.6 BCa method

This is a sophisticated version of percentile interval. In percentile interval we chose equal tails (eg for $1-2\alpha$ confidence interval each tail had probability α). In BCa these are chosen asymmetrically.

The main motivation behind this asymmetric choice comes from situations where a function g as above does not exist. In many such situations, however, we may still have

> a monotonically increasing function g such that $g(\hat{\theta})$ is approximately $N(g(\theta)-b\sigma,\sigma^2)$ where $\sigma = 1+ag(\theta)$. Here a, b are constants.

Note that we are allowing $g(\hat{\theta})$ to be biased (the constant b controls the bias) and the standard error to change linearly (slope a) with $g(\theta)$. If $a = b = 0$, we are back to the percentile interval situation. Here b is called the **bias constant** and a is called the **acceleration constant**. BCa stands for **Bias Corrected and Accelerated** method, and is a method that incorporates the effects of a and b.

In a given situation how do you know if such a transformation exists? The book by Efron and Tibshirani claims that there are many cases where such transformations exist. Anyway, the above result shows that BCa is a generalisation of the percentile method.

If a, b are known then we proceed as follows. Suppose we need a confidence interval of level $1-2\alpha$. In the BCa confidence interval the lower tail has probability

$$\begin{aligned}\alpha_1 &= \Phi(b+(b+z)/(1-a(b+z)))\\ \alpha_2 &= \Phi(b+(b-z)/(1-a(b-z))),\end{aligned}$$

where $\Phi(z)=\alpha$, Φ being the $N(0,1)$ cdf.

In practice, of course, the two numbers a and b are rarely known. So we need to estimate them. Estimating b is actually quite easy:

$$\begin{aligned}P(\hat{\theta}<\theta) &= P(g(\hat{\theta})<g(\theta))\\ &= P(g(\hat{\theta})-g(\theta)<0)\\ &= P(N(b\sigma,\sigma^2)<0)\\ &= P(N(b,1)<0)\\ &= P(N(0,1)<-b)\\ &= \Phi(-b)\end{aligned}$$

Thus, we have

$$b = -\Phi^{-1}P(\hat{\theta}<\theta)$$

By the bootstrapping principle, we may estimate $P(\hat{\theta}<\theta)$ by the fraction of $\hat{\theta}_b^*$'s exceeding $\hat{\theta}(\mathbf{X})$.

There are various ways to estimate a. One way is given in Eqn 14.15 in the main textbook.

Exercise: Solve the last exercise using the BCa method. ♡

6 More general bootstrap

`http://192.168.50.201/statcomp/resample/boot3.html`

6.1 Bootstrapping residuals

We shall discuss the problem of bootstrapping in a linear model set up.
Example: Let us consider a linear model of the form

$$Y_{ij} = \mu_i + \epsilon_{ij},$$

where $i = 1, ..., I$ and $j = 1, ..., J$. We assume that ϵ_{ij}'s are iid F. Note that the Y_{ij}'s are not identically distributed. The ϵ_{ij}'s are, but we are observing only the Y's and not the ϵ's. ♡

The question we are interested in is "How to resample from the Y's?" Just taking an SRSWR form the Y's as before is not justified because the Y's are not identically distributed. Here we **bootstrap the residuals.** The algorithm is given below.

1. Estimate μ_i's. You may do this by any reasonable method, e.g., least squares. Let the estimates be called $\hat{\mu}_i$'s.
2. Compute the residuals: $\hat{\epsilon}_{ij} = Y_{ij} - \hat{\mu}_i$.
3. Resample the residuals, ie draw a resample from the $\hat{\epsilon}_{ij}$'s. Let $\hat{\epsilon}^*_{ij}$'s denote a resample of size IJ. Here we are pretending as if the $\hat{\epsilon}$'s are iid.
4. Add the resampled residuals back to get the bootstrap data:

$$y^*_{ij} = \hat{\mu}_i + \hat{\epsilon}^*_{ij}$$

This procedure is quite general. Whenever we have any model with iid residuals, we can generate resamples as follows. Fit a model by any reasonable method (ie, estimate the parameters of the model), get the residuals using the data and the fitted model, resample the residuals, add back to the fitted model.

6.2 Using the computer

The crucial step in bootstrapping residuals is to obtain the residuals in the first place. Since this depends on the model and estimation procedure, there is no ready made function in any software to do residual-bootstrapping. We have to write our program for this. Let us solve the problem using least square method in SAS.

First we read the data. The data file data.dat contains two columns of data. The first column holds the Y's and the second column holds the i's.

SAS

```
data from_file;
  infile 'data.dat';
  input y i;
run;
```

Next do the fitting. For this we use `proc glm`.

SAS

```
proc glm;
  class i;
  model y=i;
  output out=from_glm r=yresid p=yhat;
run;
```

Now go into `proc iml` and work with it.

SAS

```
proc iml;
use from_glm;
read all var _ALL_;
close from_glm;

do b = 1 to 1000;
  rStar = resample(yresid);
  yStar = p + rStar;
end;
```

7 Jackknife

http://192.168.50.201/statcomp/mcmc/jk.html

7.1 Introduction

Jackknife is very much like CV where the test set is of size 1 and the training set is of size $n-1$. There are only n ways of partitioning a sample of size n into test and training sets of these sizes. So jackknife does not use any random sampling. Rather it averages over *all* the n possible partitions. Jackknife was originally introduced to estimate bias and variance of an estimator.

7.2 Jackknife estimate of bias

Set up: We have iid data $X_1, ..., X_n$. We have an estimator T to estimate a parameter θ. We want to estimate the bias of T.

Jackknife procedure: Call the entire sample as $\mathbf{X}$. Define the *i-deleted sample*, $\mathbf{X}_{-i}$ as

$$\mathbf{X}_{-i} = (X_1, ..., X_{i-1}, X_{i+1}, ..., X_n)$$

This is our training set when the test set is (X_i). Now compute

$$T_{-i} = T(\mathbf{X}_{-i})$$

and

$$T_{(.)} = \sum T(\mathbf{X}_{-i})/n$$

Then the jackknife bias estimator is defined as

$$bias_{jack}(T) = (n-1)(T_{(.)} - T(\mathbf{X}))$$

The Jackknife bias-corrected estimator is defined as

$$T - bias_{jack}(T).$$

Example: Suppose we have data $\mathbf{X} = (2,5,2,3)$. Thus $n = 4$ here. T is the sample mean.

$$\begin{aligned}
\mathbf{X}_{-1} &= (5,2,3) T_{-1} = 3.33 \\
\mathbf{X}_{-2} &= (2,2,3) T_{-2} = 2.33 \\
\mathbf{X}_{-3} &= (2,5,3) T_{-3} = 3.33 \\
\mathbf{X}_{-4} &= (2,5,2) T_{-4} = 3.00
\end{aligned}$$

So

$$T_{(.)} = (3.33 + 2.33 + 3.33 + 3.00)/4 = 3.$$

Also,

$$T(\mathbf{X}) = (2 + 5 + 2 + 3)/4 = 3.$$

Hence

$$bias_{jack}(T) = (4 - 1)(T_{(.)} - T(X)) = 0.$$

♡

7.3 Jackknife estimate of standard error

Using same set up and notation as above

$$V_{jack}(T) = (n - 1)/n \sum(T_{-i} - T_{(.)})^2).$$

Its square root is the jackknife estimate of standard error.
Example: Continuing with the last example,

$$V_{jack}(T) = ((4-1)/4)[(3.33-4)^2+(2.33-4^2)+(3.33-4^2)+(3.00-4)^2] = 3.5150.$$

♡

7.4 Using the computer

Jackknifing is easy with both SAS and Matlab. Assume that the data is stored in a column vector called `x`. You already have a function `T` to compute the estimator.

SAS has a function called `remove` to delete elements from an array. If `x={1,2,3,4,3,2}` then `remove(x,{5}) = {1,2,3,4,2}`. This makes it easy to create the deleted samples.

SAS

```
n = nrow(x);
val = J(n,1,0);
do i = 1 to n;
  val[i]  = T(remove(x,{i}));
end;

biasJack = (n-1)*(mean(val)-T(x));
print(biasJack);
```

Exercise: Write down a SAS program to compute the jackknife estimate of standard error. ♡

7.5 Justification of the jackknife estimator of variance

Jackknife, like any other resampling method, tries to "simulate" many datasets based on only one dataset (the original one). To understand any resampling method, therefore, it is best to start with the situation where we really can have many independent datasets. If we have many datasets and we want to estimate variance of a statistic T, then we shall just compute T for each of the N datasets to get N numbers, say, $T_1, ..., T_n$. Then we can just compute the variance of these as

$$V_{simul}(T) = \sum(T_i - \bar{T})^2)/n,$$

where $\bar{T}$ is the mean of the T_i's. Now compare this with

$$V_{jack}(T) = \sum (T_{-i} - T_{(.)})^2 (n-1)/n$$

You can see that in the jackknife formula $\mathbf{X}_{-i}$'s are playing the role of the fresh datasets. So T_{-i}'s are used in place of T_i's, and $T_{(.)}$ in place of $TBAR$.

The only thing that is still not explained is the choice of the constant. Why are we using $(n-1)/n$ and not $1/n$ or $1/(n-1)$? The answer lies in the following exercise.

Exercise: Suppose that we want to estimate the variance of a statistic T by the following formula:

$$C \sum (T_{-i} - T_{(.)})^2$$

and we want to choose the constant C to make this estimator unbiased if T is the sample mean. Then show that $C = (n-1)/n$. ♡

7.6 Justification of the jackknife estimator of bias

A very similar interpretation exists for the jackknife estimate of bias. The constant is chosen to make the bias match the exact value for sample variance.

However, note one curious point. If $T(X)$ is a constant function, then the jackknife estimator of its bias is zero! This is natural since in jackknife estimator of bias uses $T(X)$ as a proxy for θ. So if this proxy is bad, the estimator is bad, as well. However, if T is "good" then this estimator is also "good". The following result makes this precise.

Theorem

If

$$E(T(X) = \theta + b/n,$$

then

$$E(\text{Jackknife estimator of bias}) = b/n,$$

and hence

$$E(\text{Jackknife bias-corrected } T) = \theta.$$

Proof:

Direct computation. ♡

Thus, if T is already first-order unbiased (ie, bias term being b/n which is $O(1/n)$) then jackknife bias-corrected T is second-order unbiased (ie, bias term is $O(1/n^2)$).

7.7 Which is better, jackknife or bootstrap?

Jackknife is certainly easier to compute, but it uses less information. It may be viewed as an approximation to bootstrap. We shall explore this in an assignment.

The main advantage of the bootstrap is its generality. One can do bootstrapping in a wide variety of set up. But it is quite difficult to generalise the notion of jackknifing beyond estimation of bias and variance. In fact, with bootstrap available, one would hardly consider doing jackknife.

Jackknife fails if the statistic is not "smooth". The median, for instance, is not smooth. Jackknife estimator of its variance is *not* consistent. However, the bootstrap estimator is.

8 Cross-validation

`http://192.168.50.201/statcomp/resample/cv.html`

8.1 Supervised and unsupervised data

In many statistical problems we are given observations about some random variable X, and we are interested in knowing the value of some other random variable Y. Two important classes that come under this set up are regression and classification. In regression we try to predict Y based on some (estimated) function of X. In classification problem X consists of some observed features about an individual (e.g., clinical measurements) and Y is the class where the individual belongs (e.g., type of disease.)

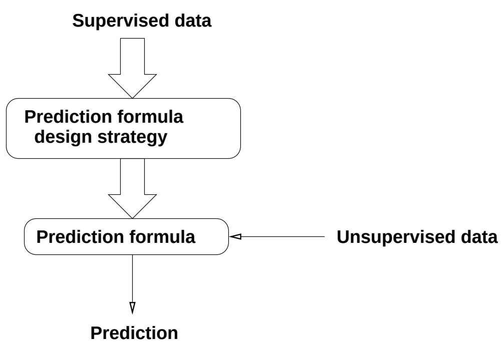
In a typical application we can observe only X but not Y. Such a data set where only X is known but not Y is called an unsupervised data. To develop a method to predict Y from X the statistician must start with some initial data where both X and Y are known. Such a data is called **supervised data**. Based on this data he can design a method to predict Y from X. Once such a method has been designed, then one can apply it to unsupervised data. The idea is that getting supervised data is possible but expensive.

Example: A software that tries to read handwritings gets inputs like the letters shown left.

Unsupervised and supervised handwriting data

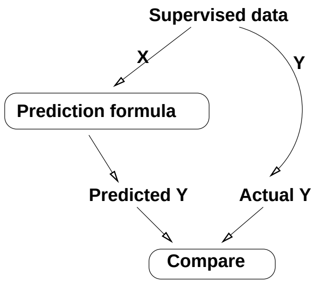
The software is supposed to recognise what these letters are. While developing the software someone had to tell the software the names of letters. The letters by themselves is the unsupervised data. The letters together with their names constitute the supervised data. Getting this supervised data is expensive because someone had to be paid to decipher the handwriting and enter the names of the letters in the computer. ♡

Example: In medical science the disease of a patient is diagnosed from various clinical measurements. To learn to do this a medical student has to be familiar with history of similar cases. The clinical measurements alone form the unsupervised data. The case histories provide supervised data consisting of both measurments for past patients as well as the true nature of their diseases (possibly obtained by post mortem analysis.) ♡



Using supervised data to make prediction formula

One can use supervised data to design a method to tackle unsupervised data later. Also one can use supervised data to assess the performance of such a method. To assess the performance of a method we first take a supervised data set consisting of (X, Y) observations. We apply the method under test to the X-part. The method predicts the Y-values as $\hat{y}$, say. Now we can compare the true Y with the predicted $\hat{y}$ to see how well the method has performed. In the handwriting recognition example we can count the number of letters the program has failed to recognise correctly.



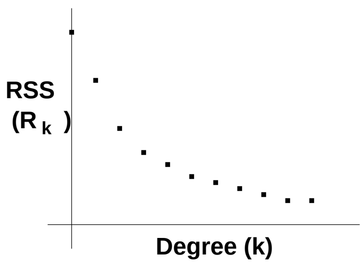
Using supervised data to assess a prediction formula

Thus, supervised data have two uses. The following analogy may help you understand these two roles of supervised data. Consider a teacher teaching a class of students. The teacher has a bank of questions that he knows how to answer. He presents some of these as solved examples in the class. This is the first use. He reserves the remaining questions for the final examination. This is the second use.

What happens if the teacher uses the same set of questions for both the purposes, *i.e.,* the questions in the exam are from the solved examples? Then a student who has crammed without understanding anything will perform quite well. Such a student can reproduce solved examples nicely, but is very bad at solving similar new problems. In statistics this is called **overfitting**. To prevent such a student from getting high grade it is important to make the exam problems different than the examples solved in class. This is the idea behind cross validation: train the students using one set of questions (the *training set*) and test the students on a different set of similar questions (the *test set.*)

8.2 An example: Polynomial fitting

Consider fitting a polynomial to data of the form (X_i, Y_i) for $i = 1, ..., n$. We shall use least square method for this purpose. Let the residual sum of squares (RSS) for the best k-degree polynomial be called R_k. Then R_k is a decreasing function of k(why?) A typical plot is shown below.



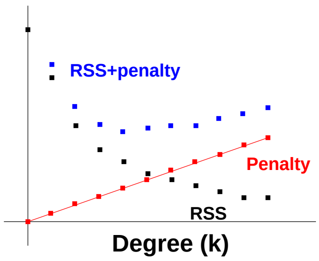
RSS decreasing with degree

From the plot we see that though R_k is decreasing monotonically, the rate of decrease is decaying off. Thus, it is not a good idea to increase k too much. We would prefer a low degree polynomial with small RSS than a high degree polynomial with slightly lower RSS. Thus, we have a trade-off between degree and RSS. We want both to be low. But reducing one tends to increase the other. One way to tackle such a trade off is by introducing a **penalty** for high degree polynomials. Instead of minimizing RSS (as we do in the least squares method) we shall minimize

$$RSS + \alpha k.$$

Here α is some positive constant that we choose. It controls the relative importance of RSS and degree. If α is large then degree is the more serious issue: better a low degree polynomial than a low RSS. If α is small, then RSS is the more important concern: we shall be happy with even a high degree polynomial if reduces RSS.

First let us see how to fit a polynomial that minimizes $RSS+\alpha k$ for a given α. For this we fit polynomials of different degrees by ordinary least squares as before to get R_k's. Now plot $R_k + \alpha k$.



$\underline{R_k + \alpha k}$

Choose k that minimizes this.

Exercise: Define, for $k = 0, 1, 2, ...$

$$a_k = R_k - R_{k+1}.$$

Show that for any α in the interval (a_{t-1}, a_t) the polynomial that minimizes $RSS + \alpha k$ has degree t. ♡

Now that we know how to fit a polynomial for any given α the next question is what α to use. There are infinitely many possible values for α. Rather than comparing all these values, we shall choose a grid of α-values:

$$\alpha_1, \alpha_2, ...\alpha_m,$$

and determine which α_k is the most suitable for us. We can use **5-fold cross validation** for this purpose as follows. We randomly partition the supervised dataset into 5 nearly equal parts. Leave the first part aside, and fit the polynomial to only the data consisting of the remaining 4 parts. This produces a sequence of best fitting polynomial P_k when $\alpha = \alpha_k$. Apply P_k to part 1 of the data and denote the RSS by S_{1k}.

Repeat the same process, this time leaving out part 2 of the data. This produces S_{5k} 's. Similarly, get S_{ik}'s for $i = 1, ..., 5$. Observe that the 5 numbers $S_{1k}, ..., S_{5k}$ all reflect the performance of α_k. We average these numbers to get the **cross validation score** for α_k which we call CV_k. The k for which this is minimum determines which α_k is the best.

8.3 p-fold CV

Here we first split the data randomly into p equal parts (or almost equal parts if the sample size is not divisible by p). Let the parts be called $X_1, ..., X_p$. Use X_t as the test set, and the rest of X as the training set. Let us denote by X_{-t} this remaining part of X. Then we apply the design strategy on X_{-t} to get a prediction formula f_{-t}.

Apply this to the test set, ie, X_t to get the predicted value $f_{-t}(X_t)$. Finally, we compare this prediction with the actual Y_t, (which we know because it is part of the supervised data). This comparison is done using a loss function

$$L(Y_t, f_{-t}(X_t)).$$

Do this as t ranges from 1 to p, and then average to get the CV score for MS_i :

$$CV_i = \sum_t (L(X_t, M_{it}))/p$$

Now choose the modelling strategy that minimises this. However, observe that CV_i's are themselves random quantities. So minor differences between them may very well be due to chance rather than any real performance difference among the corresponding modelling strategies.

8.4 leave-one-out CV

This is just like p-fold CV with p=n, the sample size. Thus, here we leave one datapoint out, fit a model and check the performance of the model on that left-out datapoint. We do it separately for each of the n datapoints, and then average. This has the advantage that it involves no randomness, so may sometimes be compute analytically. But a test data of size 1 is often judged inadequate.

A Introduction to SAS

`http://192.168.50.201/statcomp/resample/sasintro.html`

A.1 Getting started

We shall use SAS 9.1.3 for Unix.

First log on to the Stat-Math Sun machine using username stud¡n¿ and password stud¡n¿, where ¡n¿ is 1,...,6.

After logging in you will find a console window. (If you do not see it, then click on the computer icon on the taskbar to open one.) Type `sas &` in this window and hit "Enter". After some time SAS will start opening many windows. Each window has a different name written on top. Minimise all of them except 3:

1. The program editor window
2. The output window
3. The log window

A.2 The first session

Type the following in the program editor window.

SAS

```
proc iml;
x = 29;
print(x);
```

Now hit "End" to submit these three lines to SAS. You will see some messages in the log window, and the output window will display the value 29.

Look at the program editor window. The lines you wrote in it have been automatically removed after you submitted them. Now type

SAS

```
y = {4,5,6};
print(y);
```

in the program editor window and submit (i.e., hit "End").

The program editor window will again become blank. Press "F4" once. The last submission will reappear. Change the `y`'s to `z`'s. Now submit.

The above steps follow the general work flow of SAS:

1. You will enter various commands in the program editor window. This may be done by direct typing, copy-n-pasting or by hitting "F4" to replay past submissions.
2. Then you will submit the commands by pressing "End". We shall not explicitly remind you to submit the commands in future.
3. SAS will display some messages in the log window (including error messages, if any.)
4. If your command has any text output it will be shown in the output window. If your command has any graphics output, a graphics window will open to show it.

A.3 PROC IML

All manipulation of data in SAS occurs in either data statements or proc statements. In our course we shall spend much time inside `proc iml;` The command

SAS

```
proc iml;
```

invokes a powerful Matlab-like language in SAS: Interactive Matrix Language (IML). It is easy to work with matrices in IML. You can create the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

by

SAS

```
A = {1 2 3, 4 5 6, 7 8 9};
```

To see the content of the matrix `A` type `print(A);`
Exercise:

1. Create a row vector with entries 1,-3, 4.5.

2. Create a column vector with the same entries.

♡

In SAS there is no provision for making "1-dimensional arrays." If you need such an array, then you have to make it as either a column vector or a row vector. SAS considers column vectors and row vectors differently. When you store a list of values of a random variables in a vector, you should always use a column vector and not a row vector.

A.4 Subscripts

The `(i,j)`-th entry of a matrix `A` is accessed by `A[i,j]`. Rows and columns are numbered starting from 1 (not 0 as in C, C++ or Java.) If `v` is a row vector then `v[i]` means `v[1,i]`. Similarly, it means `v[i,1]` if `v` is a column vector. To find the size of a matrix `A` you can use `nrow(A)` and `ncol(A)`.

A.5 Special matrices

It is easy to make certain special matrices in SAS. Try the following commands to understand what they do.

SAS

```
b = 1:10;
c = do(0,4,0.3);
```

To create a 3 by 2 matrix with all -1's, say you can type

SAS

```
A = J(3,2,-1);
```

This is useful for initialising matrices.

A.6 Random matrices

In our course we shall need to create matrices whose entries are iid random variables. To generate a $N(0,1)$ random number use

SAS

```
X = rannor(-1);
```

The `-1` tells SAS to create a seed automatically. To generate a 2 by 3 matrix with entries iid $N(0,1)$ use the same command, replacing the -1 with a 2 by 3 matrix with each entry -1:

SAS

```
X = rannor(J(2,3,-1)); print(X);
```

Replay and submit the same line a number of times to see how `X` changes.

A.7 Working with matrices

Copy-n-paste the following two matrices to the program editor window.

SAS

```
A = {1 2, 3 4};
B = {3 -1, 0 2};
print(A+B);
print(A-B);
print(A*B);
print(A#B);
```

Can you explain the last command? To take transpose use the ‘ symbol. It is the key situated new the top left corner of your keyboard, just to the left of 1.

```
print(A‘);
```
SAS

Be careful here! The ‘ character *looks* like ’ in the program editor window. But they *mean* different things in SAS.

It is sometimes useful to *concatenate* matrices horizontally or vertically.

```
C = A || B;
print(C);
print(A//B);
```
SAS

Most math functions operate entrywise on matrices:

```
print(sin(A));
```
SAS

A.8 Submatrices

If you keep only some specific rows and columns of a matrix and delete the rest, you get a submatrix. In SAS it is easy to extract submatrices. First let us make a random 5 by 5 matrix:

```
A = ranuni(J(5,5,-1));
print(A);
```
SAS

Now we want to extract the submatrix consisting of rows 1 and 3, and columns 2, 4 and 5 of `A`.

```
B = A[{1 3},{2 4 5}];
print(B);
```
SAS

If you want to select only some rows (and keep all the columns) then:

```
C = A[{1 3},];
print(C);
```
SAS

Similarly,

```
D = A[,{1 3}];
print(D);
```
SAS

Here we choose submatrices by explicitly listing the row and column indices. There is another way to extract submatrices. Let us draw a sample of size 10 from $Uniform(0,1)$.

```
X = ranuni(J(10,1,-1));
print(X);
```
SAS

We want extract the submatrix consisting of only those observations that exceed 0.5:

```
Y = X[loc(X > 0.5)];
print(Y);
```
SAS

The `loc` function finds the locations where `X > 0.5`. Suppose that you want to make all values exceeding 0.5 equal to 0.5. You can do this as

```
X[loc(X > 0.5)] = 0.5;
print(X);
```
SAS

A.9 Summation and counting

Summing the entries of a vector is easy in IML:

SAS

```
x = 1:4;
print(sum(x));
print(x[+]);
```

Summing over the rows (or columns) of matrix is equally easy:

SAS

```
A = {1 2 3, 4 5 6};
print(A[+,]);
print(A[,+]);
```

Exercise: What do the following lines achieve?

SAS

```
temp = A[+,];
print(temp[+]);
```

♡

The summation command is very useful for counting the number of entries in a vector satisfying some property.

SAS

```
x = ranuni(J(10,1,-1));
```

We want to find how many entries of x are less than 0.3.

SAS

```
y = (x < 0.3);
print(y);
print(sum(y));
```

More often we shall write this briefly as

SAS

```
print(sum(x < 0.3));
```

A.10 Sorting

We shall sometimes need to sort the columns of a matrix by one of its columns. Consider:

SAS

```
A = {1 2,
    -1 4,
     9 3};
print(A);
call sort(A,2);
print(A);
```

Here the rows are rearranged so that the second column of A has increasing order.

Exercise: Sort the numbers 1,-3,4,3,56,-98 using SAS. ♡

A.11 Reading data from a file

It is possible to create a SAS data set by reading numbers from a file. Create a file called `mydata.dat`, say, in your home directory and type the following data in it:

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

[You may use the gedit editor for this purpose. To use it go to the console window and type `gedit mydata.dat &` and hit "Enter". This will open the gedit window. Type the numbers and click on "Save".]

To read these numbers into a data set called, say, `ddd`, type the following commands in the program editor window.

SAS

```
data ddd;
infile 'mydata.dat';
read x y;
run;
```

This creates a SAS data set `ddd` which consists of two variables `x` and `y`. The first column in the file goes inside `x`, the second inside `y`.

To use these from inside PROC IML you will need the `use` command:

SAS

```
proc iml;
use ddd;
read all into datamat;
close ddd;
```

Now `datamat` is a 2 by 4 matrix. You could also use any other name in place of `datamat`.

A.12 Programs in external files

It is possible to write commands in an external file and later submit the file to SAS. For this create a file called `myfun.sas` (or any other name, ending in `.sas`) and write the following lines in it. You may use the "gedit" editor for this.

SAS

```
print('This is inside a file.');
u = {1 2 3 4};
```

Save the file and type the following line in the SAS program editor:

SAS

```
%include 'myfun.sas';
```

Submit the command. You will see that the two lines inside the file are executed as if you have typed them in the program editor. If you now type `print(u);` the vector `1 2 3 4` will be printed in the output window.

A.13 Plots

In this course we shall need four types of plots: scatter plots, line plots, histograms and staircase plots. First let use make two vectors:

SAS
```
x = do(-6,6,0.1);
y = sin(x);
z = cos(x);
```

We want to plot y and z against x. Plotting is somewhat troublesome in SAS, so in this class we shall bury the details inside a file called `plotter.sas`. Issue the command

SAS
```
%include 'plotter.sas';
```

to load some useful plotting commands. Now type

SAS
```
%lineplot(y*x);
```

Notice the syntax `y*x`: this means "plot y against x." Once you submit this command, a graphics window will appear containing the plot. You may overlay one plot on another.

SAS
```
%lineplot(y*x z*x /overlay);
```

You can also make scatter plots:

SAS
```
%scatplot(y*x);
```

We shall need staircase plots for empirical distribution function. Suppose we have the sample 1, 7, 3, 5, 4. We can draw the empirical distribution function as

SAS
```
x = {1, 7, 3, 5, 4};
call sort(x,1); /* Sort x*/
y = do(1/5,1,1/5);
%stairplot(y*x);
```

A.14 Do-loops and if-blocks

To repeat some commands many times you can put the commands inside a `do` loop:

SAS
```
do i = 1 to 10;
  print(i);
end;
```

To take decisions you can use the if-then-else technique.

SAS
```
do i = 1 to 10;
  if i < 5 then
    print('Small');
  else
    print('Large');
end;
```

A.15 Modules

Most programming languages have a concept called "function" or "subroutine". It is a bunch of commands referred to by a single name. In SAS these are called modules. Modules are valid only inside PROC IML. Let us write a module to compute mean of a column vector `x`. For this open a file called `statutil.sas` and type:

SAS

```
start mean(x);
  val = sum(x)/nrow(x);
  return(val);
finish;
```

Now go to the SAS program editor window and type:

SAS

```
%include 'statutil.sas';
x = 1:10;
print(mean(x));
print(mean(x`));
```

Can you explain the difference in the outputs?
Exercise: Write the following modules in the file `statutil.sas`. Each should be within its own `start` and `finish`.

1. `var(x)` to compute variance of a column vector `x` using n in the denominator. Use the `##` operator to take powers of the entries of a vector.

2. `var1(x)` to compute variance of a column vector `x` using $n-1$ in the denominator.

3. `quantile(x,alpha)` to compute the `alpha`-th sample quantile of the data `x`. You will need to sort `x` and then get the `n*alpha`-th entry. SAS automatically truncates non-integer subscripts.

♡