R : Simulation and Profiling

May 28, 2016

1 str

- **str** compactly displays the internal structure of an R object, a diagnostic function and an alternative to *summary*.
- Usage:
  $str(X, ...)$
  where X is an R object.
- Ex:

```
> mat <- matrix(rnorm(90), 3, 30)
> str(mat)
 num [1:3, 1:30] 0.271 0.187 1.475 -1.964 -0.694 ...
```

Or, in contrast to *summary*:

```
> set <- rnorm(100,0,4)
> str(set)
 num [1:100] -1.28 -4.81 -4.45 -2.85 2.8 ...
> summary(set)
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-10.67000  -2.86900   0.11690  -0.02998   2.71300  10.68000
```

This even works with data sets!

```
> str(airquality)
'data.frame':   153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

Or like this:

```
> s <- split(airquality, airquality$Month)
> str(s)
List of 5
 $ 5:'data.frame':      31 obs. of  6 variables:
  ..$ Ozone  : int [1:31] 41 36 12 18 NA 28 23 19 8 NA ...
  ..$ Solar.R: int [1:31] 190 118 149 313 NA NA 299 99 19 194 ...
  ..$ Wind   : num [1:31] 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
  ..$ Temp   : int [1:31] 67 72 74 62 56 66 65 59 61 69 ...
  ..$ Month  : int [1:31] 5 5 5 5 5 5 5 5 5 5 ...
  ..$ Day    : int [1:31] 1 2 3 4 5 6 7 8 9 10 ...
```

2 Generating Random Numbers

Gaussian Normal Distribution:

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

2.1 rnorm

* *rnorm*: generate *random normal variates* with a given mean and standard deviation. These are assumed to be statistically independent and are sometimes called *normally distributed random variables*.

  – Usage:

  $rnorm(n, mean = 0, sd = 1)$
  where n is the number of items output with default values $\mu = 0$ and $\sigma = 1$.

2.2 dnorm

* *dnorm*: evaluate the normal probability density (with a given mean and standard deviation) at a point - or at a vector.

$$dnorm(x, \mu, \sigma) = \int_{-\infty}^{x} P(t)\ dt$$

dnorm calculates the probability of getting a value close to x.

  – Usage:

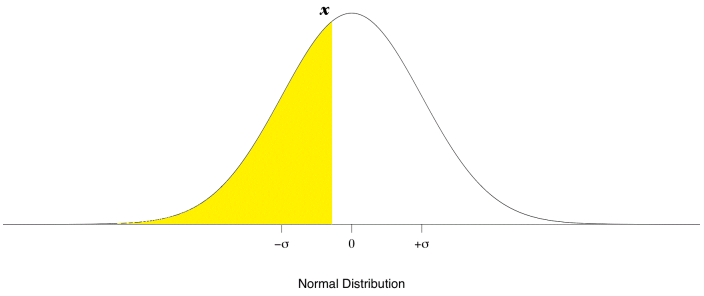
  $dnorm(n, mean = 0, sd = 1)$
  where n is the number of items output with default values $\mu = 0$ and $\sigma = 1$.

2.3 pnorm

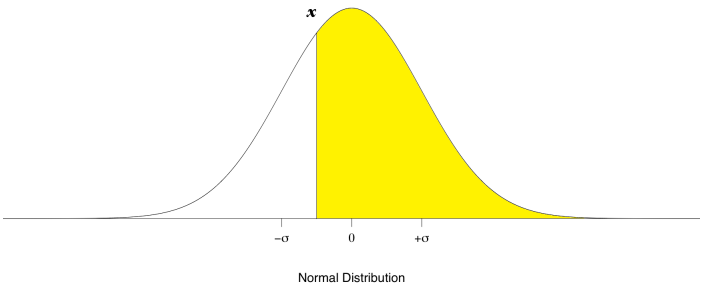
* *pnorm*: computes the proportion of observations in the normal distribution that are less than or equal to x. That is $P(X \leq x)$ where X is $N(\mu, \sigma)$.

  – Usage:

  $pnorm(n, mean, sd, lower.tail = TRUE)$



If $lower.tail = FALSE$, *pnorm* calculates the proportion of observations in the normal distribution that are greater than or equal to x. That is $P(X \geq x)$ where X is $N(\mu, \sigma)$.



For example:

```
> pnorm(.1,0,1, lower.tail=FALSE)
[1] 0.4601722
> pnorm(.1,0,1, lower.tail=TRUE)
[1] 0.5398278
```

2.4 qnorm

* *qnorm*: returns the quantile for which there is a probability of p of getting a value less than or equal to it.

  For example:

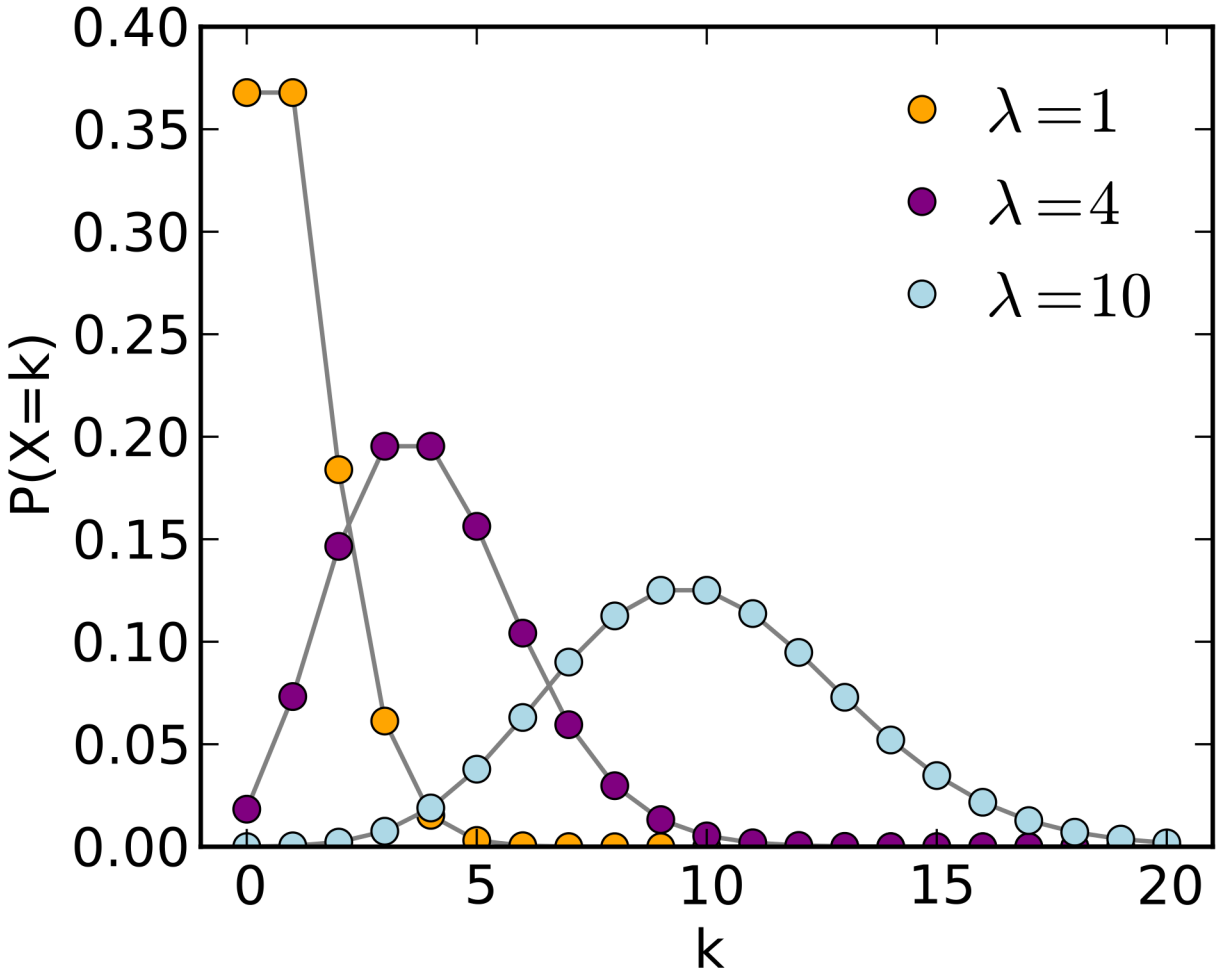
```
> qnorm(.5, mean = 0, sd = 1)
[1] 0
> qnorm(1, mean = 0, sd = 1)
[1] Inf
> qnorm(.25, mean = 0, sd = 1)
[1] -0.6744898
> qnorm(.126, mean = 0, sd = 1)
[1] -1.145505
```

  Note: *qnorm* converts proportions to quantiles while *pnorm* converts quantiles to proportions. *pnorm* and *qnorm* are inverse functions of one another.

2.5 rpois

* *rpois*: Density, distribution function, quantile function and random generation for the Poisson distribution with parameter lambda. The Poisson Distribution has density

$$P(x) = \frac{e^{-\lambda}\lambda^{x}}{x!}$$



dpois(x, lambda, log = FALSE)

ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)

qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)

rpois(n, lambda)

where:

- x is a vector of (non-negative integer) quantiles.
- q is a vector of quantiles.
- p is a vector of probabilities.
- n is the number of random values to return.
- *lambda* is a vector of (non-negative) means.
- *log*, *log.p* logical; if TRUE, probabilities p are given as $log(p)$.
- *lower.tail* logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

3 Generating Random Numbers

There are several useful probability distributions built-in to R.

rnorm: generate random normal variates with given μ and σ.

dnorm: evaluate the normal probability density (for a given μ and σ) at a point.

pnorm: evaluate the cumulative distribution function for a normal distribution.

rpois: generate random Poisson variates with a given rate.

Contrast these with *seq*:

$seq(a, b, \Delta x)$, which generates an array of equally spaced points from a to b in steps of Δx.

These (and other) R functions often use these prefixes:

i **d** for *densities*

ii **r** for *random number generation*

iii **p** for *cumulative distributions*

iv **q** for *quantile functions*

Note that **pnorm** and **qnorm** are inverses. That is, if Ω is the cumulative distribution function for a standard normal distribution, then

$$pnorm(q) = \Omega(q)$$
and
$$qnorm(p) = \Omega^{-1}(p)$$

3.1 set.seed

It is good practice to use *set.seed* in order to have reproducible results...

Usage:

$set.seed(seed, kind = NULL, normal.kind = NULL)$

Where

seed is a single value interpreted as an integer.
kind may be a character or NULL.
normal.kind sets the method used for normal generation.

Setting the random number generator seed allows for reproducibility. Examples:

```
> set.seed(314)
> rnorm(5)
[1] -1.2882358  0.7274610 -0.8329249 -0.7036765  0.1272481
> rnorm(5)
[1] -0.33494136 -0.69869011  0.01267248  0.58898930 -1.76808543
> set.seed(314)
> rnorm(5)
[1] -1.2882358  0.7274610 -0.8329249 -0.7036765  0.1272481
```

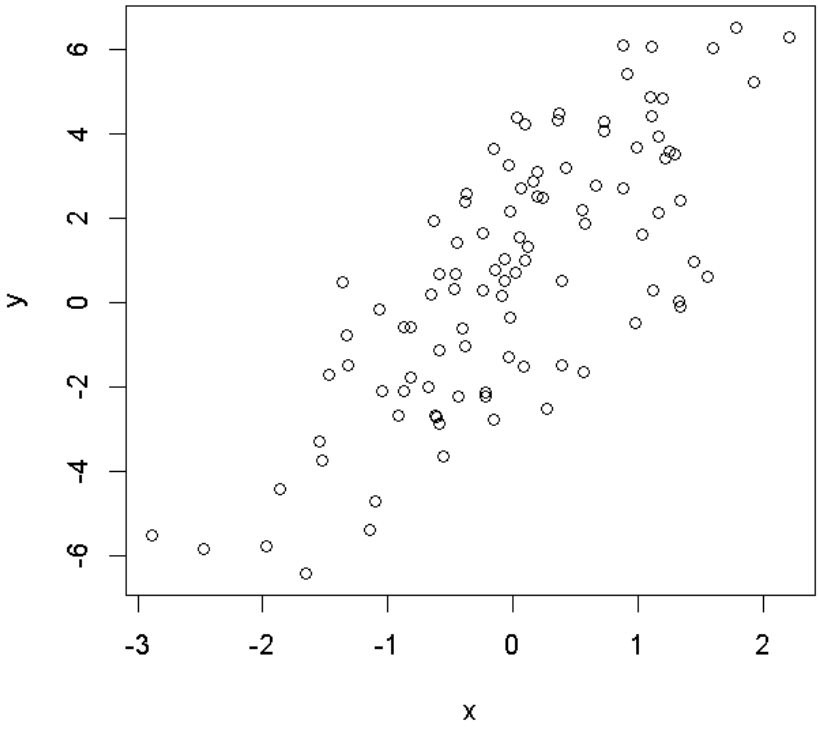
3.2 Examples

* Suppose we want to simulate

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

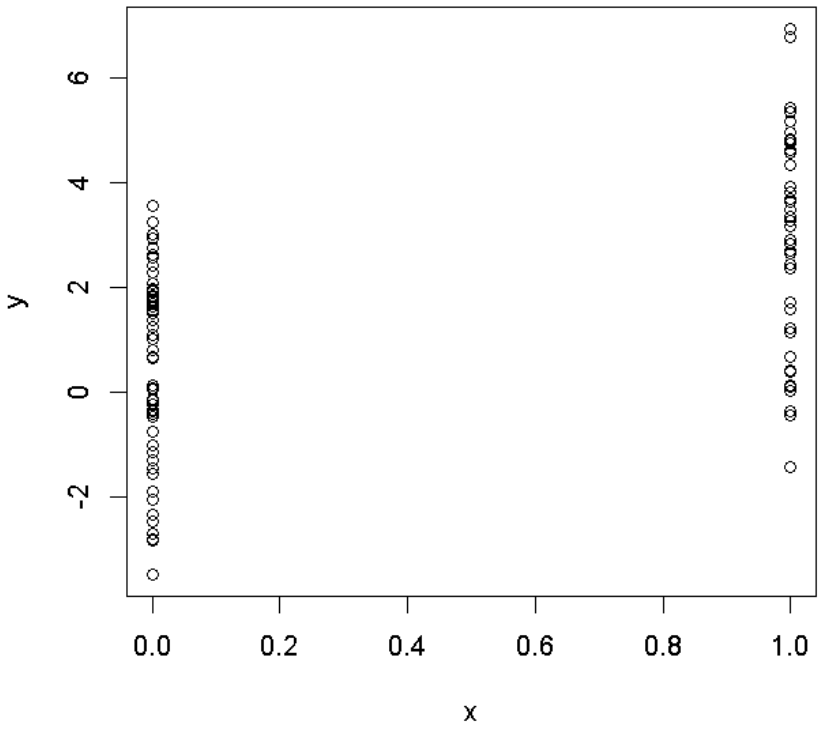
where $\varepsilon \sim N(0, 2)$, $x \sim N(0, 1)$, $\beta_0 = 0.5$, and $\beta_1 = 2$.

```
> set.seed(20)
> x <- rnorm(100)
> e <- rnorm(100,0,2)
> y <- 0.5 + 2*x + e
> summary(y)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-6.4080 -1.5400  0.6789  0.6893  2.9300  6.5050
> plot(x,y)
```



* Suppose we want a binary x.

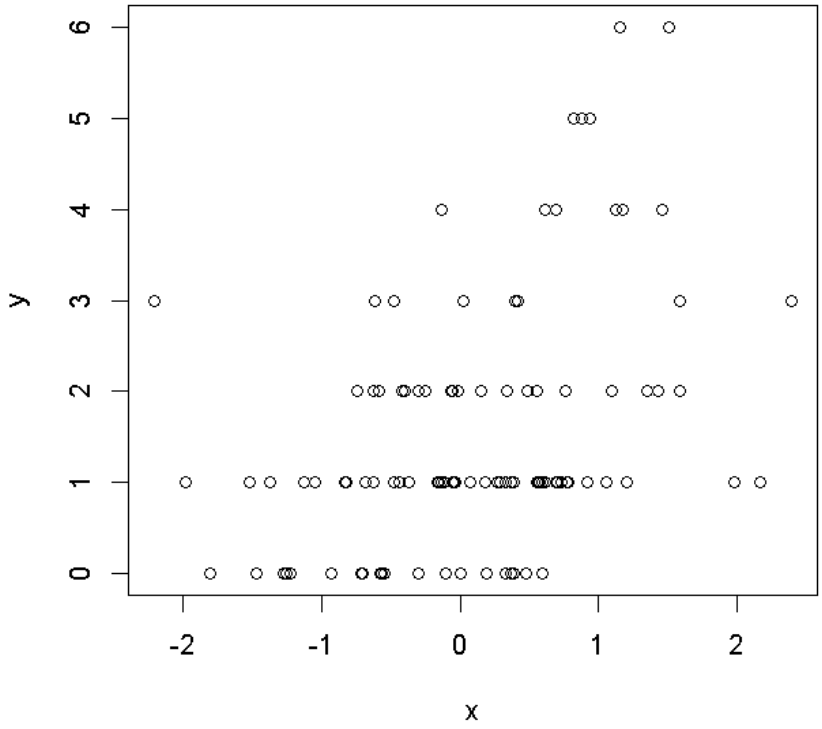
```
> set.seed(10)
> x <- rbinom(100,1,.5)
> e <- rnorm(100,0,2)
> y <- 0.5 + 2*x + e
> summary(y)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-3.4940 -0.1409  1.5770  1.4320  2.8400  6.9410
> plot(x,y)
```



* Suppose we want to simulate from a Poisson model.

$$Y \sim Poisson(\mu)$$
$$log(\mu) = \beta_0 + \beta_1$$
$$\beta_0 = 0.5 \text{ and } \beta_1 = 0.3$$

```
> set.seed(1)
> x <- rnorm(100)
> 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)
```



3.3 Random Sampling

* The *sample* function draws randomly from a specified set of scalar objects. This allows you to sample from arbitrary distributions.

```
> set.seed(1)
> sample(1:10,4)
[1] 3 4 5 7
> sample(1:10,4)
[1] 3 9 8 5
> sample(1:10,4)
[1] 7 1 2 8
> sample(letters,5)
[1] "r" "j" "s" "l" "p"
> sample(letters,5)
[1] "z" "j" "s" "v" "e"
> sample(letters,5)
[1] "q" "d" "g" "i" "a"
> sample(1:10) ## permutation
 [1]  4  8  3 10  7  9  1  5  2  6
> sample(1:10) ## permutation
 [1]  2  7  4  6  8  9  3 10  5  1
> sample(1:10, replace=TRUE)
 [1] 5 8 7 5 9 5 3 1 1 4
> sample(1:10, replace=TRUE)
 [1]  6  7  5 10  3  5  4  7  3  5
> sample(1:10, replace=TRUE)
 [1] 8 1 9 4 9 4 4 5 9 9
> sample(1:10, replace=TRUE)
 [1]  4  8 10  5  8  4  4  8  3  8
> sample(1:10, replace=TRUE)
 [1] 2 3 2 3 1 7 9 8 8 5
```