Cross Validated

Generating random samples from a custom distribution

Asked 8 years, 5 months ago Active 5 years, 9 months ago Viewed 32k times

16

12

I am trying to generate random samples from a custom pdf using R. My pdf is:

$$f_X(x) = \frac{3}{2}(1-x^2), 0 \leq x \leq 1$$

I generated uniform samples and then tried to transform it to my custom distribution. I did this by finding the cdf of my distribution ($F_X(x)$) and setting it to the uniform sample (u) and solving for x.

$$F_X(x) = \Pr[X \leq x] = \int_0^x \frac{3}{2}(1-y^2)dy = \frac{3}{2}(x - \frac{x^3}{3})$$

To generate a random sample with the above distribution, get a uniform sample $u \in [0,1]$ and solve for x in

$$\frac{3}{2}(x - \frac{x^3}{3}) = u$$

I implemented it in `R` and I don't get the expected distribution. Can anyone point out the flaw in my understanding?

```
nsamples <- 1000;
x <- runif(nsamples);

f <- function(x, u) {
  return(3/2*(x-x^3/3) - u);
}

z <- c();
for (i in 1:nsamples) {
  # find the root within (0,1)
  r <- uniroot(f, c(0,1), tol = 0.0001, u = x[i])$root;
  z <- c(z, r);
}
```

r sampling uniform

edited Jul 9 '11 at 21:56

asked Jul 9 '11 at 20:24

Anand

1,002 2 10 21

1 Must be a coding mistake. I don't use R, so I can't say what the mistake is exactly -- but I just coded up your solution (taking care to take the middle root of the cubic polynomial, which always lies between 0 and 1), and I get good agreement between the samples and the expected distribution. Could it be a problem with your root finder? What's wrong with the samples you're getting? – jpillow Jul 9 '11 at 21:58

I tried your code (which is not very efficient, by the way) and do get the expected distribution. – Aniko Jul 9 '11 at 22:20

@jpillow and @Aniko My mistake. When I used `nsamples <- 1e6` it was a good match. – Anand Jul 9 '11 at 22:32

2 @Anand One way is to observe that $x = 2\sin(\arcsin(u)/3)$, allowing direct calculation of x in terms of u. – whuber ♦ Jul 11 '11 at 14:08

1 @Anand en.wikipedia.org/wiki/… – whuber ♦ Jul 14 '11 at 16:08

|

1 Answer

11

It looks like you figured out that your code works, but @Aniko pointed out that you could improve its efficiency. Your biggest speed gain would probably come from pre-allocating memory for `z` so that you're not growing it inside a loop. Something like `z <- rep(NA, nsamples)` should do the trick. You may get a small speed gain from using `vapply()` (which specifies the returned variable type) instead of an explicit loop (there's a great SO question on the apply family).

```
> nsamples <- 1E5
> x <- runif(nsamples)
> f <- function(x, u) 1.5 * (x - (x^3) / 3) - u
> z <- c()
>
> # original version
> system.time({
+ for (i in 1:nsamples) {
+   # find the root within (0,1)
+   r <- uniroot(f, c(0,1), tol = 0.0001, u = x[i])$root
+   z <- c(z, r)
+ }
+ })
   user  system elapsed
  49.88    0.00   50.54
>
> # original version with pre-allocation
> z.pre <- rep(NA, nsamples)
> system.time({
+ for (i in 1:nsamples) {
+   # find the root within (0,1)
+   z.pre[i] <- uniroot(f, c(0,1), tol = 0.0001, u = x[i])$root
+   }
+ })
   user  system elapsed
   7.55    0.01    7.78
>
```

```
>
>
> # my version with sapply
> my.uniroot <- function(x) uniroot(f, c(0, 1), tol = 0.0001, u = x)$root
> system.time({
+   r <- vapply(x, my.uniroot, numeric(1))
+ })
   user  system elapsed
   6.61    0.02    6.74
>
> # same results
> head(z)
[1] 0.7803198 0.2860108 0.5153724 0.2479611 0.3451658 0.4682738
> head(z.pre)
[1] 0.7803198 0.2860108 0.5153724 0.2479611 0.3451658 0.4682738
> head(r)
[1] 0.7803198 0.2860108 0.5153724 0.2479611 0.3451658 0.4682738
```

And you don't need the `;` at the end of each line (are you a MATLAB convert?).

edited May 23 '17 at 12:39
Community ♦
1

answered Jul 9 '11 at 23:59
Richard Herron
1,031 2 12 19

Thanks for your detailed answer and for pointing out `vapply`. I have been coding in `C/C++` for a very long time and that is the reason for `;` affliction! – Anand Jul 10 '11 at 3:16

1 +1 It looks like replacing `uniroot` by a direct solution speeds things up by three more orders of magnitude: you should have no trouble obtaining around 10^7 variates per second. – whuber ♦ Jul 11 '11 at 17:10