STAT 513/413: Lecture 9
The “other” random numbers: distributions

(transforming uniformly distributed numbers to something else)

Rizzo 3.1, 3.2, 3.3, 3.4, 3.5

Almost all of them are somewhere out there

Naming convention in R

`d`*distrib*`(x,...)` - density function

`p`*distrib*`(x,...)` - cumulative distribution function

`q`*distrib*`(p,...)` - quantile function

`r`*distrib*`(n,...)` - random numbers

If no other parameter is specified, it assumes the "standard" version of the distribution

For instance, `rnorm(100)` generates 100 random numbers from normal distribution with mean 0 and standard deviation 1

And `runif(50)` generates 50 random numbers from the "standard random generator": random numbers with uniform distribution on $(0, 1)$

And almost all of the distributions can be found somewhere out there, if not in base R - but beware: not all packages are made equal!

Discrete distributions

For discrete distributions with finitely many outcomes, `sample()` is a way to go - but beware whether it is sampling with replacement or without (default)!

```
> sample(1:20,10)
 [1]  9 18 13 15 19  4 20  3  6 16
> sample(1:5,10)
Error in sample.int(length(x), size, replace, prob) :
  cannot take a sample larger than the population when 'replace = FALSE'
> sample(1:5,10,replace=TRUE)
 [1] 2 3 3 5 4 3 1 4 3 3
> sample(1:20,10,replace=TRUE)
 [1]  7  1 15  2  7 17 10 19 15 19
> sample(c(0,1),20,replace=TRUE,prob=c(0.1,0.9))
 [1] 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0
```

1. Inversion method

Method: apply quantile function of the desired distribution on the standard uniform random numbers

Quantile function: the inverse to the cumulative distributioin function, $Q(u) = F^{-1}(u)$

Standard uniform random numbers: uniform on $(0, 1)$, `runif()`

Example: exponential distribution, $f(x) = \lambda e^{-\lambda x}$

$F(x) = 1 - e^{-x}$ $\qquad Q(u) = -\log(1 - u)$

Generator: $-\log(1 - U)$ where U is uniform on $(0, 1)$

Inverse method works well, but sometimes may be hard to come by

Example: normal distribution

$$f(x) = \frac{1}{2\pi\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

There is no closed form F, and neither that of Q - but we can compute Q numerically - in R it is function `qnorm()`

So for $\mu = 1$ and $\sigma = 2$ we get

```
> y=1+2*qnorm(runif(10000))
```

although we could use also

```
> z=qnorm(runif(10000),1,2)
```

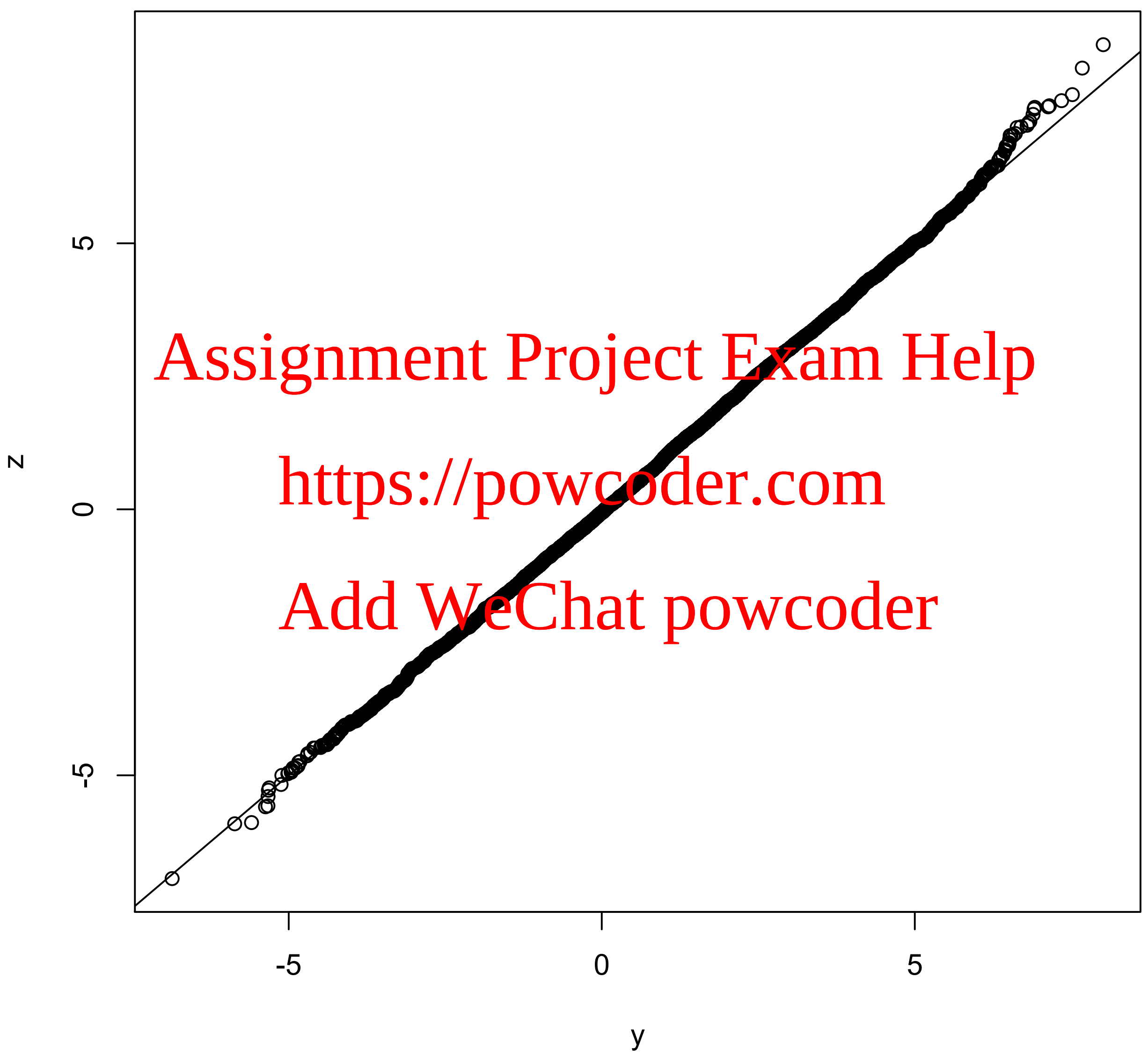
Let us check that on a qqplot; the special version, when normal distribution is checked, is

```
> qqnorm(y)
> qqline(y)
```

If we want to check whether two version above come to the same, we do (See Lecture 7 again if necessary)

```
> qqplot(y,z)
> abline(0,1)
```

Seems like it works



Why does inversion work?

A very short, but illuminating proof: we show what is the cumulative distribution function of $Q(U) = F^{-1}(U)$. For simplicity, we may assume that F is strictly increasing, $F(u) < F(v)$, whenever $u < v$ - so there is no problem with inverting F and so on. (The principle, however, works in greater generality)

$$P[Q(U) \leqslant x] = P[F^{-1}(U) \leqslant x]$$

and apply F on both sides of the inequality $F^{-1}(U) \leqslant x$; as F is nondecreasing, the inequality is preserved, which means that

$$P[F^{-1}(U) \leqslant x] = P[F(F^{-1}(U)) \leqslant F(x)]$$

Now $F(F^{-1}(U)) = U$, by the definition of $F^{-1}(u)$, and

$$P[F^{-1}(U) \leqslant x] = P[F(F^{-1}(U)) \leqslant F(x)] = P[U \leqslant F(x)] = F(x)$$

as the last probability is for U uniform on $(0,1)$ equal to $F(x)$. We obtained that $P[F^{-1}(U) \leqslant x] = F(x)$, that is, the cumulative distribution function $F^{-1}(U)$ of the generated random number is equal to the desired $F(x)$ - which was desired

The default for `rnorm()`: inversion

To learn more about random number generators in R:

```
> ?RNG
> RNGkind()
[1] "Mersenne-Twister" "Inversion"
```

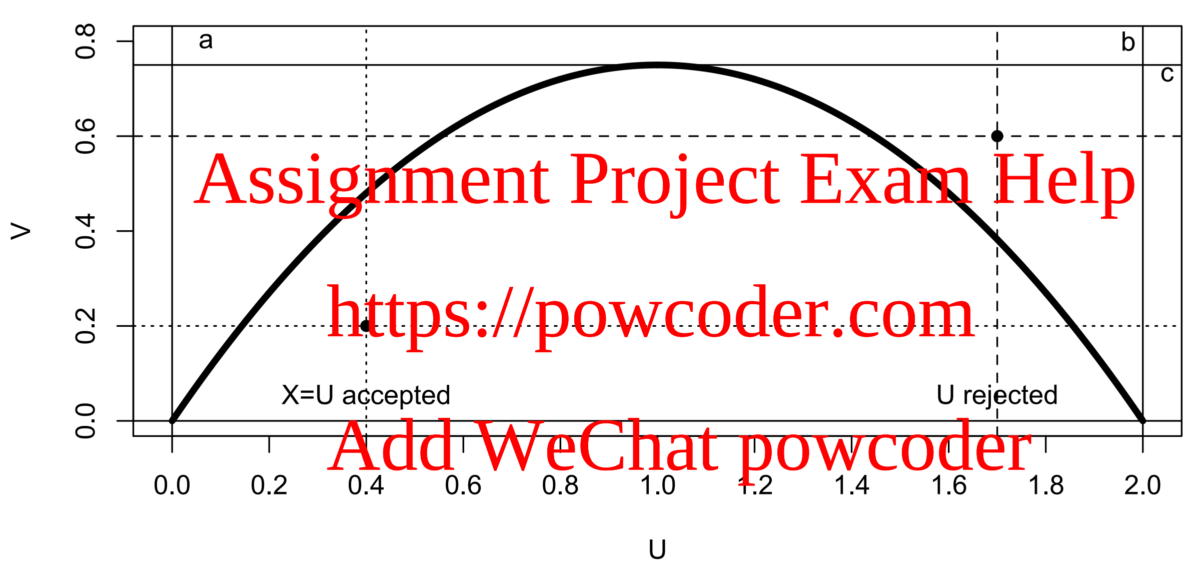
These are defaults. The second one specifies the way of generating random numbers with normal distribution - we will discuss that later. ... `normal.kind` can be `"Kinderman-Ramage"`, `"Buggy Kinderman-Ramage"` (not for `set.seed`), `"Ahrens-Dieter"`, `"Box-Muller"`, `"Inversion"` (the default), or `"user-supplied"`. (For inversion, see the reference in `qnorm`.) The Kinderman-Ramage generator used in versions prior to 1.7.0 (now called `"Buggy"`) had several approximation errors and should only be used for reproduction of old results. The `"Box-Muller"` generator is stateful (sic!) as pairs of normals are generated and returned sequentially. The state is reset whenever it is selected (even if it is the current normal generator) and when kind is changed.

2. Acceptance-Rejection method

Example: obtain random numbers with a density

$$f(x) = \frac{3}{4}x(2-x), \ \ x \in [0,2]$$

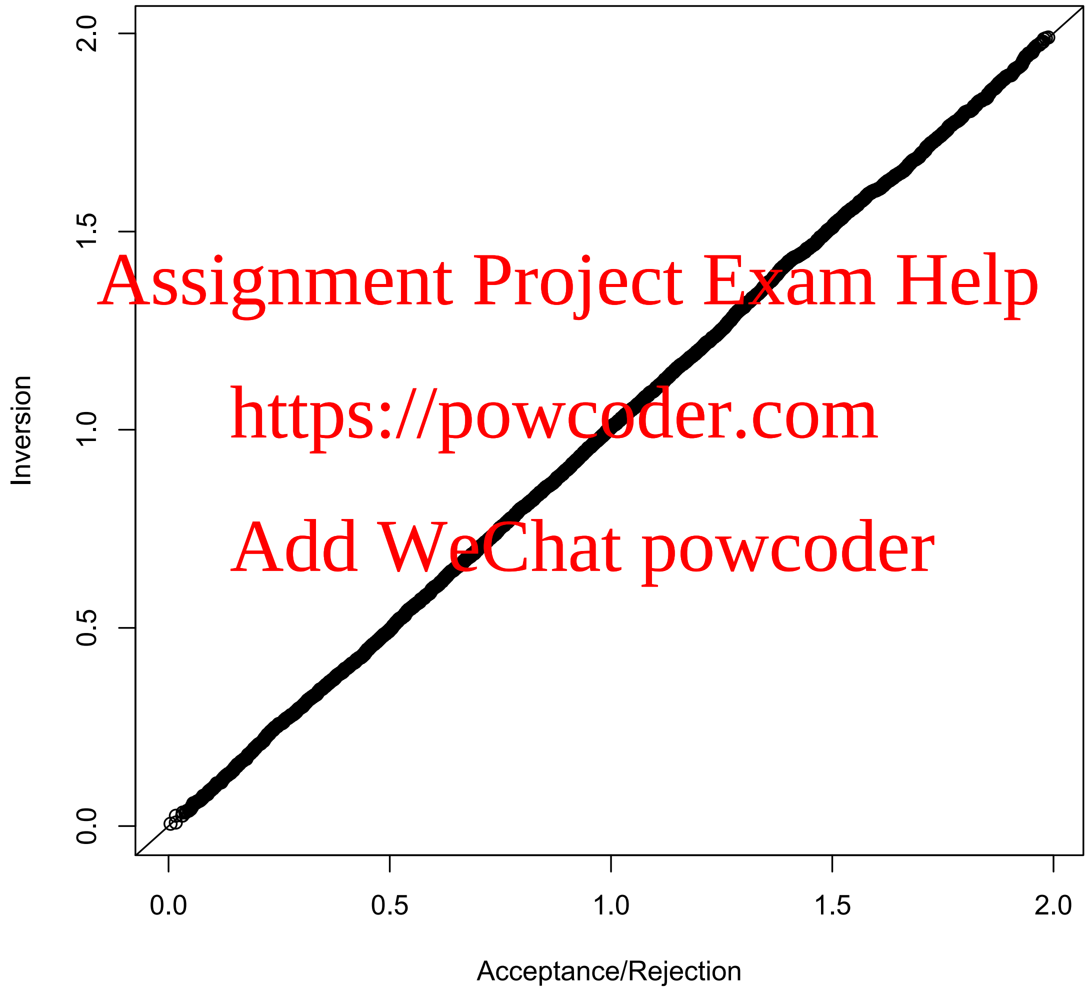
x is between a=0 and b=2, y between 0 and c=0.75



Important: the argument of the density is within $[a,b]$, the values within $[0,c]$. The desired random number is produced as follows:

1. Generate U uniform on $[a,b]$ and V uniform on $[0,c]$
2. if $f(U) \geqslant V$, take $X = U$ (accept)
3. if not, repeat (that is, reject, return to 1, and repeat until once X is obtained, that is, accepted)

It works!



Theoretical justification

Notation: a bit tricky, while otherwise straightforward. Let the density of *accepted* U be $h(x)$. We would like to show that it is actually $f(x)$, the density we would like the random numbers to have

The density of V is $f_V(x) = \dfrac{1}{c}$ on $[0, c]$

The density of U is $f_U(x) = \dfrac{1}{b-a}$ on $[a, b]$

the latter is not the same as the conditional density we are to start with, but for simplicity we also use f_U:

$$h(x) = f_U(x|x \text{ accepted}) = \frac{f_U(x)\mathbb{P}[x = U \ \& \ f(U) \leqslant V]}{\displaystyle\int f_U(z)\mathbb{P}[z = U \ \& \ f(U) \leqslant V]\, dz}$$

$$= \frac{f_U(x)\mathbb{P}[f(x) \leqslant V]}{\displaystyle\int f_U(z)\mathbb{P}[f(z) \leqslant V]\, dz} = \frac{\dfrac{1}{b-a} f(x) \dfrac{1}{c}}{\displaystyle\int \dfrac{1}{b-a} f(z) \dfrac{1}{c}\, dz} = \frac{f(x)}{\displaystyle\int f(z)\, dz}$$

$$= \frac{f(x)}{1} = f(x) \qquad \text{as was desired}$$

3. Transformation methods

When the desired distribution is a distribution of transformed variable with other distribution, and the transformation is not difficult - then why not to go this way

Example: Chi-square - sum of k squares of standard normal

Example: Box-Muller transform. If U, V are independent random variables with uniform distribution on $(0, 1)$, then

$\sqrt{-2 \log U} \cos(2\pi V)$ and $\sqrt{-2 \log U} \sin(2\pi V)$

are independent random variables with standard normal distribution

4. What else?

As a special case of transformation methods, one can consider also convolutions (that is, sums) and mixtures

Example. For instance, it was once popular to generate random numbers with normal distribution by a sum of k random numbers with uniform distribution on $(0, 1)$, Usually, k was set to be 12 or 48, as in such case the sum had convenient variance (you may want to exercise yourself to see why). The motivation came from the central limit theorem: with growing k, the distribution of sums of identically distributed random variables is better and better approximated by normal distribution.

In fact, already for $k = 12$ the approximation is not that bad - in the centre (this is why the limit theorem is called “central”). But not on tails: sum of 12 uniform random numbers cannot be larger than 12 and less than 0. Thus, nowadays other methods are preferred for generating normal random numbers

Another example: Poisson-Gamma mixture

```
> lambda = rgamma(6,4,3)
> lambda
[1] 1.4796001 3.4630581 0.4897232 0.5519801 0.4214378 2.1400598
> set.seed(1)
> rpois(6,lambda)
[1] 1 3 0 2 0 4
> set.seed(1)
> rpois(6,lambda[1])
[1] 1 1 2 3 0 3
> set.seed(1)
> x=NULL
> for (k in (1:length(lambda))) x[k] = rpois(1,lambda[k])
> x
[1] 1 3 0 2 0 4
```

Beware: it usually works, but better check...

Finally, Rizzo again: 3.1, 3.2, 3.3, 3.4, 3.5