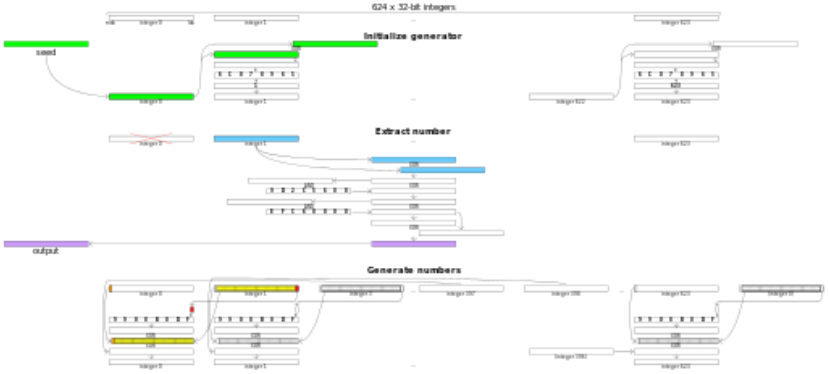
Introduction to Statistical Learning

Notes by Tim Brown and Owen Jones

Module 10: Simulation and Bayes Inference; Unsupervised Learning

Contents

Figure 1: Visualization of Mersenne-Twister Algorithm



1 Introduction

1.1 Big Picture

Where we are going?

In module 11, there will be a very quick introduction to Bayesian inference. The key idea is that beliefs prior to collecting data are expressed in a pdf or pmf for the unkown parameters. Bayes theorem then says how to update the prior beliefs in the light of the data to obtain the posterior distribution for the parameters - the distribution that combines prior beliefs and data. In some cases, the posterior distribution is a member of the same family of distributions as the prior - these are called conjugate distributions. In many realistic cases - for example generalised linear models - numerical calculation or approximation of the posterior is necessary.

Where are we going?

In module 10, two topics are sketched. One is an outline of simulation as a method of approximating probabilities and expectations. Simulation uses pseudo-random numbers which are actually deterministic but have similar properties to truly random numbers and can be repeated. After general techniques for simulation, module 11 will be covered followed by Gibbs Sampling and its application to approximating posterior distributions. The other topic is a brief introduction to unsupervised learning through principal components and clustering.

1.2 Simulating iid uniform random variables

Random numbers in R

R uses a pseudo-random number generator called the *Mersenne-Twister* to produce psuedo-random numbers $X_0, X_1, \cdots$ which behave just like independent and identically distributed uniform random variables on $(0, 1)$. Pseudo-random number generators generally repeat after a certain number have been produced - this is called the *cycle length*. The Mersenne-Twister has a cycle length of $2^{19937} - 1$.

Seeding

The number X_0 is called the seed. If you know the seed, then you can reproduce the whole sequence exactly. This is a very good idea from a scientific point of view; being able to repeat an experiment means that your results are verifiable. To generate n pseudo-random numbers in R, use `runif(n)`.

For a given value of `seed` (assumed integer), the command `set.seed(seed)` always puts you at the same point on the cycle of pseudo-random numbers. The current state of the random number generator is kept in the vector `.Random.seed`. You can save the value of `.Random.seed` and then use it to return to that point in the sequence of pseudo-random numbers. If the random number generator is not initialised before you start generating pseudo-random numbers, then R initialises it using a value taken from the system clock.

```
set.seed(42)
runif(2)

## [1] 0.9148060 0.9370754

RNG.state <- .Random.seed
runif(2)

## [1] 0.2861395 0.8304476

set.seed(42)
runif(4)

## [1] 0.9148060 0.9370754 0.2861395 0.8304476

.Random.seed <- RNG.state
runif(2)

## [1] 0.2861395 0.8304476
```

2 Discrete RVs

2.1 General method

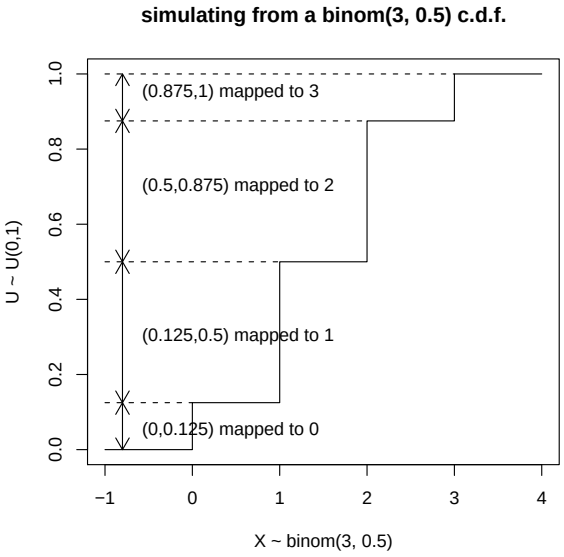
Simulating discrete random variables

Let X be a discrete random variable taking values in the set $\{0, 1, \ldots\}$ with cdf F and pmf p. The following snippet of code takes a uniform random variable U and returns a discrete random variable X with cdf F.

```
# given U ~ U(0,1)
X <- 0
while (F(X) < U) {
    X <- X + 1
}
```

When the algorithm terminates we have $F(X) \geq U$ and $F(X-1) < U$, that is $U \in (F(X-1), F(X)]$. That is,

$$\mathbb{P}(X = x) = \mathbb{P}(U \in (F(x-1), F(x)]) = F(x) - F(x-1) = p(x).$$



2.2 In R

Efficiency and common distributions

This method always works but it can inefficient. Also the cdf needs to be known or stored in a table. If parameters change a new cdf must be calculated. There are a number of tricks that speed up generation of random numbers and R has these implemented for many distributions. A good general purpose method used by R, in the `sample` command, the `alias method`. To simulate binomial, geometric, negative-binomial or Poisson rv's in R, use `rbinom`, `rgeom`, `rnbinom` or `rpois`.

`sample` in R

For simulating other (finite) discrete rv's R provides
`sample(x, size, replace = FALSE, prob = NULL)`.
The inputs are

`x` A vector giving the possible values the rv can take;

`size` How many rv's to simulate;

`replace` Set this to `TRUE` to generate an iid sample, otherwise the rv's will be conditioned to be different from each other;

`prob` A vector giving the probabilities of the values in `x`. If omitted then the values in `x` are assumed to be equally likely.

3 Continuous RVs

3.1 General method

Simulating continuous random variables

Suppose that we are given $U \sim U(0,1)$ and want to simulate a continuous rv X with (strictly increasing) cdf F_X on the range of the rv.

Put $Y = F_X^{-1}(U)$ then we have

$$F_Y(y) = \mathbb{P}(Y \leq y) = \mathbb{P}(F_X^{-1}(U) \leq y) = \mathbb{P}(U \leq F_X(y)) = F_X(y).$$

That is, Y has the same distribution as X. Thus, if we can simulate a $U(0,1)$ rv, then we can simulate any continuous rv X for which we know F_X^{-1}. This is called the *inverse transformation method* or simply the *inversion method*.

3.2 Examples

Uniform to uniform

If

$$X \sim U(1,3)$$

then

$$F_X(x) = (x-1)/2 \text{ for } x \in (1,3)$$

and thus

$$F_X^{-1}(y) = 2y + 1 \text{ for } y \in (0,1)$$

Uniform to exponential

If

$$X \sim \exp(\lambda)$$

then

$$F_X(x) = 1 - e^{-\lambda x} \text{ for } x \geq 0$$

and thus

$$F_X^{-1}(y) = -\frac{1}{\lambda}\log(1-y) \text{ for } y \in (0,1)$$

Random variable simulators in R

Distribution	R command
binomial	`rbinom`
Poisson	`rpoisson`
geometric	`rgeom`
negative binomial	`rnbinom`
uniform	`runif`
exponential	`rexp`
normal	`rnorm`
gamma	`rgamma`
beta	`rbeta`
student t	`rt`
F	`rf`
chi-squared	`rchisq`
Weibull	`rweibull`

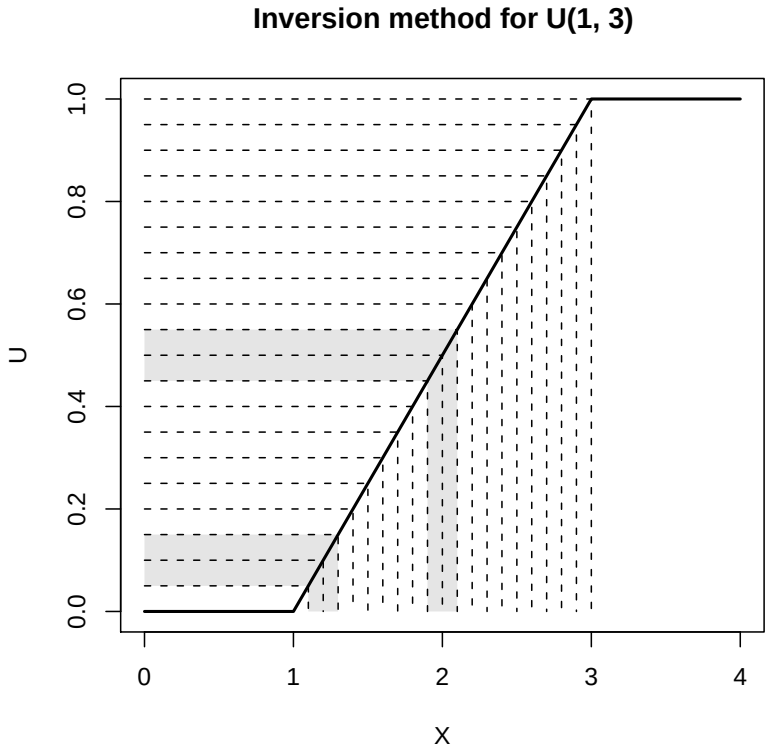


Figure 2: $X \sim U(1,3)$ - inversion from y-axis

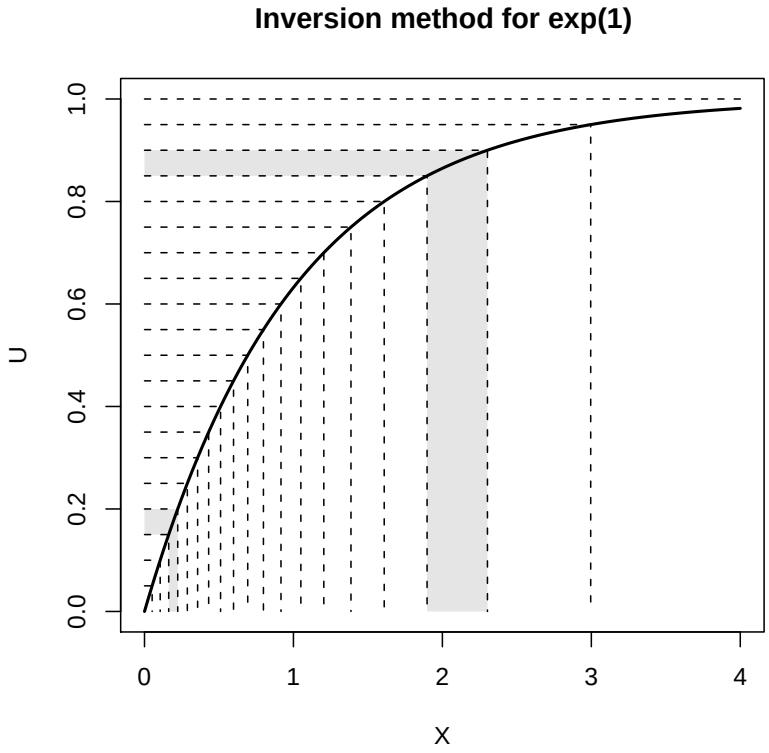


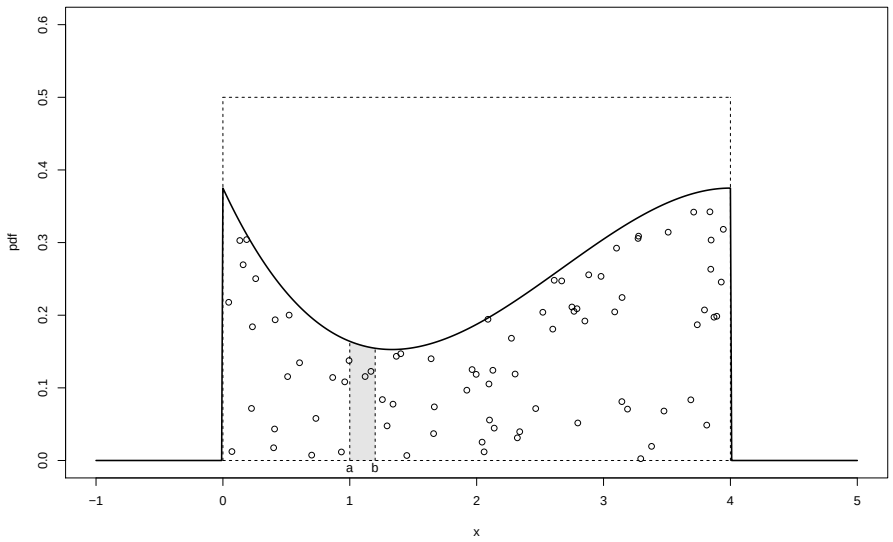
Figure 3: $Y \sim \exp(\lambda)$ - - inversion from y-axis

4 Rejection method

4.1 Definition

The rejection method

The inversion method works well if we can find F^{-1} analytically. If not, we can use root-finding techniques to invert F numerically, but this can be time-consuming. An alternative method in this situation, which is often faster, is the rejection method, often called the acceptance-rejection or AR method. Suppose that we have a continuous random variable X with pdf f_X concentrated on the interval $(0, 4)$. We imagine 'sprinkling' points $P_1, P_2, \ldots$, uniformly at random under the density function, and consider the distribution of X_1, the x coordinate of P_1.



Let R be the shaded region under f_X between a and b, then

$$\begin{aligned} P(a < X_1 < b) &= \mathbb{P}(P_1 \text{ hits R}) \\ &= \frac{\text{Area of R}}{\text{Area under density}} \\ &= \frac{\int_a^b f_X(x)dx}{1} \\ &= \int_a^b f_X(x)dx. \end{aligned}$$

So X_1 has the same distribution as X. What is an efficient, simple way to generate the P_i uniformly under f_X? The answer is to generate points with x-coordinates uniform in $[0, 4]$ and y-coordinates uniform in $[0, 0.5]$, and then *accept* only those that fall below the pdf.

Rejection method (uniform envelope) Suppose that f_X is non-zero only on $[a, b]$, and $f_X \leq k$.

1. Generate $X \sim U(a, b)$ and $Y \sim U(0, k)$ independent of X (so $P = (X, Y)$ is uniformly distributed over the rectangle $[a, b] \times [0, k]$).
2. If $Y < f_X(X)$ then return X, otherwise go back to step 1.

4.2 Example: triangular density

Consider the triangular pdf f_X defined as

$$f_X(x) = \begin{cases} x & \text{if } 0 < x < 1; \\ (2 - x) & \text{if } 1 \leq x < 2; \\ 0 & \text{otherwise.} \end{cases}$$

The rejection method can be applied as follows

```
rejectionK <- function(fx, a, b, K) {
# simulates from the pdf fx using the rejection algorithm
# assumes fx is 0 outside [a, b] and bounded by K
# note that the infinite loop is exited
# by the return statement
while (TRUE) {
x <- runif(1, a, b)
y <- runif(1, 0, K)
if (y < fx(x)) return(x)
}
}
fx<-function(x){
# triangular density
if ((0<x) && (x<1)) {
return(x)
} else if ((1<x) && (x<2)) {
return(2-x)
} else {
return(0)
}
}
```

```
# generate a sample
set.seed(21)
nreps <- 3000
Observations <- rep(0, nreps)
for(i in 1:nreps)   {
Observations[i] <- rejectionK(fx, 0, 2, 1)
}

# plot a scaled histogram of the sample and the density on top
hist(Observations, breaks = seq(0, 2, by=0.1), freq = FALSE,
ylim=c(0, 1.05), main="")
lines(c(0, 1, 2), c(0, 1, 0))
```

Figure 4: Triangular density and histogram of sample size 3000 using rejection

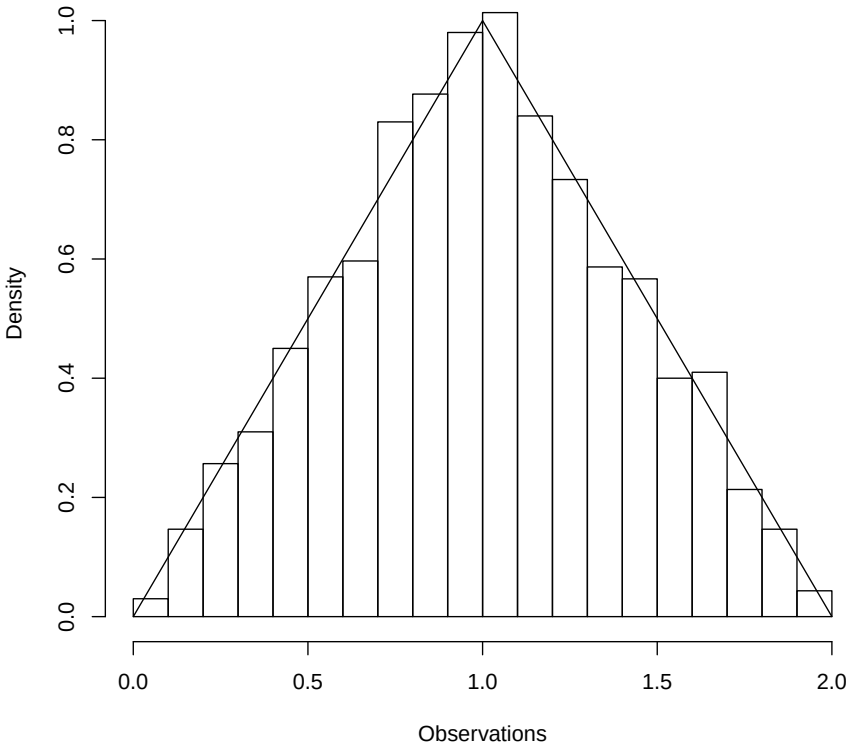


Figure 4 shows the density and the sample of size 3000.

4.3 General rejection method

General rejection method

The rejection method above uses a rectangular envelope to cover the target density f_X. What to do if f_X is unbounded? Suppose X has pdf h and let $Y \sim U(0, kh(X))$, then (X, Y) is uniformly distributed under the curve kh:

$$\begin{aligned} &P((X,Y) \in (x, x+dx) \times (y, y+dy)) \\ &\quad = P(Y \in (y, y+dy) \,|\, X \in (x, x+dx))\mathbb{P}(X \in (x, x+dx)) \\ &\quad = \frac{dy}{kh(x)} h(x)dx \\ &\quad = \frac{1}{k}\, dx\, dy. \end{aligned}$$

Rejection method for general density f_X

Let h be a density that is easy to simulate from. Choose k such that

$$k \geq k^* = \sup_x \frac{f_X(x)}{h(x)}.$$

Then kh forms an envelope for f_X, and the method above permits easy generation of points uniformly within this envelope. Accepting points below the curve f_X defines the general rejection method:

General rejection method
To simulate from the density f_X, we assume that we have envelope density h from which you can simulate, and that we have some $k < \infty$ such that $\sup_x f_X(x)/h(x) \leq k$.

1. Simulate X from h.
2. Generate $Y \sim U(0, kh(X))$.
3. If $Y < f_X(X)$ then return X, otherwise go back to step 1.

Suppose that $f_X^* \propto f_X$, $h^* \propto h$, and $f_X^*(x)/h^*(x) \leq 1$, then the general rejection method is equivalent to:

1. Simulate X from h.
2. Generate $Y \sim U(0, 1)$.
3. If $Y < f_X^*(X)/h^*(X)$ then return X, otherwise go back to step 1.

4.4 Efficiency

Efficiency

The efficiency of the rejection method is measured by the expected number of times you have to generate a candidate point (X, Y).

The area under the curve kh is k and the area under the curve f_X is 1, so the probability of accepting a candidate is $1/k$.

Thus the number of times N we have to generate a candidate point has distribution $1 + \text{geom}(1/k)$, with mean

$$\mathbb{E}N = 1 + (1 - 1/k)/(1/k) = k.$$

So, the closer h is to f_X, the smaller we can choose k, and the more efficient the algorithm.

4.5 Example: Gamma

Example: Gamma

For $m, \lambda > 0$ the $\Gamma(\lambda, m)$ density is

$$f(x) = \lambda^m x^{m-1} e^{-\lambda x}/\Gamma(m), \text{ for } x > 0,$$

There is no simple explicit formula for the cdf F or its inverse, which suggests trying the rejection method to simulate from f. Using the inversion method,

it is easy to simulate from an exponential distribution: $-\log(U)/\alpha \sim \exp$, where $U \sim U(0,1)$ and α is the rate. This suggest using an envelope based on $h(x) = \alpha e^{-\alpha x}$, for $x > 0$.

To envelop f we need to find

$$k^* = \sup_{x>0} \frac{f(x)}{h(x)} = \sup_{x>0} \frac{\alpha^m x^{m-1} e^{(\alpha-\lambda)x}}{\alpha\Gamma(m)}.$$

Clearly k^* will be infinite if $m < 1$ or $\lambda \leq \alpha$. For $m = 1$ the gamma is just an exponential. Thus we will assume $m > 1$ and choose $\alpha < \lambda$. For $m \in (0,1)$ the rejection method can still be used, but a different envelope is required. To find k^*, it is easiest to take the derivative of the *log* of right-hand side above and set it to zero, to find the point where the maximum occurs. This is at the point $x = (m-1)/(\lambda - \alpha)$, which gives

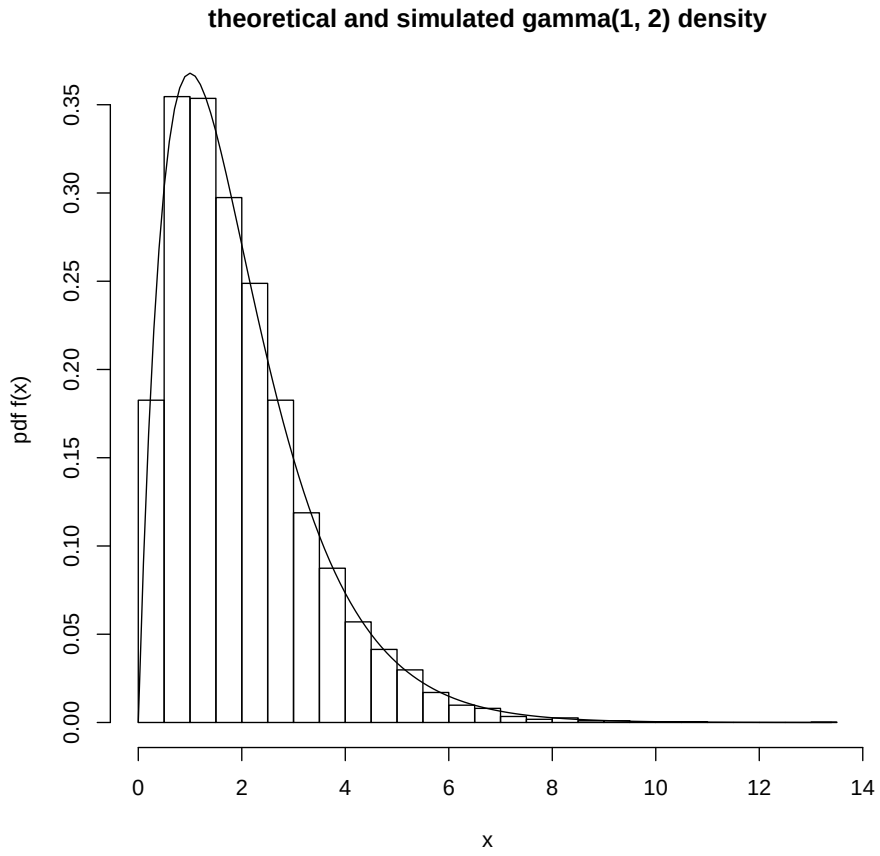
$$k^* = \frac{\lambda^m (m-1)^{m-1} e^{-(m-1)}}{\alpha(\lambda-\alpha)^{m-1}\Gamma(m)}.$$

To obtain the best efficiency it is necessary to choose our envelope to make k^* as small as possible. Looking at the formula for k^* this means choosing α to make $\alpha(\lambda-\alpha)^{m-1}$ as large as possible. Setting the derivative of the $\log(\alpha(\lambda-\alpha)^{m-1})$ with respect to α to zero, it is easy to see that the maximum occurs when $\alpha = \lambda/m$. Plugging this back in gives $k^* = m^m e^{-(m-1)}/\Gamma(m)$. This leads to the following code:

```
gamma.sim <- function(lambda, m) {
# sim a gamma(lambda, m) rv using rejection
# with an exponential envelope
# assumes m > 1 and lambda > 0
f <- function(x) lambda^m*x^(m-1)*exp(-lambda*x)/gamma(m)
h <- function(x) lambda/m*exp(-lambda/m*x)
k <- m^m*exp(1-m)/gamma(m)
while (TRUE) {
X <- -log(runif(1))*m/lambda
Y <- runif(1, 0, k*h(X))
if (Y < f(X)) return(X)
}
}

set.seed(1999)
n <- 10000
g <- rep(0, n)
for (i in 1:n) g[i] <- gamma.sim(1, 2)
hist(g, breaks=20, freq=F, xlab="x", ylab="pdf f(x)",
main="theoretical and simulated gamma(1, 2) density")
x <- seq(0, max(g), .1)
lines(x, dgamma(x, 2, 1))
```

Figure 5 shows the density and the sample of size 10000.

Figure 5: Gamma density and histogram of sample size 10000 using rejection



4.6 Alternative formulations

Alternative formulation
To simulate from the density f_X, we assume that we have envelope density h from which you can simulate, and that we have some $k < \infty$ such that $\sup_x f_X(x)/h(x) \leq k$.
Also suppose that we have a density f_Y which we can simulate from, with cdf F_Y.

1. Simulate X from h.
2. Generate Y from f_Y.
3. If $F_Y(Y) < \frac{f_X(X)}{kh(X)}$ then return X, o/w go back to step 1.

Another alternative formulation

1. Simulate X from h.
2. Generate Y from f_Y.
3. If $Y < g(X)$ then return X, o/w go back to step 1.

The pdf of the returned X is then $\propto h(x)F_Y(g(x))$.

5 Gibbs sampler

5.1 Definition

The Gibbs sampler

The Gibbs sampler is a random vector generation method that does not require the complete specification of the target multivariate probability distribution. Instead, it only requires the information of a set of associated conditional distributions. The Gibbs sampler is computationally feasible provided the conditional distributions are easily simulated.

Introduction to Gibbs sampler

- Suppose the aim is to generate a vector $\mathbf{u} = (u_1, \cdots, u_K)$ from the random vector $\mathbf{U} = (U_1, \cdots, U_K)$ which has a joint pdf $f(\mathbf{u})$.
- Suppose the pdf $f(\mathbf{u})$ has a very complicated form. But for each $k = 1, \cdots, K$, the conditional pdf of U_k given $\mathbf{U}_{-k} = (U_1, \cdots, U_{k-1}, U_{k+1}, \cdots, U_K)$ is known and relatively easy to simulate. The notation

  $$f(u_k|\mathbf{u}_{-k}) \equiv f(u_k|u_1, \cdots, u_{k-1}, u_{k+1}, \cdots, u_K)$$

  is used for such a conditional pdf.
- Then the Gibbs sampler used to generate one observation of $\mathbf{U}$ can be described as following:

Gibbs sampler

Algorithm A.1 Gibbs sampler

1° Arbitrarily generate/assign an initial vector $\mathbf{u}^{(0)} = (u_1^{(0)}, \cdots, u_K^{(0)})$ from the support of $f(\mathbf{u})$.

2° Generate a value $u_1^{(1)}$ from $f(u_1|u_2 = u_2^{(0)}, \cdots, u_K = u_K^{(0)})$; then generate a value $u_2^{(1)}$ from $f(u_2|u_1 = u_1^{(1)}, u_3 = u_3^{(0)}, \cdots, u_K = u_K^{(0)})$;
continue until a value $u_{K-1}^{(1)}$ is generated from $f(u_{K-1}|u_1 = u_1^{(1)}, \cdots, u_{K-2} = u_{K-2}^{(1)}, u_K = u_K^{(0)})$,
and generate a value $u_K^{(1)}$ from $f(u_K|u_1 = u_1^{(1)}, \cdots, u_{K-1} = u_{K-1}^{(1)})$.

The generated values are delivered as $\mathbf{u}^{(1)} = (u_1^{(1)}, \cdots, u_K^{(1)})$.

Gibbs sampler

Algorithm A.1 Gibbs sampler ctd

3° For $j = 1, 2, \cdots$, do the following: Generate a value $u_1^{(j)}$ from $f(u_1|u_2 = u_2^{(j-1)}, \cdots, u_K = u_K^{(j-1)})$;

4° then generate a value $u_2^{(j)}$ from $f(u_2|u_1 = u_1^{(j)}, u_3 = u_3^{(j-1)}, \cdots, u_K = u_K^{(j-1)})$;

5° continue until a value $u_{K-1}^{(j)}$ is generated from $f(u_{K-1}|u_1 = u_1^{(j)}, \cdots, u_{K-2} = u_{K-2}^{(j)}, u_K = u_K^{(j-1)})$,

6° and generate a value $u_K^{(j)}$ from $f(u_K|u_1 = u_1^{(j)}, \cdots, u_{K-1} = u_{K-1}^{(j)})$. The generated values are delivered as $\mathbf{u}^{(j)} = (u_1^{(j)}, \cdots, u_K^{(j)})$. **Note the conditioning is always based on the latest values of** $(u_1, \cdots, u_K)$.

The Gibbs sampler

Using the theory of Markov chains, it can be shown that

$$\mathbf{u}^{(j)} \xrightarrow{d} f(\mathbf{u}) \quad \text{as } j \to \infty$$

under fairly general conditions. Details will be given in the computational statistics course next year.

This implies that $\mathbf{u}^{(j)}$ can be roughly regarded as an observation of $\mathbf{U}$ from pdf $f(\mathbf{u})$ when j is sufficiently large. Empirical methods are available (details not pursued here) to determine how large j should be.

In practice, we usually generate a long sequence $\mathbf{u}^{(1)}, \cdots, \mathbf{u}^{(J)}, \mathbf{u}^{(J+1)}, \cdots, \mathbf{u}^{(J+m)}$ using the Gibbs sampler; then ignore the **burn-in** sequence $\mathbf{u}^{(1)}, \cdots, \mathbf{u}^{(J)}$ and use only the part $\mathbf{u}^{(J+1)}, \cdots, \mathbf{u}^{(m+J)}$ as a random sample from $f(\mathbf{u})$.

Estimating with the Gibbs sampler

Although $\mathbf{u}^{(m+1)}, \cdots, \mathbf{u}^{(m+J)}$ are not independent of each other, they constitute a Markov chain. So probabilities and expectations can be estimated from the values of the chain as follows:

1. For any u, the proportion of $\mathbf{u}_K^{(J+1)}, \cdots, \mathbf{u}_K^{(m+J)} = u$ is a consistent estimate, as $m, J \to \infty$, of $\mathbb{P}(U_K = u)$.
2. The sample mean of $\mathbf{u}_K^{(J+1)}, \cdots, \mathbf{u}_K^{(m+J)}$ is a consistent estimate of $\mathbb{E}(U_K)$.
3. Amongst $j \in \{J+1, \cdots J+m\}$ for which $\mathbf{u}_1^{(j)} = v$, the proportion of $\mathbf{u}_K^{(j)} = u$ is a consistent estimate of $\mathbb{P}(U_K = u | U_1 = v)$.
4. Amongst $j \in \{J+1, \cdots J+m\}$ for which $\mathbf{u}_1^{(j)} = v$, the sample mean of $\mathbf{u}_K^{(j)}$ is a consistent estimate of $\mathbb{E}(U_K | U_1 = v)$.

Applying the Gibbs sampler to posteriors

Suppose U_K is the parameter and U_1 is the observation (both might be vectors but we are assuming for simplicity they are rv's). Then from the estimates 3 and 4 in the previous slide:

1. The posterior pmf or pdf of $U_K | U_1 = v$ is estimated from the sample of values $\mathbf{u}_K^{(j)}$ for which $\mathbf{u}_1^{(j)} = v$.
2. The mean of the posterior distribution is estimated by the sample mean of the values $\mathbf{u}_K^{(j)}$ for which $\mathbf{u}_1^{(j)} = v$.

5.2 Gibbs sampler: example

Example - insect survival

Insect eggs in a certain area hatch and remain in the area so they can be caught with probability p. Suppose there are N eggs laid in the area, and that X eggs of them hatch to produce insects that can then be caught. Suppose p and N are unknown whilst X is observed. The scientist is interested in p as a measure of survival. A Bayesian approach to estimate p might take independent prior distributions for $N \sim \text{Poi}(16)$ and $p \sim \text{Beta}(2, 4)$ based on previous information. Find:

1. the joint pdf of (X, N, p)
2. the posterior pdf of p given $X = x$
3. a Gibbs sampler for generating (X, p, N).

Implement the algorithm in R with $m = 1,000,000$ and $J = 10,000$ and

1. find and plot an estimate of the posterior pdf of p given $X = 5, 10$ or 15
2. plot the posterior mean of p given $X = x$ versus x together with a 95% posterior probability interval

Repeat the R commands with $m = 1,000$ and $J = 100$ and compare the results.

When $N \sim \text{Poi}(16)$ and $p \sim \text{Beta}(2,4)$, the joint pdf of (X, N, p) is

$$\begin{aligned} f(x,N,p) &= f(x|p,N) \cdot f(N) f(p) \cdot \\ &= \binom{N}{x} p^x (1-p)^{N-x} \cdot \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} p^{a-1}(1-p)^{b-1} \cdot \frac{\lambda^N}{N!} e^{-\lambda} \\ &= 20 e^{-16} \binom{N}{x} \frac{16^N}{N!} p^{x+1}(1-p)^{N-x+3}; \\ & \quad x = 0, 1, \cdots, N;\ 0 \le p \le 1;\ N = 0, 1, 2, \cdots . \end{aligned}$$

The joint pdf of X, p is

$$\begin{aligned} f(x,p) &= \sum_{N=x}^{\infty} f(x,p,N) \\ &= \sum_{N=x}^{\infty} 20 e^{-16} \binom{N}{x} \frac{16^N}{N!} p^{x+1}(1-p)^{N-x+3} \end{aligned}$$

The marginal pmf of X is

$$\begin{aligned} f(x) &= \sum_{N=x}^{\infty} \int_0^1 f(x,p,N) dp \\ &= \sum_{N=x}^{\infty} \frac{20 e^{-16}(x+1)(N-x+3)(N-x+2)(N-x+1)16^N}{(N+5)!} \end{aligned}$$

both of which are difficult to further simplify. The posterior pdf of p given $X = x$ is the ugly expression $f(x,p)/f(x)$.

But it is not difficult to find the full conditional pdf's:

$$(X|p,N) \stackrel{\text{d}}{=} \text{bin}(N,p), \qquad (p|x,N) \stackrel{\text{d}}{=} \text{Beta}(x+2, N-x+4),$$
$$\text{and} \qquad (N-x|x,p) \stackrel{\text{d}}{=} \text{Poi}(16(1-p)).$$

Proof:

$$\begin{aligned} f(p|x,N) &= \frac{f(x,p,N)}{f(x,N)} = \frac{20 e^{-16}\binom{N}{x}\frac{16^N}{N!} p^{x+1}(1-p)^{N-x+3}}{\int_0^1 20 e^{-16}\binom{N}{x}\frac{16^N}{N!} p^{x+1}(1-p)^{N-x+3} dp} \\ &= \frac{p^{x+1}(1-p)^{N-x+3}}{\int_0^1 p^{x+1}(1-p)^{N-x+3} dp} = \frac{\Gamma(N+6)}{\Gamma(x+2)\Gamma(N-x+4)} p^{x+1}(1-p)^{N-x+3} \end{aligned}$$

which is a pdf of $\text{Beta}(x+2, N-x+4)$.

$$\begin{aligned} f(N|x,p) &= \frac{f(x,p,N)}{f(x,p)} = \frac{20 e^{-16}\binom{N}{x}\frac{16^N}{N!} p^{x+1}(1-p)^{N-x+3}}{\sum_{N=x}^{\infty} 20 e^{-16}\binom{N}{x}\frac{16^N}{N!} p^{x+1}(1-p)^{N-x+3}} \\ &= \frac{\binom{N}{x}[16(1-p)]^N \frac{1}{N!}}{\sum_{\tilde{N}=x}^{\infty}\binom{\tilde{N}}{x}\frac{16^{\tilde{N}}}{\tilde{N}!}(1-p)^{\tilde{N}}} = \frac{\frac{1}{(N-x)!}[16(1-p)]^N}{\sum_{\tilde{N}=x}^{\infty}\frac{1}{(\tilde{N}-x)!}[16(1-p)]^{\tilde{N}}} \\ &= \frac{[16(1-p)]^{N-x}}{(N-x)!} e^{-16(1-p)}; \quad N = x, x+1, x+2, \cdots . \end{aligned}$$

This implies that $(N - x|x, p) \stackrel{\text{d}}{=} \text{Poisson}(16(1-p))$. □

Now a sample of (X, p, N) can be generated using the following Gibbs sampler algorithm

Gibbs sampler algorithm for (X, p, N)]

1° Arbitrarily choose an initial vector, e.g. $(x^{(0)}, p^{(0)}, N^{(0)}) = (8, 0.5, 16)$.

2° $(x^{(1)}, p^{(1)}, N^{(1)})$ is obtained by: generating $x^{(1)}$ from $\text{Bin}(N^{(0)}, p^{(0)})$; generating $p^{(1)}$ from $\text{Beta}(x^{(1)} + 2, N^{(0)} - x^{(1)} + 4)$; and generating an $N^{(1)}$ from $\text{Poi}(16(1 - p^{(1)})) + x^{(1)}$.

3° $(x^{(j)}, p^{(j)}, N^{(j)})$, $j = 1, 2, \cdots$, is obtained by: generating $x^{(j)}$ from $\text{Bin}(N^{(j-1)}, p^{(j-1)})$; generating $p^{(j)}$ from $\text{Beta}(x^{(j)} + 2, N^{(j-1)} - x^{(j)} + 4)$; and generating an $N^{(j)}$ from $\text{Poi}(16(1 - p^{(j)})) + x^{(j)}$.

Once a sample of (X, p, N) is obtained, the questions can be answered from the relevant parts of the sample.

Gibbs Sampler

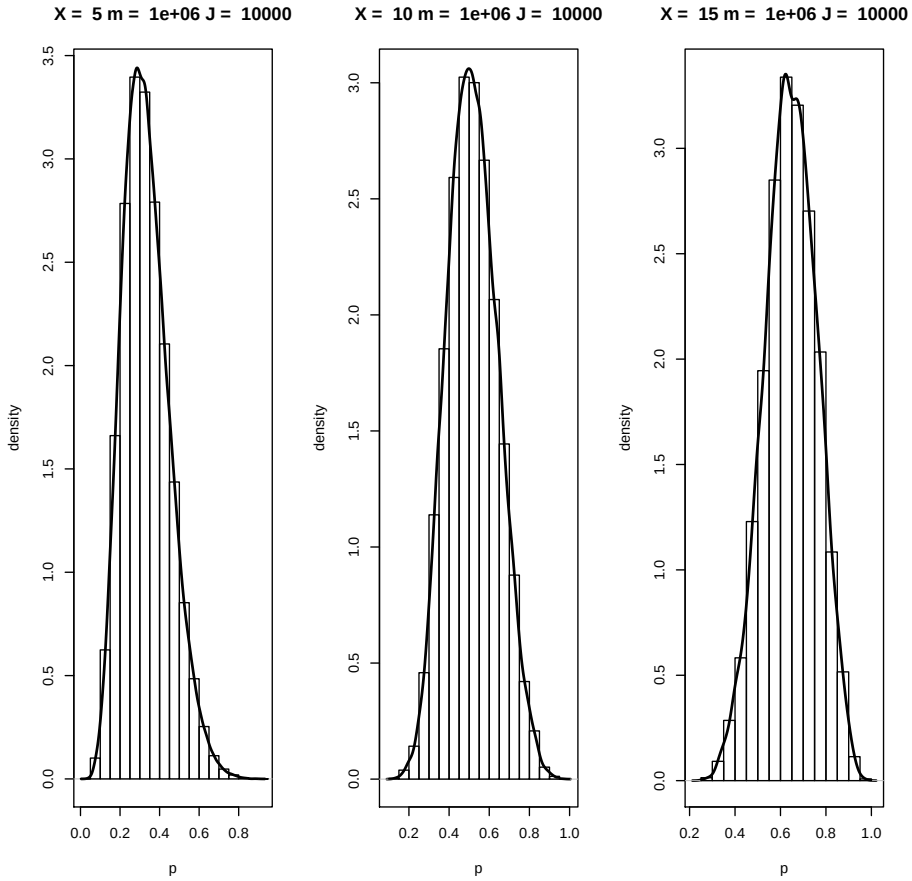
```
gibbs.f2 = function(x0, p0, N0, m, J){
# Geneerates m samples of (x,p,N) values by the Gibbs sampler.
# In total J+m samples are generated but the first J are discarded.
# (x0,p0,N0) is the initial value.
    x.seq <- p.seq <- N.seq <- rep(-1, J+m+1)
    x.seq[1] <- x0; p.seq[1] <- p0; N.seq[1] <- N0
    for(j in 2:(J+m+1)) {x.seq[j] <- rbinom(1, N.seq[j-1], p.seq[j-1])
       p.seq[j] <- rbeta(1, (x.seq[j] + 2), (N.seq[j-1] - x.seq[j] + 4))
       N.seq[j] <- rpois(1, 16 * (1 - p.seq[j])) + x.seq[j]}
    result <- list(X=x.seq[(J+2):(J+m+1)],p=p.seq[(J+2):(J+m+1)],
                   N=N.seq[(J+2):(J+m+1)])
    result
    }
```

Sample and plot posterior densities

```
J=10000
m=1000000
set.seed(456)
gibbsam2=gibbs.f2(8, 0.5, 16, m, J)
par(mfrow=c(1,3))
# plot the three conditional densities with histograms
# selecting the right X values
for (x in c(5,10,15))
{plot(density(gibbsam2$p[gibbsam2$X==x]), xlab="p",
ylab="density", lwd=2,main=paste("x = ",x,"m = ",m,"J = ",J),
ylim=range(hist(gibbsam2$p[gibbsam2$X==x],plot=F)$density))
hist(gibbsam2$p[gibbsam2$X==x], freq=F, add=T)}
```

Find posterior means and probability intervals

Figure 6: Post. dens. for p given $X = x$, m = sample size, J = burn-in



```
# find posterior means  and upper and lower points of 95% prob. int.
meanp0 <- tapply(gibbsam2$p,gibbsam2$X,mean)
xval0 <- as.integer(row.names(meanp0))
lowerp0 <- tapply(gibbsam2$p,gibbsam2$X,quantile,prob=c(0.025))
upperp0 <- tapply(gibbsam2$p,gibbsam2$X,quantile,prob=c(0.975))
# select only x values for which there is sufficient range in p
xval <- xval0[lowerp0<upperp0]
meanp <- meanp0[lowerp0<upperp0]
lowerp <- lowerp0[lowerp0<upperp0]
upperp <- upperp0[lowerp0<upperp0]
```

Plot posterior means and probability intervals

```
# plot means and prob.ints
par(mfrow=c(1,1))
plot(xval, meanp,
ylim=range(c(lowerp, upperp)),
pch=19, xlab="x",
ylab="Post. Mean and Prob. Int.",
main=paste("m = ",m,"J = ",J))
# hack: we draw arrows
# but with very special "arrowheads"
arrows(xval, lowerp, xval, upperp,
length=0.05, angle=90, code=3)
```

Figures 8 and 9 show the same plots but with the burn-in just at 100 and the sample size just 1000. The plots are much less regular and bumpy indicating that both the sample size and possible also the burn-in are too small.

To rub this in Figures 10 and 11 show this with another seed, 789.

Whereas in Figures 12 and 13 $J = 10,000$ and $m = 1,000,000$ is used with seed 789.

5.3 Gibbs sampler for parameters and data

More complex models often have:

- a number of parameters
- the pdf or pmf of the data specificed conditional on the parameters, - eg a random sample from a particular distribution using the parameters
- a complicated model for the data but simpler conditional distributions
- dependencies that can be expressed in a graph, a DAG - *Directed Acyclic Graph* - like we saw in Module 11 and repeated in Figure
- direct capacity to simulate the conditional distributions, as above or
- further approximations in the simulation of the conditional distributions

Figure 7: Post. mean and prob. int. for p given $X = x$, m = sample size, J = burn-in

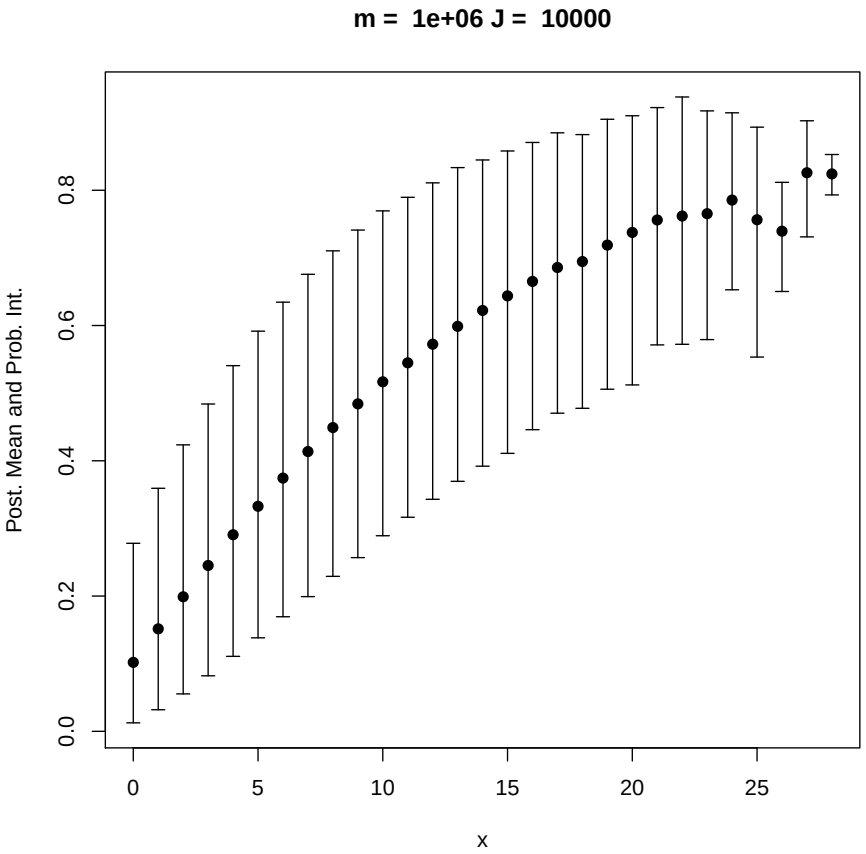


Figure 8: Post. dens. for p given $X = x$, m = sample size, J = burn-in

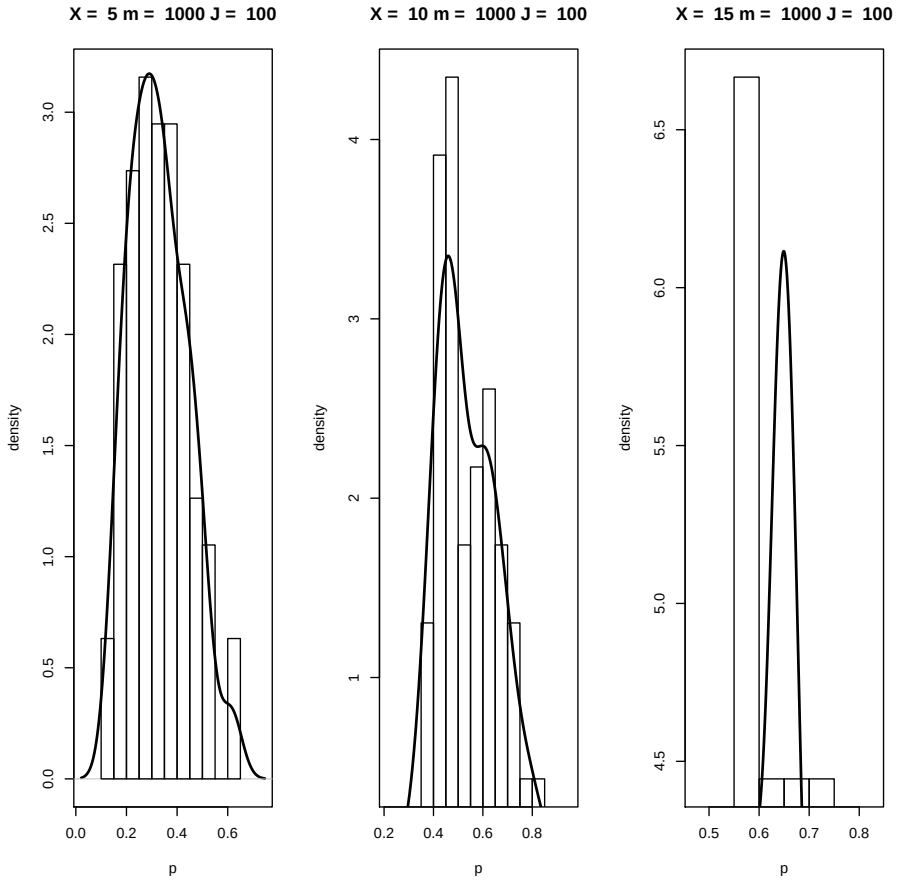


Figure 9: Post. mean and prob. int. for p given $X = x$, m = sample size, J = burn-in

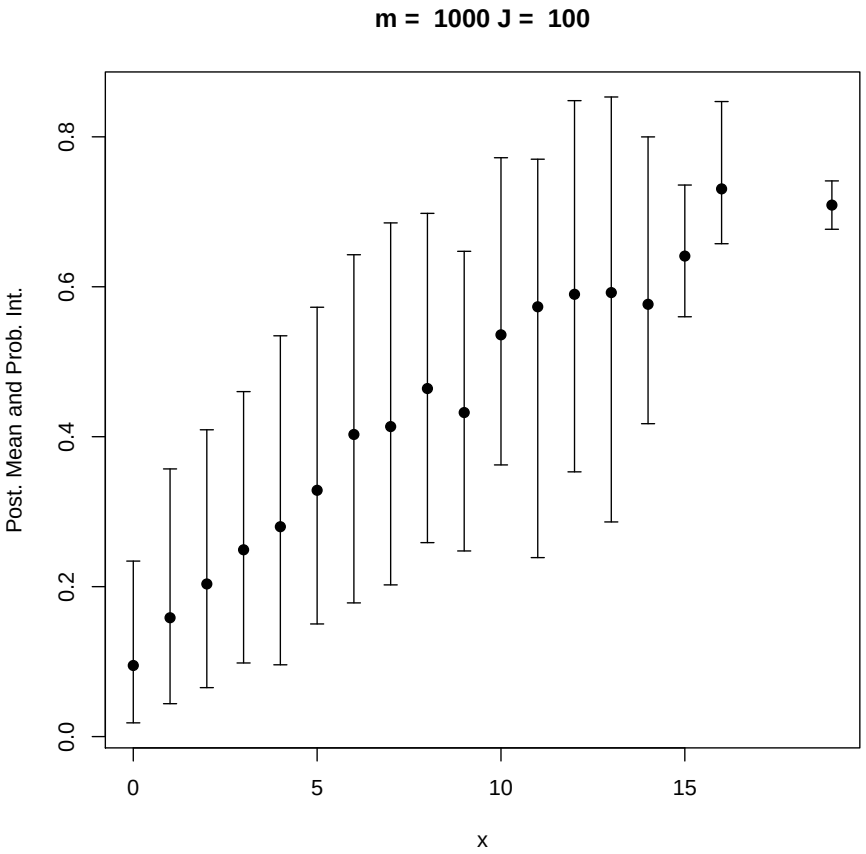


Figure 10: Post. dens. for p given $X = x$, m = sample size, J = burn-in

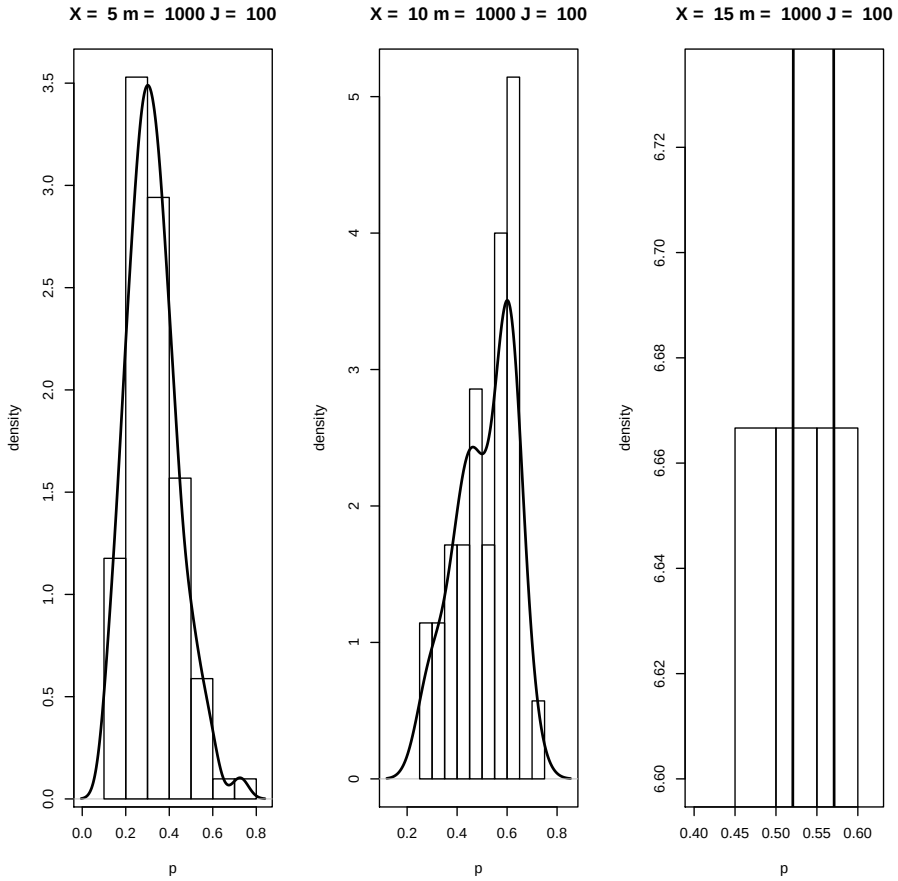


Figure 11: Post. mean and prob. int. for p given $X = x$, m = sample size, J = burn-in

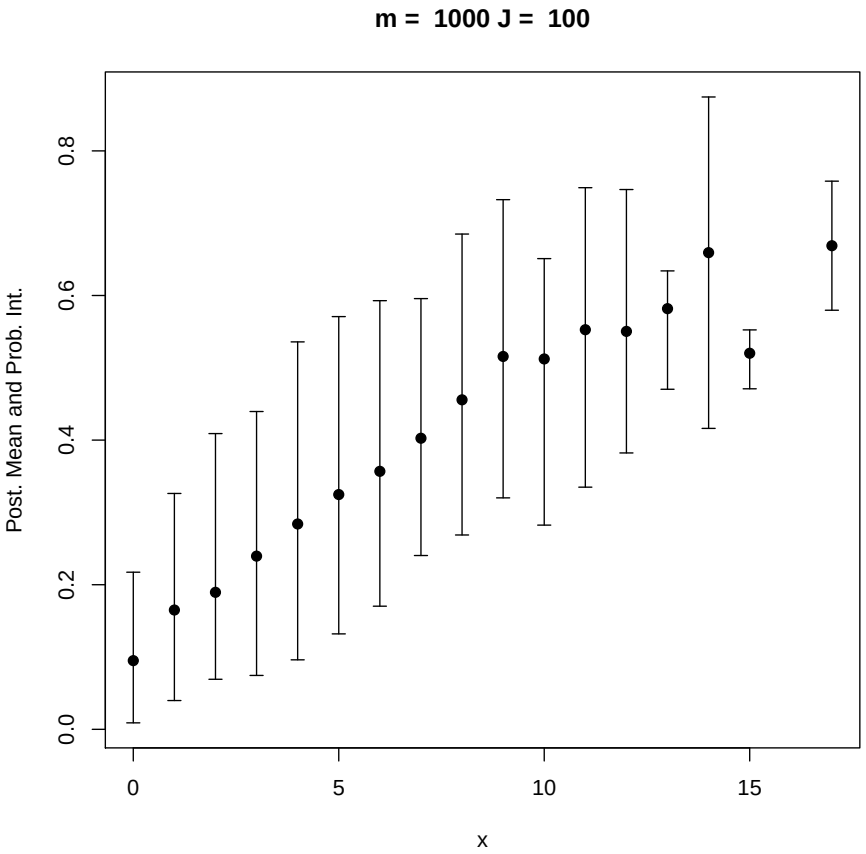


Figure 12: Post. dens. for p given $X = x$, m = sample size, J = burn-in

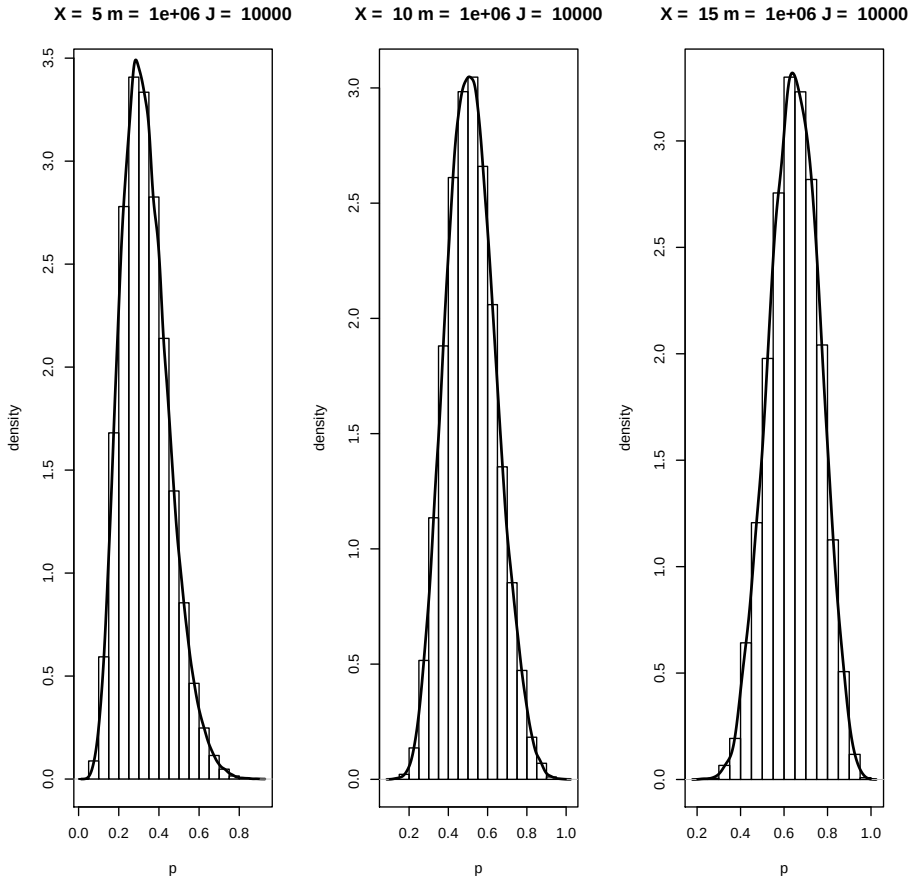


Figure 13: Post. mean and prob. int. for p given $X = x$, m = sample size, J = burn-in

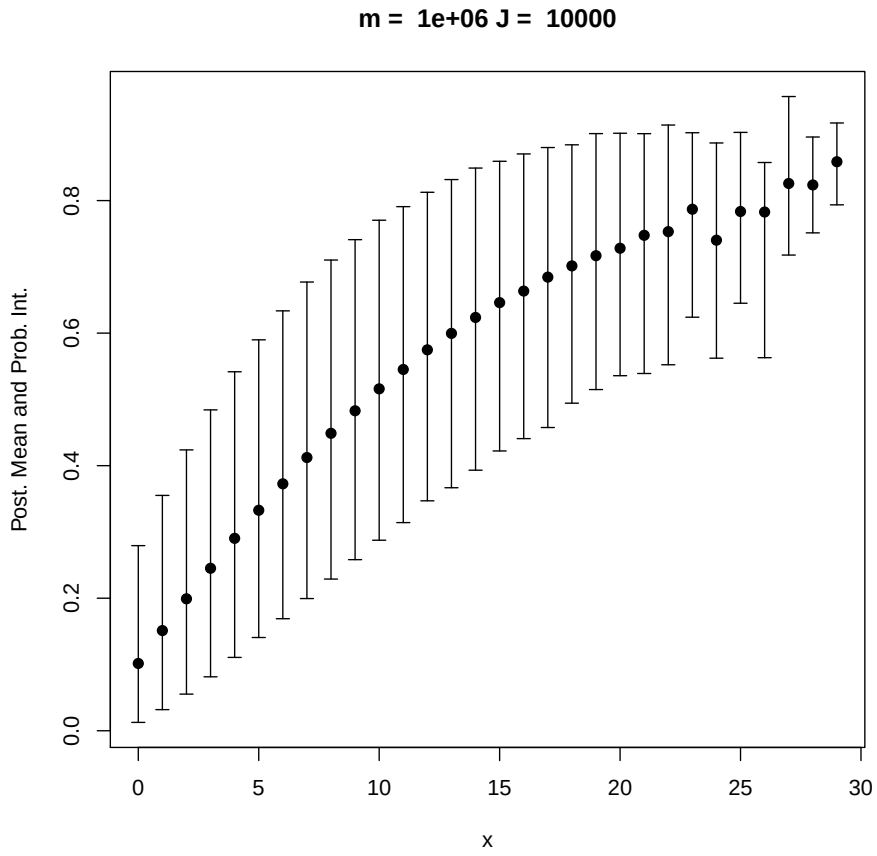
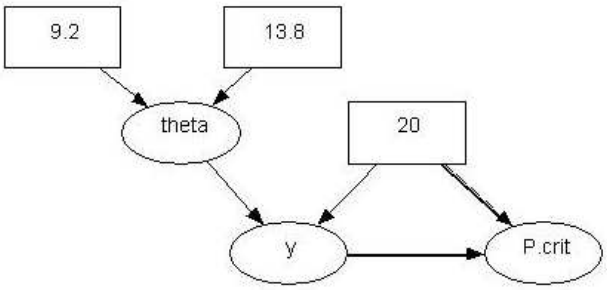


Figure 14: Simple Directed Acyclic Graph



Packages for Bayesian inference:

- WinBUGs and OpenBugs stand-alone, open source forgetting posteriors using DAGs
- JAGS (Just Another Gibbs Sampler) uses the same language as WinBUGS and is available in R
- Stan (named after the pioneer of simulation Stanislaw Ulam) has a different language and programs are compiled, but it is also available in R
- there are no guarantees that the simulations will work and care is required in implementation

6 Unsupervised Learning

6.1 Introduction

Introduction

Unsupervised learning refers to description of data with a number of variables in which no specific response variable is indentified. Traditional statistical methods using linear models or generalised linear models may be appropriate once response variables have been identified. At the earliest stage, appropriate plots and other methods of exploratory description are needed. Here the aim is just to describe a multivariate data frame with p columns and n rows. Two techniques will be described:

- Principal Component Analysis - picking linear combinations of the variables to summarise the variability in the observations
- K Means Clustering - picking groups of observations with similar values.

6.2 Principal Components Analysis (PCA)

Principal Components Analysis (PCA)

Suppose Σ is the covariance matrix of a p random vector of $\mathbf{X}$. This can include the case where $\mathbf{X}$ is a dataframe with n rows and p variables, because the random p vector then becomes uniform random selection from the n rows of the dataframe. In this case, the sample covariance matrix of the dataframe is the covariance matrix of $\mathbf{X}$. From results on matrices in Module 2 p.12, $\Sigma = P\Lambda P^T$ where

- P is an orthogonal matrix with the eigenvectors of Σ as its columns
- Λ is a diagonal matrix with the (non-negative) eigenvalues of Σ along the diagonal
- The eigenvalues can be - and from now on are assumed to be - organised in descending order from the top left hand entry of Λ

Recall that the eigenvectors in P must have length one and be orthogonal - P is called a rotation matrix as a result.

For any column vector $\mathbf{a}$, if $\mathbf{a}$ has unit length $\mathbf{a}^T\mathbf{a} = 1$, the column vector $\mathbf{b} = P^T\mathbf{a}$ also has unit length, since

$$\mathbf{b}^T\mathbf{b} = \mathbf{a}^T PP^T\mathbf{a} = \mathbf{a}^T I\mathbf{a} = 1$$

recalling that the inverse of an orthogonal matrix is its transpose. Further, from variance properties in Module 4 p.3,

$$Var(\mathbf{a}^T\mathbf{X}) = \mathbf{a}^T\Sigma\mathbf{a} = \mathbf{b}^T\Lambda\mathbf{b}.$$

And so

$$Var(\mathbf{a}^T\mathbf{X}) = \mathbf{b}^T\Lambda\mathbf{b} = \sum_{i=1}^{p}\Lambda_{ii}b_i^2 \leq \Lambda_{11}\sum_{i=1}^{p}b_i^2 = \Lambda_{11} \tag{1}$$

If $\mathbf{a}$ is taken to be the first column of P, that is the eigenvector corresponding to the largest eigenvalue of Σ, then $\mathbf{b}^T = \mathbf{a}^T P = (1, 0, \cdots, 0)$. The bound in inequality (1) is then achieved so that $\mathbf{a}^T\mathbf{X}$ has the greatest variance amongst all linear combinations of elements of $\mathbf{X}$. This linear combination $\mathbf{a}^T\mathbf{X}$ is called the *first principal component* . For data, the *first principal component* is the linear combination of the variables with weights equal to the first eigenvector. The *first principal component* is often said to be the linear combination of variables which most *explains* the variation in the multivariate data. If $\mathbf{a}$ is taken to be the second column of P, it is the eigenvector corresponding to the second largest eigenvalue. The random variable $\mathbf{a}^T\mathbf{X}$ is called the *second principal component*.

For data, the *second principal component* is the linear combination of the variables with weights equal to the second eigenvector of the sample covariance matrix. This has maximal variance amongst all linear combinations of the variables that are orthogonal to the first linear combination. For data, the orthogonality is uncorrelatedness of the components. Often principal components is performed with the correlation matrix - the covariance matrix for the standardised random variables. The reason for this is that it eliminates the scale of the random variables as a determinant of the principal components. The first two principal components are often plotted against each other as this can reveal structure in the data. This is particularly appropriate if the sum of their variances accounts for a large proportion of the total of variances of the principal components.

6.3 Example: Global Temperatures

Example: Global Temperatures

The Kaggle web site, Climate Change Earth Surface Temperature Data, contains data from the organisation Berkeley Earth. Berkeley Earth is affiliated with Lawrence Berkeley National Laboratory in the US. The Berkeley Earth Surface Temperature Study combines 1.6 billion temperature reports from 16 pre-existing archives. Figure 15 shows their estimates of land-surface average temperatures together with a 95% confidence interval. Also shown are estimates from the most US Government National Oceanigraphic and Atmospheric Administration, NASA and the UK Government Hadley Centre of the Meterological Office.

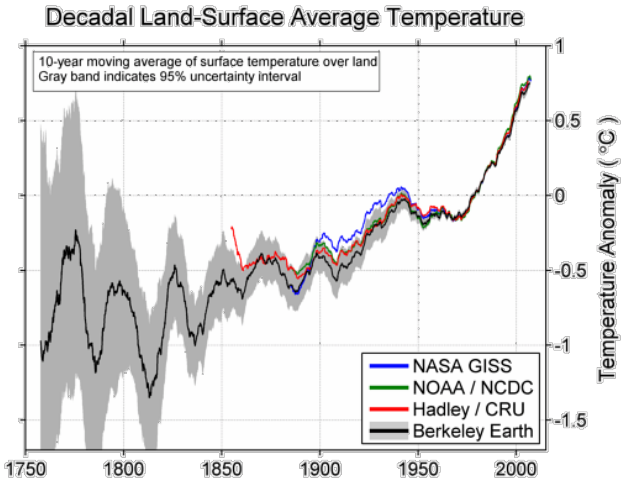


Figure 15: Berkeley Earth Global Temperatures From 1750

PCA can be illustrated from one of the Berkley Earth data sets, Global Land and Ocean-and-Land Temperatures (GlobalTemperatures.csv), which is available in the data folder on the LMS. The data set gives average land temperature from 1850, max and min land temperatures and global ocean and land temperatures. It also gives uncertainty estimates for each average, so that the average plus and minus the uncertainty estimate gives a 95% confidence interval for the average. The next slide gives descriptions of each of the variables.

1. Date: the date at the start of the month
2. L_Ave: global average land temperature in celsius
3. L_AveCI: half the width of the 95% confidence interval around the average
4. L_Max: global average maximum land temperature in celsius
5. L_MaxCI: half the width of the 95% confidence interval around the average
6. L_Min: global average minimum land temperature in celsius
7. L_MinCI: half the width of the 95% confidence interval around the average
8. LO_Ave: global average land and ocean temperature in celsius
9. LO_AveCI: half the width of the 95% confidence interval around the average

```
GT <- read.csv("../data/GlobalTemperatures.csv")
GT$Date <- as.Date(GT$Date)
GT$Year <- as.integer(format(GT$Date,"%Y"))
GT$YearGroup <-
cut(GT$Year, breaks = c(1849,1899,1939,1979,1999,2020),
```

```
labels = c("pre 1900","pre 1940",
"pre 1980","pre 2000","after 2000"))
GT$Month = format(GT$Date,'%b')
```

```
#  do Principal Components on Columns 2 to 9
#  summary gives the proportion of variance for each component
PCAtemp <- prcomp(GT[2:9])
summary(PCAtemp)


## Importance of components%s:
##                           PC1     PC2     PC3     PC4     PC5     PC6
## Standard deviation     7.4453 0.74341 0.33829 0.25820 0.17922 0.17140
## Proportion of Variance 0.9857 0.00983 0.00203 0.00119 0.00057 0.00052
## Cumulative Proportion  0.9857 0.99555 0.99758 0.99877 0.99934 0.99986
##                           PC7     PC8
## Standard deviation     0.08660 0.01549
## Proportion of Variance 0.00013 0.00000
## Cumulative Proportion  1.00000 1.00000


#  PCAtemp$ rotation has the component weightings
PCAtemp$rotation


##                   PC1         PC2          PC3           PC4         PC5
## L_Ave      0.571946823  0.04001266  0.039868197  0.7436785600 -0.13625963
## L_AveCI   -0.004512822  0.27501118 -0.045303280 -0.0055524370 -0.15597650
## L_Max      0.577714155  0.08747253 -0.708251464 -0.3749478512  0.11347921
## L_MaxCI   -0.009022128  0.76075681  0.180640746  0.0008976024  0.59460736
## L_Min      0.557041711 -0.07698389  0.676395111 -0.4656649725 -0.08395457
## L_MinCI   -0.010408586  0.55934918 -0.008052543 -0.0046540451 -0.68469486
## LO_Ave     0.169161866 -0.09745747  0.064428657  0.2990879162  0.33460064
## LO_AveCI -0.001370918  0.08983038 -0.019511093 -0.0023589538 -0.05344326
##                   PC6         PC7          PC8
## L_Ave     -0.31008624 -0.04301755 -0.007607443
## L_AveCI   -0.04266753  0.90723977 -0.270244467
## L_Max      0.04250666 -0.04106506 -0.004265789
## L_MaxCI   -0.13651667 -0.12767968 -0.006243556
## L_Min      0.01821578  0.04493341  0.004960284
## L_MinCI    0.37956797 -0.27203576 -0.007162697
## LO_Ave     0.85793382  0.14038710  0.023771881
## LO_AveCI -0.03358999  0.24768133  0.962399015
```

The summary shows that the first two Principal Components account for most of the variation in all eight components The component weightings show that the First Principal Component is mostly an average of the Land Ave, Max and Min with a smaller weighting on the Land and Ocean Average, and very small weightings on the CI components. The Second Principal Component is mostly and average of the CI for Max and Min on Land with a smaller weighting for the land average and very small weightings on the other variables.

```
# libraries \texttt{ggplot2} and \texttt{ggfortify} are needed
# for easy plots of the first two principal components
library(ggplot2)
library(ggfortify)
# Plot of first two Principal Components
autoplot(PCAtemp)
# colour by Year Group
autoplot(PCAtemp,data= GT,colour="YearGroup")
# colour by Month
autoplot(PCAtemp,data= GT,colour="Month")
```

The next three slides shows the plots in Figures 16, 17 and 18. The non-coloured plot shows that there is structure and the coloured ones show that both month and year produce clusters.

Clearly the CI columns are on a different scale to the various temperature columns, so it would be better to do the plots by scaling: the mean is subtracted from each variable and the result is divided by the standard deviation. This

Figure 16: Plot of First Two Principal Components against each other

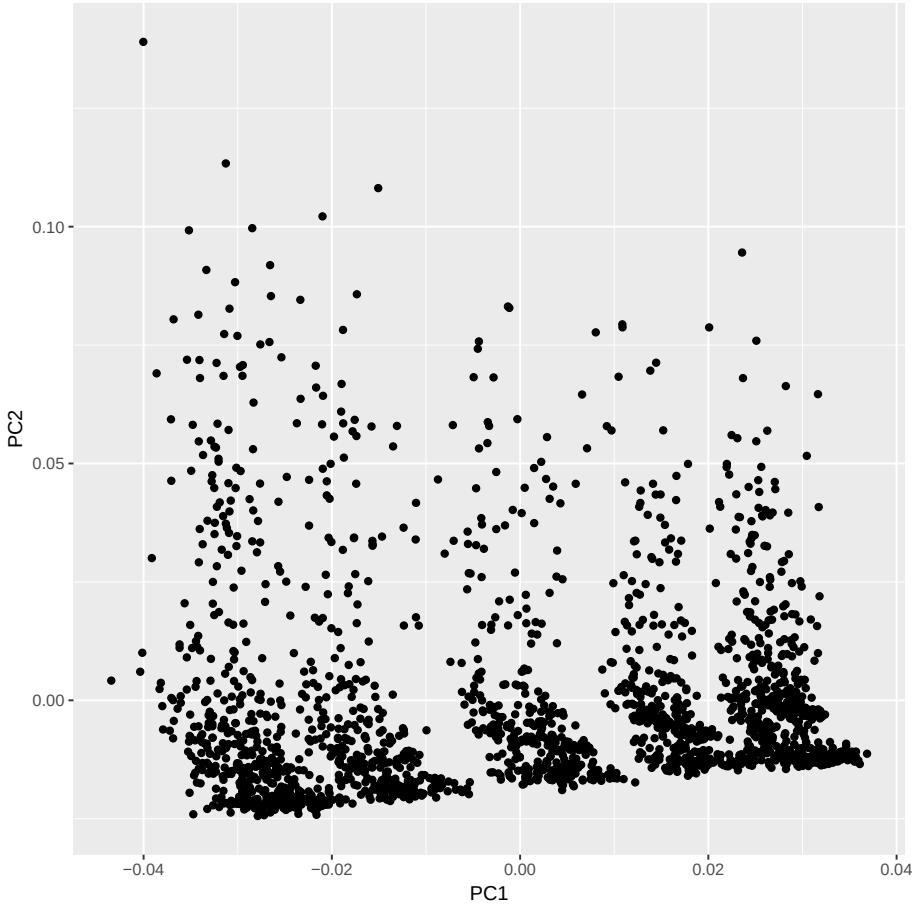


Figure 17: PCA Plot with Year Group coloured

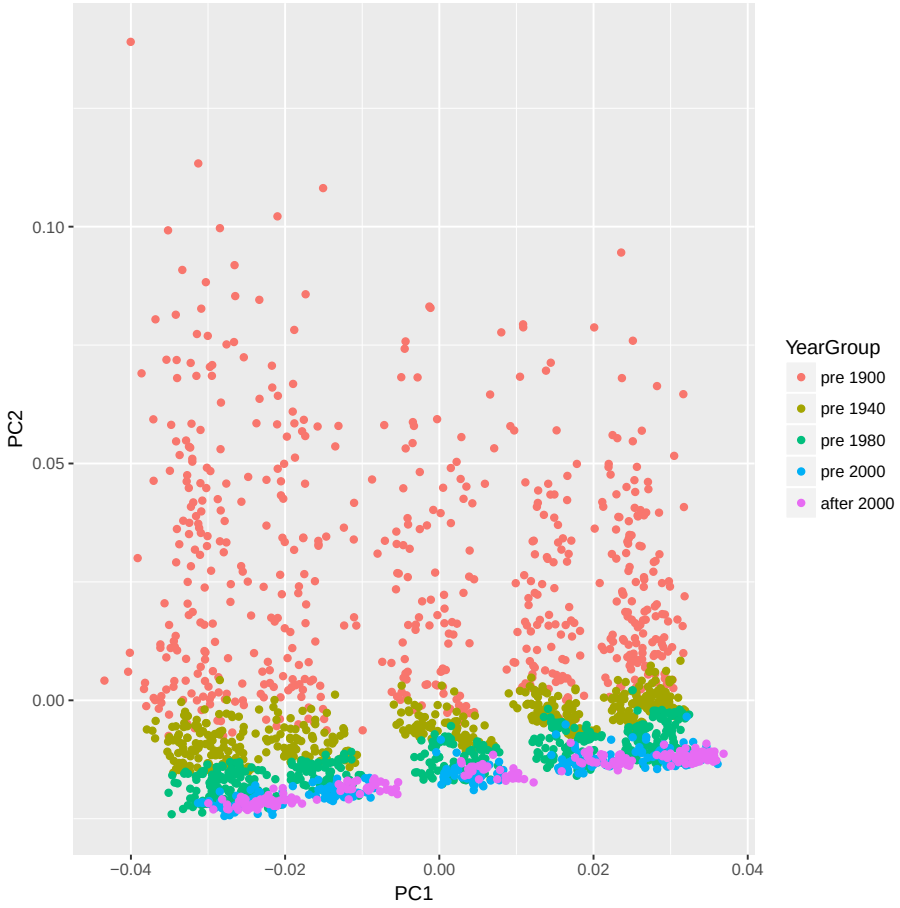
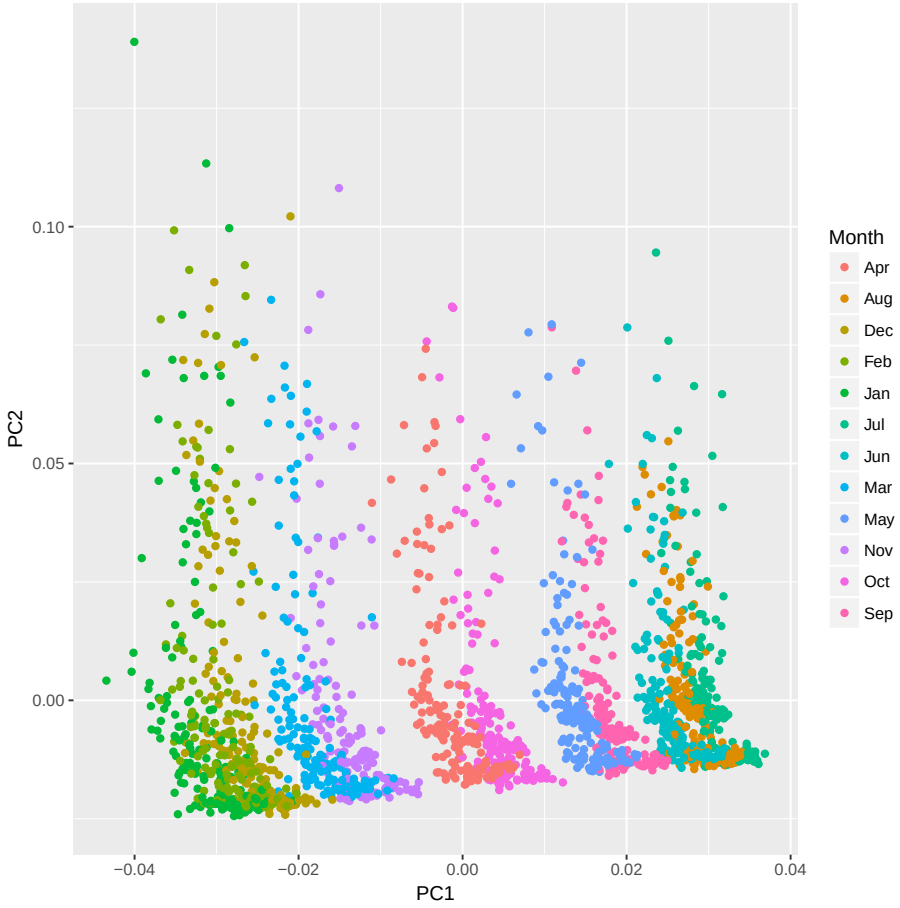


Figure 18: PCA Plot with Month coloured



is just doing the PCA with the correlation matrix rather than the covariance matrix. The results show that the first two principal components still account for most of the total variance in the principal components. The first principal component is close to the difference between the average of the temperature variables and the uncertainty variables. The second principal component is close to an average of all variables. From 1980 onwards the temperature variables are high and the uncertainties lower as can be seen in the plots.

```
# standardise variables by subtracting mean and dividing SD
PCAtempcor <- prcomp(GT[2:9],scale = TRUE)
summary(PCAtempcor)


## Importance of components%s:
##                           PC1    PC2     PC3     PC4     PC5     PC6
## Standard deviation     2.113 1.7838 0.42115 0.35380 0.16829 0.11880
## Proportion of Variance 0.558 0.3977 0.02217 0.01565 0.00354 0.00176
## Cumulative Proportion  0.558 0.9558 0.97794 0.99359 0.99713 0.99889
##                            PC7     PC8
## Standard deviation     0.07554 0.05633
## Proportion of Variance 0.00071 0.00040
## Cumulative Proportion  0.99960 1.00000


PCAtempcor$rotation


##                PC1       PC2         PC3           PC4         PC5
## L_Ave     0.3883726 0.3190540 -0.02014057  0.0005663997 -0.06476292
## L_AveCI  -0.3157703 0.3977348 -0.44317094 -0.1254976876  0.71341443
## L_Max     0.3855833 0.3223742 -0.05419762  0.0114395136 -0.10248587
## L_MaxCI  -0.2927403 0.3955458  0.69087016 -0.5282526974 -0.01581318
## L_Min     0.3934632 0.3089703  0.01519964 -0.0045519228 -0.04887134
## L_MinCI  -0.3169789 0.3772258  0.27841818  0.8234703267 -0.03621378
## LO_Ave    0.4036124 0.2864475  0.04937042  0.0346052614  0.22533309
## LO_AveCI -0.3112757 0.3997697 -0.49271672 -0.1604575077 -0.64932068
##                   PC6          PC7           PC8
## L_Ave     0.132165675 -0.008086470 -0.8516090093
## L_AveCI   0.143846268  0.004429598 -0.0165687360
## L_Max     0.324679898  0.711601274  0.3493355744
## L_MaxCI   0.008702339  0.034181359  0.0002258244
## L_Min     0.357743364 -0.699281260  0.3607066271
## L_MinCI   0.016555557 -0.002012846 -0.0039242571
## LO_Ave   -0.824301453 -0.008014694  0.1452505946
## LO_AveCI -0.220500413 -0.057512440  0.0350681032
```

Now do the plots using the PCA from the Correlation Matrix

```
autoplot(PCAtempcor)
autoplot(PCAtempcor,data= GT,colour="YearGroup")
autoplot(PCAtempcor,data= GT,colour="Month")
```

The next three slides shows these plots in Figures 19, 20 and 21.

Clearly from the plots of the Principal Components, both for the covariance or the correlation matrix, Month is important - probably reflecting more Northern Hemisphere collecting stations than Southern Hemisphere. This is not surprising since 90% of the world's population lives in the Northern Hemisphere (Wikipedia), collecting stations are more on land than ocean and the Northern Hemisphere has two thirds of the land (Wikipedia). It makes sense, therefore, to aggregate the data by year, recording the mean of each of the variables for the year. For uncertainty estimates, the additional averaging will make a uniform difference for all variables. Scaling, that is using the correlation matrix, will remove this difference and so all the results from now on are done with correlation matrices.

```
# Select Year and the temperature and CI variables
# Aggregate by Year using the mean of the monthly variables
# Set up grouping by Year
GTYear <- GT[2:10]
```

Figure 19: Plot of First Two Principal Components on Correlation Matrix

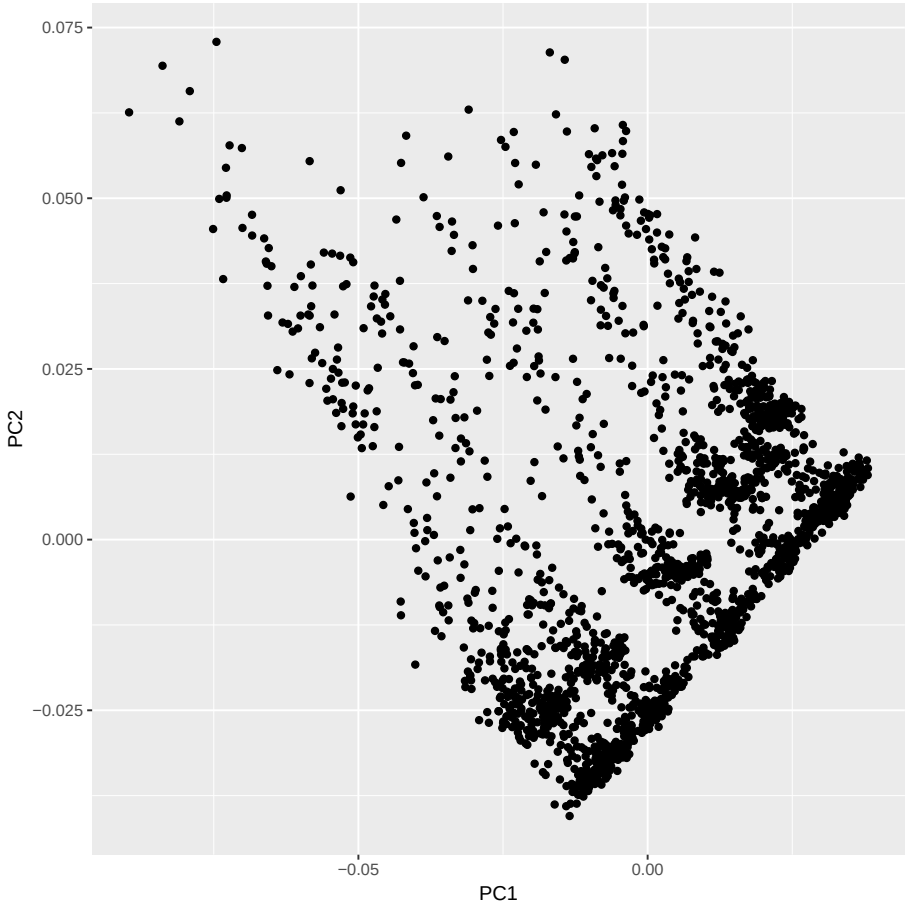


Figure 20: PCA Plot on Correlation Matrix with Year Group coloured

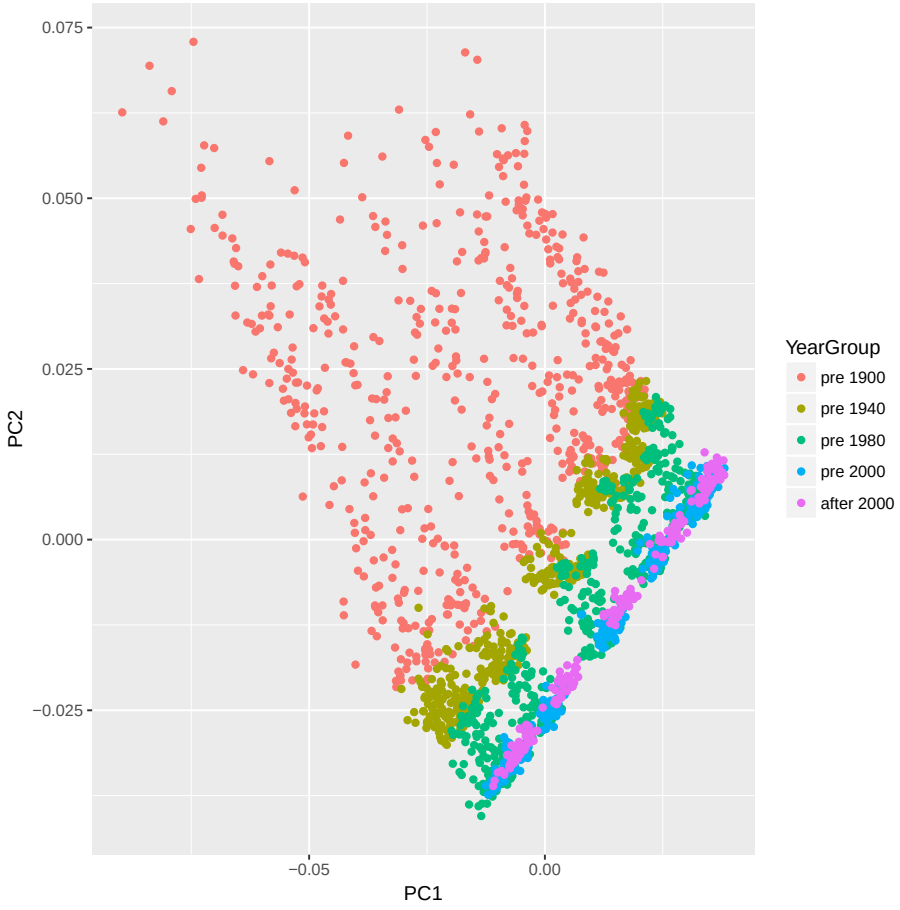
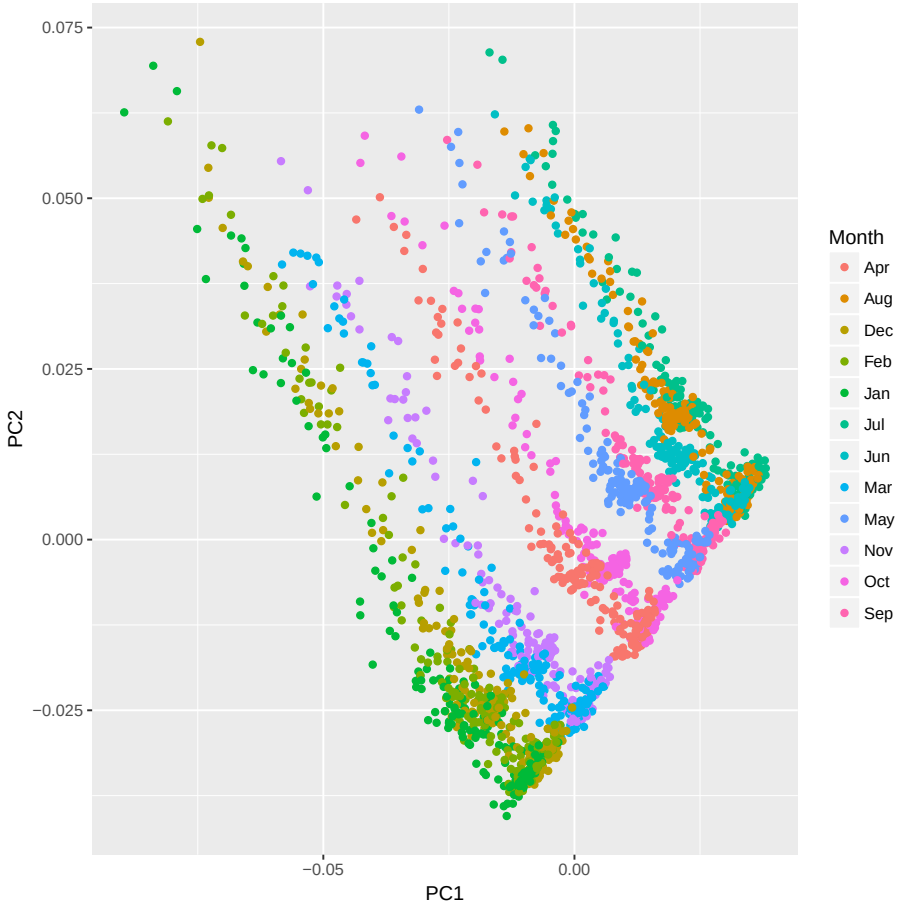


Figure 21: PCA Plot on Correlation Matrix with Month coloured



```
GTYear <- aggregate(. ~ Year,data=GTYear, FUN= "mean")
GTYear$YearGroup <-
cut(GTYear$Year, breaks = c(1849,1899,1939,1979,1999,2020),
labels = c("pre 1900","pre 1940","pre 1980",
"pre 2000","after 2000"))
```

```
# Redo previous analysis with yearly averages
PCAYear <- prcomp(GTYear[2:9],scale=TRUE)
summary(PCAYear)

## Importance of components%s:
##                           PC1    PC2     PC3     PC4     PC5     PC6
## Standard deviation     2.5265 1.1425 0.38174 0.25521 0.20372 0.16954
## Proportion of Variance 0.7979 0.1631 0.01822 0.00814 0.00519 0.00359
## Cumulative Proportion  0.7979 0.9611 0.97928 0.98742 0.99261 0.99620
##                            PC7     PC8
## Standard deviation     0.15572 0.07827
## Proportion of Variance 0.00303 0.00077
## Cumulative Proportion  0.99923 1.00000

PCAYear$rotation

##                 PC1       PC2         PC3         PC4         PC5
## L_Ave     0.3554530 0.3613321 -0.05044360  0.29848929 -0.2794324
## L_AveCI  -0.3682680 0.3063994 -0.15365629  0.16644851 -0.1575010
## L_Max     0.3526953 0.3083686 -0.66532373 -0.38757566  0.4227651
## L_MaxCI  -0.3395425 0.4104530  0.40763061 -0.35393838  0.2920075
## L_Min     0.3531396 0.3418129  0.42024091 -0.49627869 -0.3483652
## L_MinCI  -0.3502887 0.3840346  0.04215379  0.22224333  0.3915426
## LO_Ave    0.3450857 0.3979668  0.15917660  0.55024318  0.1095643
## LO_AveCI -0.3631183 0.2988301 -0.40163448 -0.09622665 -0.5885950
##                  PC6           PC7         PC8
## L_Ave     0.3261299537  0.6538040165  0.19886930
## L_AveCI   0.2327987972  0.0029660579 -0.80010755
## L_Max     0.0354899344 -0.0522254081 -0.07019235
## L_MaxCI   0.5202142853 -0.0615202830  0.25520196
## L_Min    -0.3837787905 -0.0006713813 -0.25868223
## L_MinCI  -0.6353229423  0.3381582994  0.08575749
## LO_Ave    0.0001920529 -0.6160751895  0.05367076
## LO_AveCI -0.1289965010 -0.2685500822  0.41601985
```

The first two principal components now account for 96% of the total variation in principal components. The first principal component is very close to the difference between the average of the temperature variables and the uncertainty variables. The second principal component is verl close to the average of all the variables. Figure 22 shows the plot of the first two principal components in the yearly averages with Year Group coloured. The contrasts in accuracy and level over time are not simple.

The next slides show the same analyses for the Temperature variables (Land Average, Maximum, Minimum and Land-Ocean Average) and Uncertainty Variables (variables as for Temperature). Figures 23 and 24 show the plots. The rotations, summaries and pictures emphasise the increasing accuracy over time as well as the increasing levels. The accuracy has not increased uniformly across variables as witnessed by the changing composition of the second principal component for accuracy and the plots. Beware facll conclusions from this data - such as the focus on the year 1998 by some commentators.

```
#  Separate out Temperature variables
PCAYearTempOnly <- prcomp(GTYear[,c(2,4,6,8)],scale = TRUE)
summary(PCAYearTempOnly)

## Importance of components%s:
##                           PC1     PC2     PC3     PC4
## Standard deviation     1.9509 0.33257 0.23860 0.16310
## Proportion of Variance 0.9515 0.02765 0.01423 0.00665
## Cumulative Proportion  0.9515 0.97912 0.99335 1.00000

PCAYearTempOnly$rotation

##              PC1         PC2        PC3         PC4
## L_Ave  0.5067688 -0.04411395  0.2963995 -0.80832336
## L_Max  0.4919729  0.82886262 -0.1784180  0.19777869
## L_Min  0.4984445 -0.46920519 -0.7253963  0.07210918
## LO_Ave 0.5026938 -0.30147506  0.5950752  0.54981560
```

Figure 22: PCA Plot on Yearly Averages with Year Group coloured

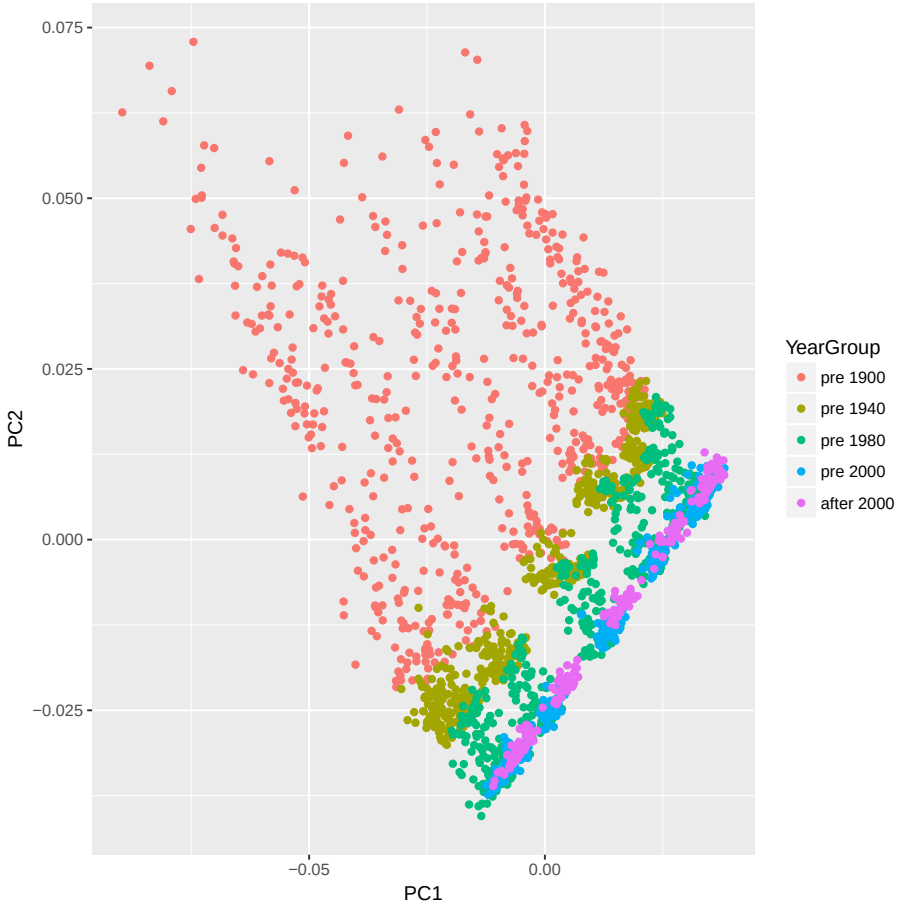
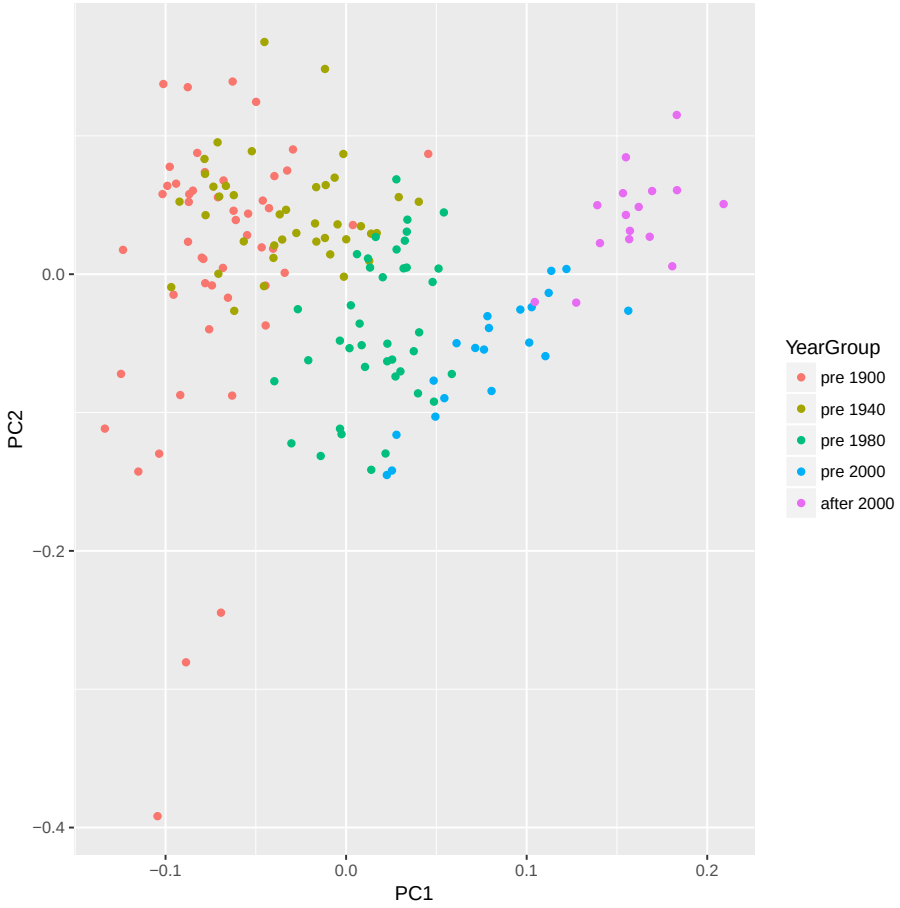


Figure 23: PCA Plot on Yearly Temperatures with Year Group coloured



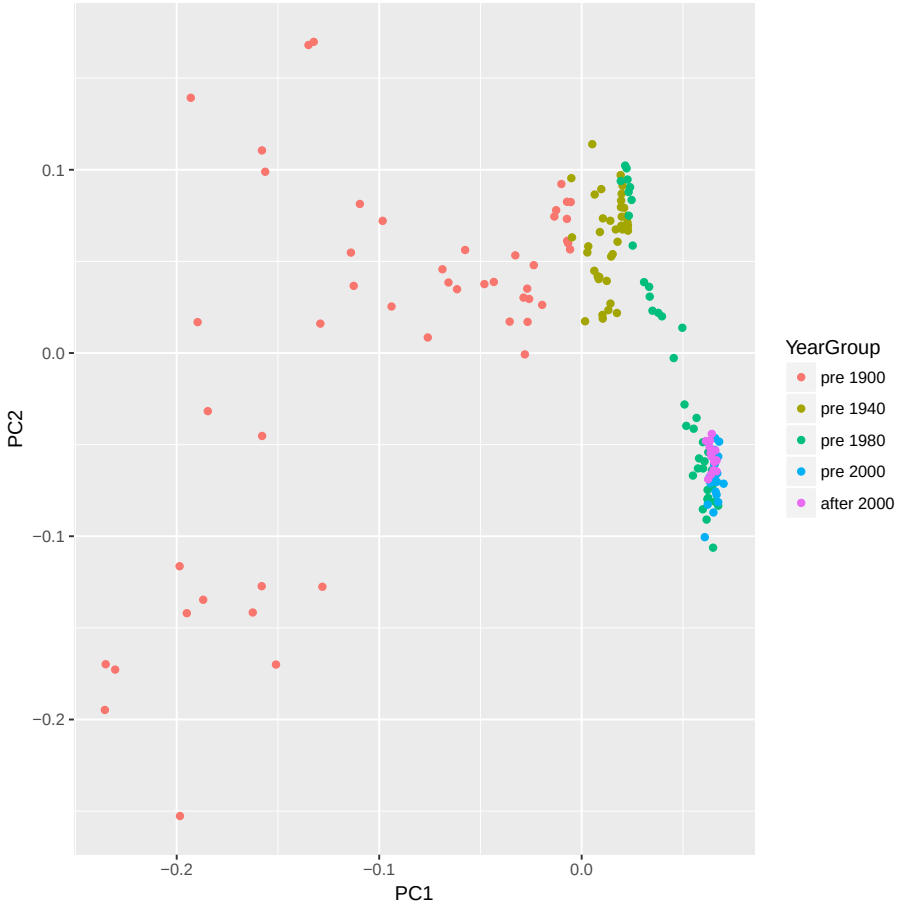

```
# Separate out Uncertainty variables - half width of CI's
PCAYearUncertaintyOnly <- prcomp(GTYear[,c(3,5,7,9)],scale = TRUE)
summary(PCAYearUncertaintyOnly)


## Importance of components%s:
##                           PC1     PC2     PC3    PC4
## Standard deviation     1.9655 0.29962 0.18672 0.1095
## Proportion of Variance 0.9658 0.02244 0.00872 0.0030
## Cumulative Proportion  0.9658 0.98829 0.99700 1.0000


PCAYearUncertaintyOnly$rotation


##                 PC1        PC2         PC3        PC4
## L_AveCI  -0.5051690  0.2537221 -0.08094938 -0.8208999
## L_MaxCI  -0.4948882 -0.7124824 -0.47971424  0.1316388
## L_MinCI  -0.5018124 -0.1776917  0.82883819  0.1721547
## LO_AveCI -0.4980706  0.6296187 -0.27631268  0.5283534
```

6.4 K-means clustering

Introduction

The Global Temperatures example has shown the way in which principal components can summarise a number of variables with several linear combinations of them. The plots of the first two principal components were coloured by

Figure 24: PCA Plot on Yearly Temperatures with Year Group coloured



the Year Grouping because both temperature and uncertainty varied over years. However, these groupings were somewhat arbitrary. Clustering techniques aim to group or *cluster* observations so that those in the same cluster have values of a number of variables that are close to each other. K Means clustering uses the Euclidean distances (one for each pair of observations) between the vector of variables in order to find the clusters. It is necessary to start with the desired number K of clusters, but a number of techniques are availble for helping find an appropriate number of clusters, for example *within-group sum of squares* and *silhouettes*.

Definition

Suppose the data set has n observations which are p-vectors $\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n$. The aim is to find K subsets $S_1, S_2, \cdots S_K$, the *clusters*, of $\{1, 2, \cdots, n\}$. The cluster S_i has the observation numbers that are in that cluster. Each observation must be in one cluster and there are no overlaps between clusters, that is $S_1, S_2, \cdots S_K$ partitions $\{1, 2, \cdots, n\}$. The clusters are chosen to minimise over all possibilities $S_1, S_2, \cdots S_k$ the *total within cluster sum of squares*:

$$\sum_{i=1}^{K} \sum_{j \in S_i} (\mathbf{x}_j - \bar{\mathbf{x}_i})^T (\mathbf{x}_j - \bar{\mathbf{x}_i}) \tag{2}$$

where $\bar{\mathbf{x}_i} = 1/|S_i| \sum_{\mathbf{x} \in S_i} \mathbf{x}$ is the arithmetic mean of the observations in cluster S_i.

Computation

It is a very difficult task computationally to minimise over all partitions with k components. However, a fast approximate algorithm, the Lloyd-Forgy algoriothm is available and used in R. The Lloyd-Forgy algoriothm is iterative and can be started with different random starts to check on the validity of the result. A fit is good if the *total within cluster sum of squares* is small. A plot of the *total within cluster sum of squares* versus the cluster number often shows an initial steep descent followed by a leveling off. A common choice for the number of clusters would be the the start of the leveling off.

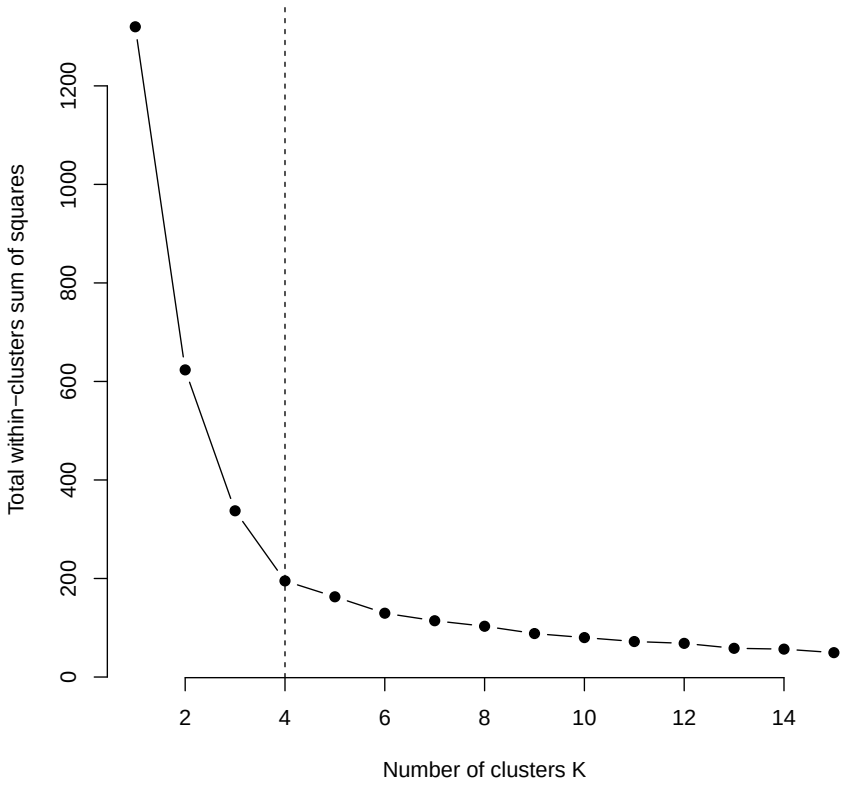
Silhouettes

Another method to choose the number of clusters uses the idea of silhouettes introduced by Rousseeuw in 1987. Let $a(i)$ be the average squared distance of $\mathbf{x}_i$ from the points in its cluster. Let $b(i)$ be the lowest (amongst all of the other $K-1$ clusters) of the average squared distance of $\mathbf{x}_i$ from the points in another cluster. The silhouette $s(i)$ for observation i is $1 - a(i)/b(i)$ if $a(i) < b(i)$ and $b(i)/a(i) - 1$ if $a(i) \geq b(i)$, so $-1 \leq s(i) \leq 1$. Values of the silhouette close to 1 indicate the observation is a good fit to the cluster and negative values indicate a poor fit. The number of clusters can thus be chosen to minimise the average silohouette.

6.5 Clustering Example: Global Temperatures

Plots of the clusters generally use the first two principal components, so it is necessary to standardise the data if the principal comoponents are to be scaled.

Figure 25: Total within cluster sum of squares vs. K

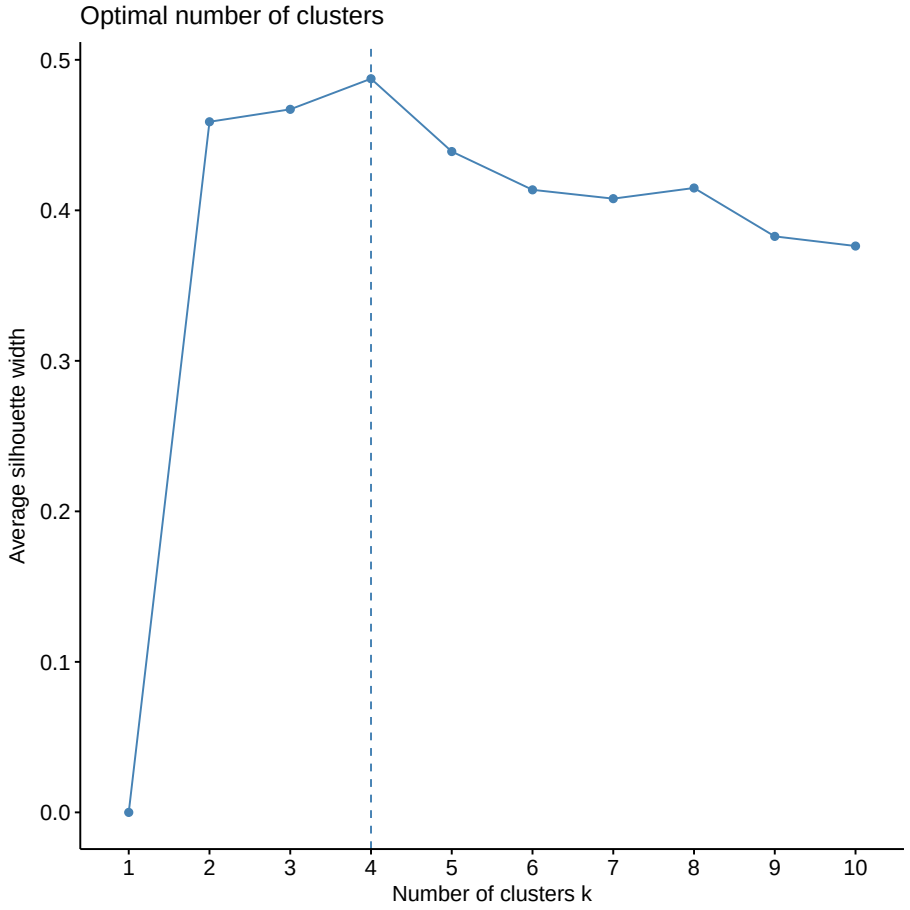


This can be done use the `scale` command in R. The other commands are to produce the plot in Figure 25 of *total within cluster sum of squares* from (2) versus the number of clusters k.

```
library(cluster)
k.max <- 15 # Maximal number of clusters
GTYearScale <- scale(GTYear[,2:9])
wss <- sapply(1:k.max,
function(k){kmeans(GTYearScale, k, nstart=10 )$tot.withinss})
plot(1:k.max, wss,
type="b", pch = 19, frame = FALSE,
xlab="Number of clusters K",
ylab="Total within-clusters sum of squares")
abline(v = 4, lty =2)
```

The leveling off at the number of clusters being 4 indicates this would be a good choice for the number of clusters. The average silohouette method produces the plot in Figure 26. The average silohouette is maximum for $K = 4$ confirming the previous choice.

Figure 26: Average silohouette over all points vs. K




```
library(factoextra)
library(NbClust)
library(fpc)
fviz_nbclust(GTYearScale, kmeans, method = c("silhouette"))
```

Having decided that 4 clusters is appropriate, that is $K = 4$, the `kmeans` command produces the clusters. The `autoplot` command on the result produces the plot in Figure 27. This plot shows the first two principal components of each of the 8 vectors in the standardised data set. The plot is similar to Figure 22 but the colours are done with the cluster number rather than the Year Group.

```
KYear <- kmeans(GTYearScale,4)
autoplot(KYear,data=GTYearScale)
```

To check the validity of the clusters, a silohouette plot shows the silohouette value for each observation (in this case each of the years 1850 to 2015) organised by clusters. In this case, there are no negative silohouette values and most

Figure 27: First two principal components coloured by cluster

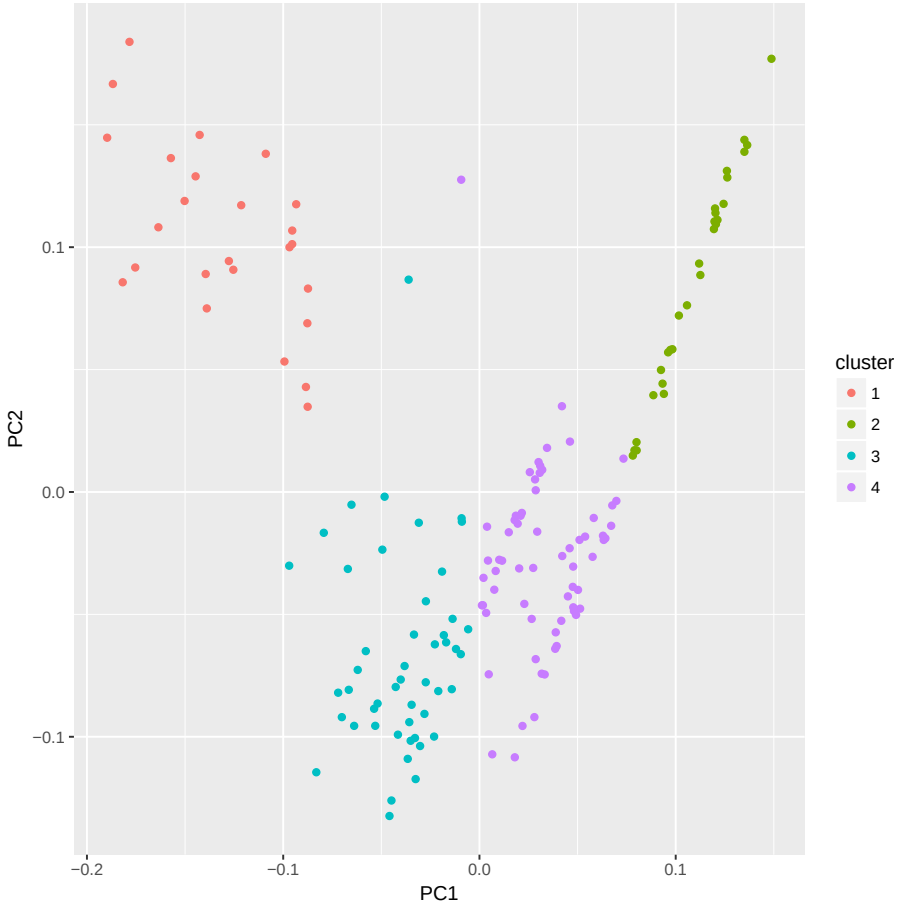
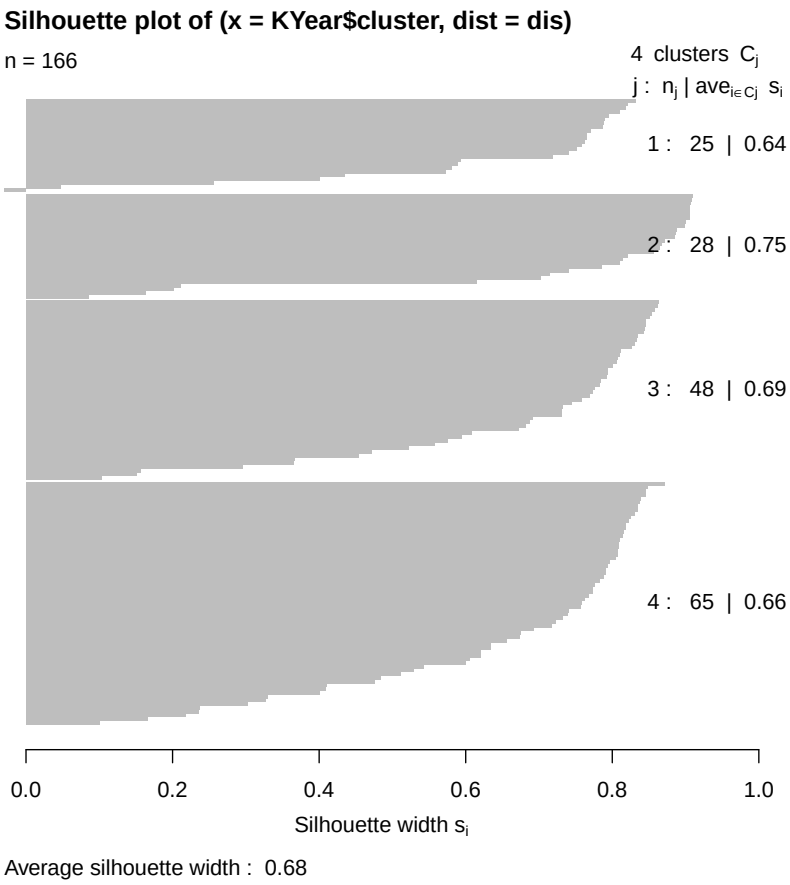


Figure 28: Individual Silohouette values organised by cluster



of the years in each cluster have a moderately silohouette value. This adds weight to the validity of the clusters, as illustrated in Figure 28.

```
dis = dist(GTYearScale)^2
sil = silhouette (KYear$cluster, dis)
plot(sil)
```

To compare the clusters to time, a plot of cluster number versus year is useful.

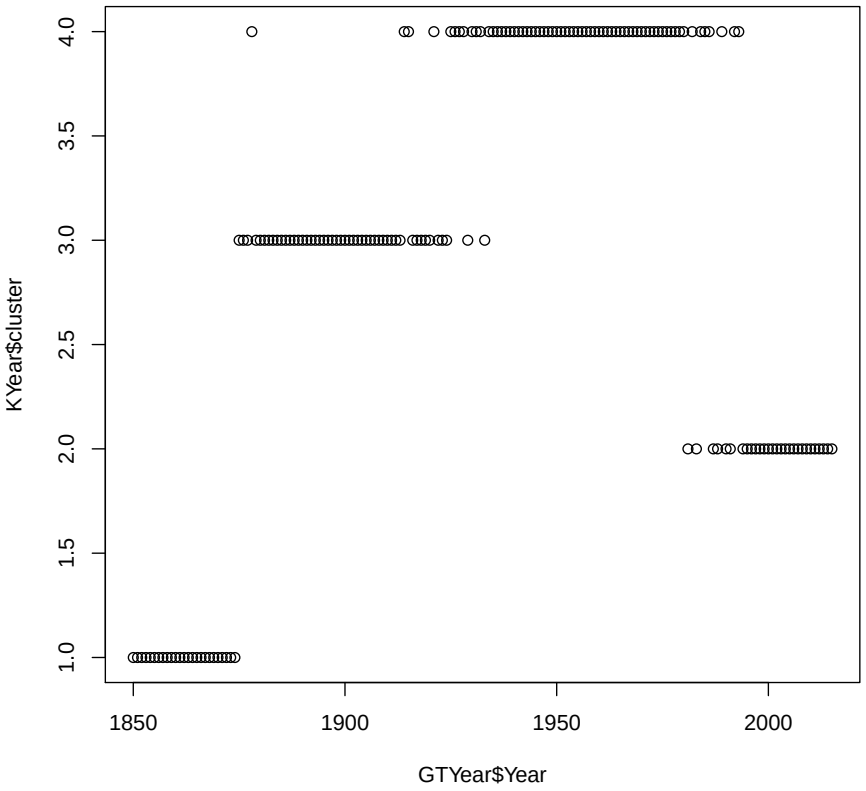
```
plot(GTYear$Year,KYear$cluster)
```

6.6 Coda

Conclusions

The clusters from the k-means analysis of the Global Temperatures data confirm the importance of time in describing the variation in the temperature

Figure 29: Cluster numbers versus year



values and their associated uncertainties. Whilst these analyses can't attribute cause in the increase in temperature, they do make it hard to hold that temperatures are constant over the period since 1850. For the machine learners, "Lies, Damned Lies and Statistics" might well be replaced by "Stories, Evidence and Statistical Methods". Statistical methods, both theory and practice, are vital in uncovering stories that guide productive action!