"Random problems with R", ovvero: quanto andava bene la (vecchia) funzione sample?

Al cuore di una simulazione in ambito probabilistico (ma non solo) c'è (dovrebbe esserci) un meccanismo "genuinamente" casuale.

Questa affermazione ci porterebbe a dover prima dare una definizione di che cosa sia il *caso*, ma ci guardiamo bene dal farlo, anche perché, con ogni probabilità, sconfineremmo in ambito filosofico. Potremmo dire che, se crediamo che la meccanica quantistica sia una teoria *completa*,[1] dove le probabilità sono *non epistemiche*, allora un meccanismo "genuinamente" casuale sarebbe qualcosa che coinvolge un fenomeno di tipo quantistico, completamente imprevedibile (come la determinazione dello *spin* di un elettrone o della polarizzazione di un fotone). Si tratta comunque di procedure ancora accessibili a pochi ricercatori. Tutti gli altri devono per il momento "accontentarsi" di metodi in grado di generare *numeri pseudo-casuali*, cioè numeri che, pur generati impiegando un algoritmo deterministico, manifestano un insieme di proprietà statistiche tipiche dei numeri generati attraverso un meccanismo genuinamente casuale.[2]

R ha da sempre offerto più di un algoritmo in grado di generare numeri pseudo-casuali; la versione 4.0.5 (del 31/3/2021) ne offre 7, più un ottavo che può essere definito dall'utente (per ulteriori dettagli si veda l'*help* della funzione `RNGkind`). Una delle funzioni fondamentali nelle simulazioni è `sample`, che restituisce un campione estratto (con o senza rimessa) da una "popolazione" i cui elementi sono contenuti in un vettore che rappresenta il primo argomento di `sample`.[3] Molte delle simulazioni eseguite nel nostro libro sono basate proprio su `sample`.

È quindi facilmente immaginabile lo sconcerto procurato nella comunità degli utilizzatori di R dall'articolo *Random problems with R* pubblicato da Kellie Ottoboni e Philip B. Stark il 18 settembre 2018 su arXiv[4] e rilanciato

[1] Einstein non è mai stato d'accordo, e non è l'unico.

[2] Per ulteriori dettagli rimandiamo al Capitolo 1 del nostro *Problemi ed esperimenti di statistica con* R.

[3] In effetti, il primo argomento di questa funzione può essere un vettore (di uno o più elementi) oppure un intero positivo.

[4] `https://arxiv.org/abs/1809.06520`. La versione (riveduta) attualmente presente è del 13 novembre 2018.

da un *post* di Carl Boettiger[5] sulla lista pubblica R-devel[6] dedicata a domande e a discussioni sullo sviluppo del codice di R.

Mentre rimandiamo il lettore che voglia approfondire l'argomento al lavoro citato e alla discussione seguita al *post* di Carl Boettiger, qui ricordiamo soltanto che Ottoboni e Stark hanno messo in evidenza una situazione nella quale i risultati prodotti (in particolare, ma non esclusivamente) da `sample` sono in modo del tutto evidente lontani da quanto ci si aspetterebbe. Sembra che questo comportamento compaia solo quando la popolazione dalla quale si estrae il campione è molto (ma davvero *molto*) grande. Per capire quanto grande, possiamo vedere questo esempio proposto da Duncan Murdoch in un suo *post*[7] e che noi abbiamo riprodotto (con risultati numericamente diversi, ma sostanzialmente simili) con la versione 3.3.2 di R:

```
> m <- (2/5)*2^32
> x <- sample(m, 1000000, replace = TRUE)
> table(x %% 2)

0      1
399592 600408
```

Dal momento che `m` è un numero dispari, la vera proporzione di numeri pari e di numeri dispari estratti dovrebbe essere 0.5; il risultato è straordinariamente lontano da quello atteso.

A seguito del *post* di Carl Boettiger, sulla lista R-devel si è "scatenata" una vera e propria *querelle* che ha avuto per protagonisti principali da una parte Duncan Murdoch, che, pur riconoscendo il *bug* (non avrebbe potuto fare altrimenti), avrebbe voluto mantenere la vecchia versione di `sample` (sottolineando l'eccezionalità della situazione in cui questo *bug* si appalesava) e Philip Stark (ma non solo) dall'altra, che sottolineava puntualmente come questa circostanza poteva non essere poi così eccezionale. Qui di seguito riportiamo alcuni "scambi di opinione" fra i due ricercatori:

- Murdoch: I think the analyses are correct, but I doubt if a change to the default is likely to be accepted as it would make it more difficult to reproduce older results.[8]

[5] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076817.html`.
[6] `https://stat.ethz.ch/mailman/listinfo/r-devel`.
[7] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076827.html`.
[8] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076818.html`

- Stark: The 53 bits only encode at most 2^{32} possible values, because the source of the float is the output of a 32-bit PRNG (the obsolete version of MT). 53 bits isn't the relevant number here. The selection ratios can get close to 2. Computer scientists don't do it the way R does, for a reason.[9]

- Murdoch: No, two calls to `unif_rand()` are used. There are two 32 bit values, but some of the bits are thrown away.[10]

- Stark: No, the 2nd call only happens when $m > 2^{31}$.[11]

- Murdoch: Yes, you're right. Sorry! So the ratio really does come close to 2. However, the difference in probabilities between outcomes is still at most 2^{-32} when m is less than that cutoff. That's not feasible to detect; the only detectable difference would happen if some event was constructed to hold an abundance of outcomes with especially low (or especially high) probability. ... I'm still not convinced that there has ever been a simulation run with detectable bias compared to Monte Carlo error unless it (like this one) was designed specifically to show the problem.[12]

- Stark: ... Regarding backwards compatibility: as a user, I'd rather the default `sample()` do the best possible thing, and take an extra step to use something like `sample(..., legacy=TRUE)` if I want to reproduce old results.[13]

  One more thing, apropos this: *I'm still not convinced that there has ever been a simulation run with detectable bias compared to Monte Carlo error unless it (like this one) was designed specifically to show the problem.* I often use random permutations to simulate $p-$values to calibrate permutation tests. If I'm trying to simulate the probability of a low-probability event, this could matter a lot.[14]

- Murdoch: ... Pseudo-random number generators always have test functions whose sample averages are quite different from the expectation

[9] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076830.html`
[10] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076826.html`
[11] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076832.html`
[12] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076827.html`
[13] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076833.html`
[14] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076834.html`

under the true distribution. Remember Von Neumann's "state of sin" quote. The bug in `sample()` just means it is easier to find such a function than it would otherwise be. The practical question is whether such a function is likely to arise in practice or not. I am pretty confident that this bug rarely matters.[15]

- Hugh-Jones: It doesn't seem too hard to come up with plausible ways in which this could give bad results. Suppose I sample rows from a large dataset, maybe for bootstrapping. Suppose the rows are non-randomly ordered, e.g. odd rows are males, even rows are females. Oops! Very non-representative sample, bootstrap $p-$values are garbage.[16]

- Murdoch: That would only happen if your dataset was exactly 1717986918 elements in size. (And in fact, it will be less extreme than I posted: I had x set to 1717986918.4, as described in another thread. If you use an integer value you need a different pattern; add or subtract an element or two and the pattern needed to see a problem changes drastically.)[17]

- Stark: The same issue occurs in `walker_ProbSampleReplace()` in `random.c`, lines $386 - 387$.[18]

- Murdoch: ... By the way, there are actually quite a few examples of very large m besides $m = (2/5)2^{32}$ where performance of `sample()` is noticeably bad. You'll see problems in `y %% 2` for any integer $a > 1$ with $m = 2/(1 + 2a)2^{32}$, problems in `y %% 3` for $m = 3/(1 + 3a)2^{32}$ or $m = 3/(2 + 3a)2^{32}$, etc. So perhaps I'm starting to be convinced that the default `sample()` should be fixed.[19]

Quindi, alla fine, anche Duncan Murdoch si è convinto della necessità di correggere il *bug*, modificando la funzione `sample` con la versione 3.6.0 di R (rilasciata il 26 aprile 2019).[20] In quella versione, tra i "significant user-visible changes" troviamo scritto "The default method for generating from a discrete uniform distribution (used in `sample()`, for instance) has been changed. This addresses the fact, pointed out by Ottoboni and Stark,

[15] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076835.html`
[16] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076836.html`
[17] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076837.html`
[18] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076870.html`
[19] `https://stat.ethz.ch/pipermail/r-devel/2018-September/076863.html`
[20] `https://stat.ethz.ch/pipermail/r-announce/2019/000641.html`

that the previous method made `sample()` noticeably non-uniform on large populations. See PR#17494 for a discussion. The previous method can be requested using `RNGkind()` or `RNGversion()` if necessary for reproduction of old results. Thanks to Duncan Murdoch for contributing the patch and Gabe Becker for further assistance. The output of `RNGkind()` has been changed to also return the 'kind' used by `sample()`".

Quindi, per riassumere, a partire dalla versione 3.6.0 di R la funzione `RNGkind()` restituisce come *default*

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

mentre nelle versioni precedenti (almeno dalla 3.0.0 in poi) restituiva

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

Nella nuova versione di `sample` il *bug* è stato corretto:

```
> m <- (2/5)*2^32
> x <- sample(m, 1000000, replace = TRUE)
> table(x %% 2)

0      1
499470 500530
```

Ovviamente i risultati che si ottengono chiamando la nuova versione di `sample` non coincidono con quelli prodotti dalla versione precedente (usando lo stesso innesco). Ad esempio, la nuova versione dà

```
> set.seed(123456)
> sample(c(0:9),5,replace=TRUE)
[1] 9 9 0 6 5
```

mentre la vecchia (usata sulla versione 3.3.2) dà

```
> set.seed(123456)
> sample(c(0:9),5,replace=TRUE)
[1] 7 7 3 3 3
```

È comunque possibile, chiamando la nuova versione di `sample`, riprodurre i vecchi risultati:

```
> RNGkind(sample.kind="Rounding")
Warning message:
In RNGkind(sample.kind = "Rounding") : non-uniform 'Rounding' sampler used
> set.seed(123456)
> sample(c(0:9),5,replace=TRUE)
[1] 7 7 3 3 3
```

Tutte le simulazioni presentate nel libro sono state eseguite impiegando la vecchia versione di `sample`. Tuttavia la "dimensione" della popolazione è stata sempre tale da "garantire" una *performance* adeguata della funzione. Al lettore dubbioso non resta che verificare i nostri risultati ricorrendo alla nuova versione.