20 Simulation

20.1 Générer des nombres aléatoires

Regarder une vidéo de cette section

La simulation est un sujet important (et important) à la fois pour les statistiques et pour divers autres domaines dans lesquels il est nécessaire d'introduire le caractère aléatoire. Parfois, vous souhaitez implémenter une procédure statistique nécessitant la génération ou l'échantillonnage de nombres aléatoires (chaîne de Markov Monte Carlo, le bootstrap, les forêts aléatoires, l'ensachage) et parfois, vous souhaitez simuler un système et des générateurs de nombres aléatoires peuvent être utilisés pour modéliser des entrées aléatoires.

R est fourni avec un ensemble de générateurs de nombres pseuodo-aléatoires qui vous permettent de simuler à partir de distributions de probabilités bien connues telles que la normale, la loi de Poisson et le binôme. Quelques exemples de fonctions pour les distributions de probabilité en R

- `rnorm` : générer des variables normales aléatoires avec une moyenne et un écart type donnés
- `dnorm` : évaluer la densité de probabilité normale (avec une moyenne / SD donnée) en un point (ou un vecteur de points)
- `pnorm` : évaluer la fonction de distribution cumulative pour une distribution normale
- `rpois` : générer des variables de Poisson aléatoires avec un taux donné

Pour chaque distribution de probabilité, il y a généralement quatre fonctions disponibles qui commencent par «r», «d», «p» et «q». La fonction «r» est celle qui simule réellement les nombres aléatoires à partir de cette distribution. Les autres fonctions sont préfixées par un

- `d` pour la densité
- `r` pour la génération de nombres aléatoires
- `p` pour la distribution cumulative
- `q` for quantile function (inverse cumulative distribution)

If you're only interested in simulating random numbers, then you will likely only need the "r" functions and not the others. However, if you intend to simulate from arbitrary probability distributions using something like rejection sampling, then you will need the other functions too.

Probably the most common probability distribution to work with the is the Normal distribution (also known as the Gaussian). Working with the Normal distributions requires using these four functions

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Here we simulate standard Normal random numbers with mean 0 and standard deviation 1.

```
> ## Simulate standard Normal random numbers
> x <- rnorm(10)
> x
 [1]  0.01874617 -0.18425254 -1.37133055 -0.59916772  0.29454513
 [6]  0.38979430 -1.20807618 -0.36367602 -1.62667268 -0.25647839
```

We can modify the default parameters to simulate numbers with mean 20 and standard deviation 2.

```
> x <- rnorm(10, 20, 2)
> x
 [1] 22.20356 21.51156 19.52353 21.97489 21.48278 20.17869 18.09011
 [8] 19.60970 21.85104 20.96596
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  18.09   19.75   21.22   20.74   21.77   22.20 
```

If you wanted to know what was the probability of a random Normal variable of being less than, say, 2, you could use the `pnorm()` function to do that calculation.

```
> pnorm(2)
[1] 0.9772499
```

You never know when that calculation will come in handy.

20.2 Setting the random number seed

When simulating any random numbers it is essential to set the *random number seed*. Setting the random number seed with `set.seed()` ensures reproducibility of the sequence of random numbers.

For example, I can generate 5 Normal random numbers with `rnorm()` .

```
> set.seed(1)
> rnorm(5)
[1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078
```

Note that if I call `rnorm()` again I will of course get a different set of 5 random numbers.

```
> rnorm(5)
[1] -0.8204684  0.4874291  0.7383247  0.5757814 -0.3053884
```

If I want to reproduce the original set of random numbers, I can just reset the seed with `set.seed()` .

```
> set.seed(1)
> rnorm(5)    ## Same as before
[1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078
```

In general, you should **always set the random number seed when conducting a simulation!** Otherwise, you will not be able to reconstruct the exact numbers that you produced in an analysis.

It is possible to generate random numbers from other probability distributions like the Poisson. The Poisson distribution is commonly used to model data that come in the form of counts.

```
> rpois(10, 1)    ## Counts with a mean of 1
 [1] 0 0 1 1 2 1 1 4 1 2
> rpois(10, 2)    ## Counts with a mean of 2
 [1] 4 1 2 0 1 1 0 1 4 1
> rpois(10, 20)   ## Counts with a mean of 20
 [1] 19 19 24 23 22 24 23 20 11 22
```

20.3 Simulating a Linear Model

Watch a video of this section

Simulating random numbers is useful but sometimes we want to simulate values that come from a specific *model*. For that we need to specify the model and then simulate from it using the functions described above.

Suppose we want to simulate from the following linear model

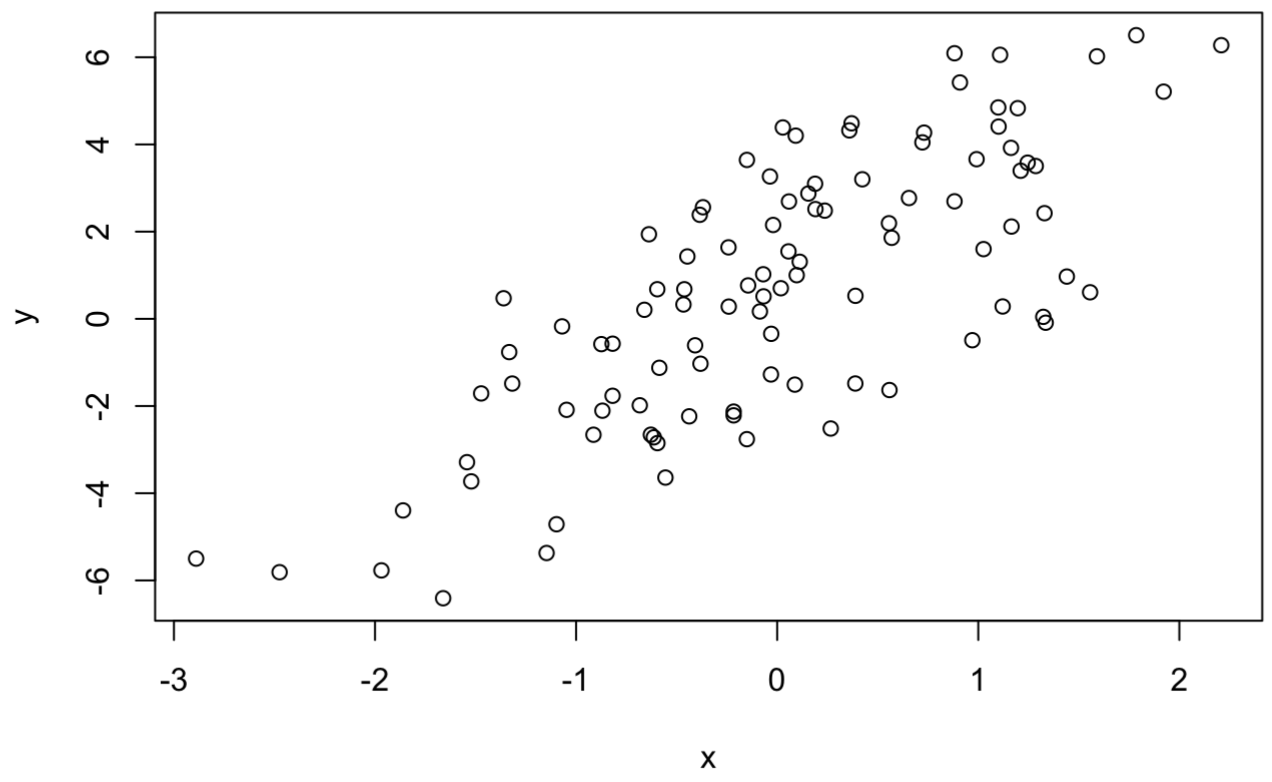
$$y = \beta_0 + \beta_1 x + \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0, 2^2)$. Assume $x \sim \mathcal{N}(0, 1^2)$, $\beta_0 = 0.5$ and $\beta_1 = 2$. The variable `x` might represent an important predictor of the outcome `y`. Here's how we could do that in R.

```
> ## Always set your seed!
> set.seed(20)
> 
> ## Simulate predictor variable
> x <- rnorm(100)
> 
> ## Simulate the error term
> e <- rnorm(100, 0, 2)
> 
> ## Compute the outcome via the model
> y <- 0.5 + 2 * x + e
> summary(y)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
-6.4084 -1.5402  0.6789  0.6893  2.9303  6.5052 
```

We can plot the results of the model simulation.

```
> plot(x, y)
```

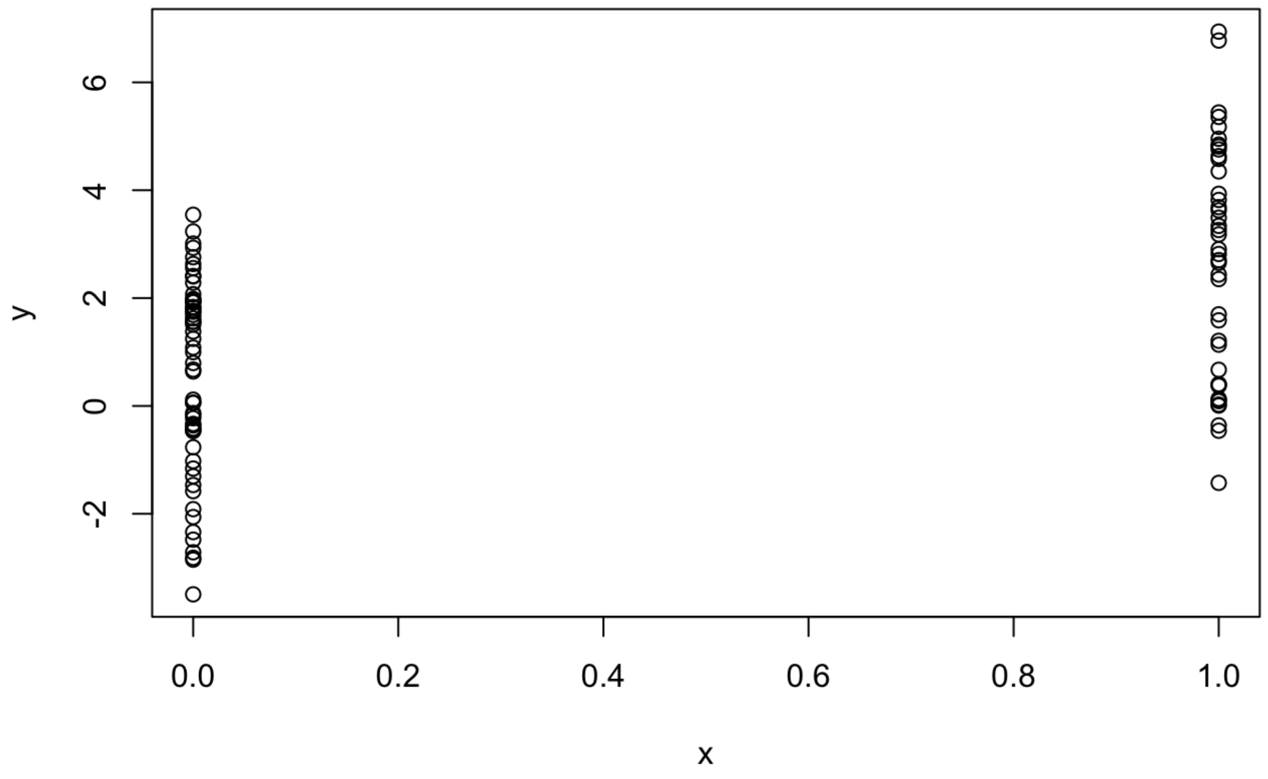


What if we wanted to simulate a predictor variable `x` that is binary instead of having a Normal distribution. We can use the `rbinom()` function to simulate binary random variables.

```
> set.seed(10)
> x <- rbinom(100, 1, 0.5)
> str(x)    ## 'x' is now 0s and 1s
 int [1:100] 1 0 0 1 0 0 0 0 1 0 ...
```

Then we can procede with the rest of the model as before.

```
> e <- rnorm(100, 0, 2)
> y <- 0.5 + 2 * x + e
> plot(x, y)
```



We can also simulate from *generalized linear model* where the errors are no longer from a Normal distribution but come from some other distribution. For examples, suppose we want to simulate from a Poisson log-linear model where

$$Y \sim Poisson(\mu)$$

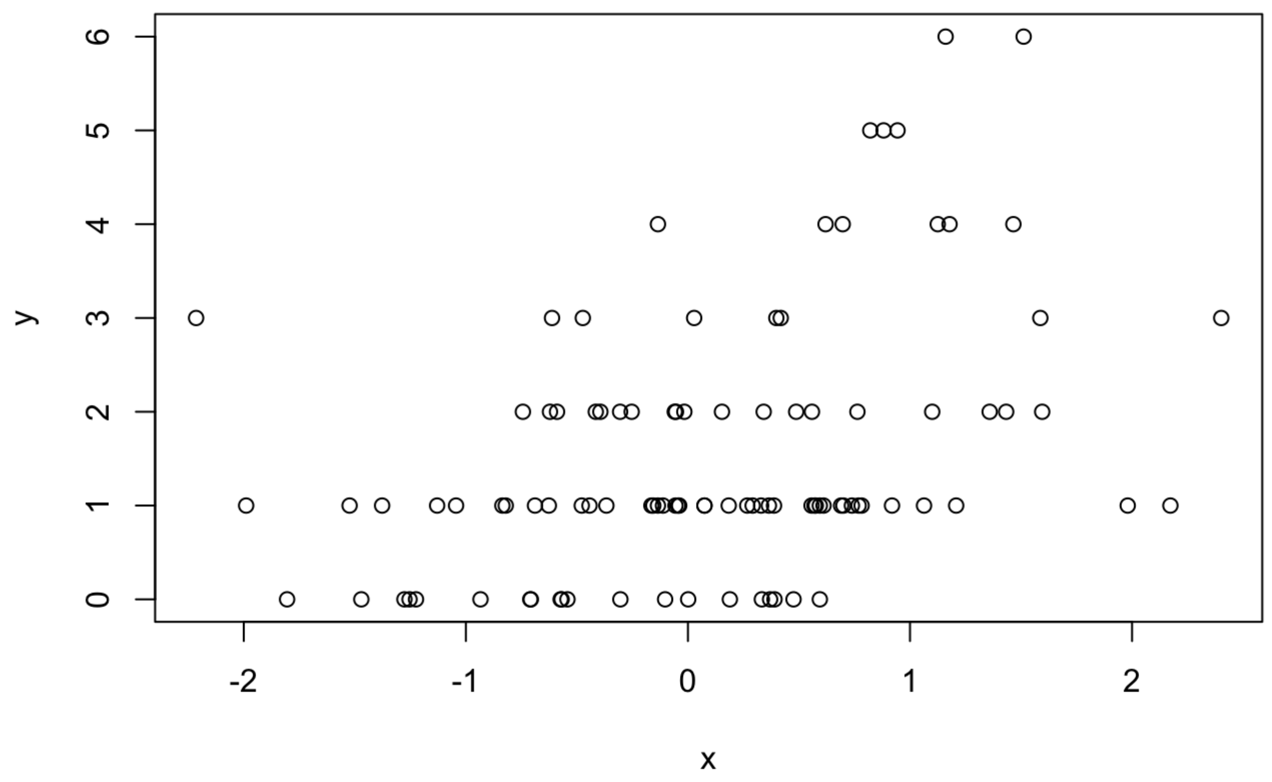
$$\log \mu = \beta_0 + \beta_1 x$$

and $\beta_0 = 0.5$ and $\beta_1 = 0.3$. We need to use the `rpois()` function for this

```
> set.seed(1)
>
> ## Simulate the predictor variable as before
> x <- rnorm(100)
```

Now we need to compute the log mean of the model and then exponentiate it to get the mean to pass to `rpois()`.

```
> log.mu <- 0.5 + 0.3 * x
> y <- rpois(100, exp(log.mu))
> summary(y)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   0.00    1.00    1.00    1.55    2.00    6.00 
> plot(x, y)
```



You can build arbitrarily complex models like this by simulating more predictors or making transformations of those predictors (e.g. squaring, log transformations, etc.).

20.4 Random Sampling

Watch a video of this section

The `sample()` function draws randomly from a specified set of (scalar) objects allowing you to sample from arbitrary distributions of numbers.

```
> set.seed(1)
> sample(1:10, 4)
[1] 9 4 7 1
> sample(1:10, 4)
[1] 2 7 3 6
>
> ## Doesn't have to be numbers
> sample(letters, 5)
[1] "r" "s" "a" "u" "w"
>
> ## Do a random permutation
> sample(1:10)
 [1] 10  6  9  2  1  5  8  4  3  7
> sample(1:10)
 [1]  5 10  2  8  6  1  4  3  9  7
>
> ## Sample w/replacement
> sample(1:10, replace = TRUE)
 [1]  3  6 10 10  6  4  4 10  9  7
```

To sample more complicated things, such as rows from a data frame or a list, you can sample the indices into an object rather than the elements of the object itself.

Here's how you can sample rows from a data frame.

```
> library(datasets)
> data(airquality)
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

Now we just need to create the index vector indexing the rows of the data frame and sample directly from that index vector.

```
> set.seed(20)
>
> ## Create index vector
> idx <- seq_len(nrow(airquality))
>
> ## Sample from the index vector
> samp <- sample(idx, 6)
> airquality[samp, ]
    Ozone Solar.R Wind Temp Month Day
107    NA      64 11.5   79     8  15
120    76     203  9.7   97     8  28
130    20     252 10.9   80     9   7
98     66      NA  4.6   87     8   6
29     45     252 14.9   81     5  29
45     NA     332 13.8   80     6  14
```

Other more complex objects can be sampled in this way, as long as there's a way to index the sub-elements of the object.

20.5 Summary

- Drawing samples from specific probability distributions can be done with "r" functions
- Standard distributions are built in: Normal, Poisson, Binomial, Exponential, Gamma, etc.
- La `sample()` fonction peut être utilisée pour dessiner des échantillons aléatoires à partir de vecteurs arbitraires
- Définir la valeur de départ du générateur de nombres aléatoires via `set.seed()` est essentiel pour la reproductibilité