Subject: Re: "bootstrap" statistics
Posted by Wayne Landsman on Mon, 20 May 2002 20:58:16 GMT
View Forum Message <> Reply to Message

> I have found:
>
> http://www.astro.washington.edu/deutsch-bin/getpro/library14 .html?PERMUTE
>
> which is a somewhat better way, but still slow. Is there a no-loops version?

Interesting question. I had modified the above PERMUTE program to use a vector call to RANDOMN(/DOUBLE), (the /DOUBLE keyword has been available since V5.4). My feeling was that with ~5e15 distinct double precision numbers between 0 and 1, that probablity of RANDOMN returning two identical numbers was vanishingly small, in a typical call of say less than 10,000 numbers

But I've never been comfortable enough with the modification to actually use it. I suppose I should add a check for any equal numbers in the RANDOMN(/DOUBLE) call, and then randomize those numbers.

--Wayne

```
FUNCTION PERMUTE,M, RSEED, OUTSEED = seed
;+
; NAME:
;       PERMUTE
; PURPOSE:
;       Randomly permute the elements of a vector
; USAGE:
;       NewIndex = PERMUTE( M, [ InSeed, OUTSEED = ] )
; INPUT:
;       M = length of vector
; OPTIONAL INPUT-OUTPUT:
;       SEED = random number seed to be passed to RANDOMU
;              Upon return, it will updated as a vector containing a new seed
; OUTPUT:
;       PERMUTE returns M randomly shuffled integers between 0 and M-1
; EXAMPLE:
;       To shuffle the elements of a vector V,
;
;       V = V [ PERMUTE(N_ELEMENTS(V) ) ]
;
; REVISION HISTORY:
;       Written,  H.T. Freudenreich, HSTX, 2/95
;       Use vectorized call   W. Landsman  SSAI  December 2001
;-
;
; Select M numbers at random, repeating none.
```

```
    if N_elements(rseed) GT 0 then seed = rseed
    return, sort( randomn(seed, m,/Double) )
```

Subject: Re: "bootstrap" statistics
Posted by Dick Jackson on Mon, 20 May 2002 21:50:40 GMT
View Forum Message <> Reply to Message

<wmc@bas.ac.uk> wrote in message news:3ce95177@news.nwl.ac.uk...
> Hello group. I want to do what I think of as "bootstrap" statistics, viz
> given a timeseries I take a random subsample (with, say, half the number
> of elements), compute some statistic (say, then mean); then take another
> random subsample; then again lots of times (say 1000 or 10000) and end
> up with a distribution of the statistic concerned.
>
> So: to do this I need a means to generate n/2 random (non-repeating)
> indices from 0...n-1.

I am certainly not an expert on bootstrap stats, but I did consult with one extensively when I implemented this for a former client. My instructions were to always choose from our sample dataset *with* replacement, generating random (possibly-repeating) indices. This, of course, is faster and would be trivial for you to write!

So if this applies to your situation, it "unasks" your otherwise very interesting question, which still deserves a good answer! :-)

Cheers,
--
-Dick

Dick Jackson / dick@d-jackson.com
D-Jackson Software Consulting / http://www.d-jackson.com
Calgary, Alberta, Canada / +1-403-242-7398 / Fax: 241-7392

Subject: Re: "bootstrap" statistics
Posted by Ivan Valtchanov on Tue, 21 May 2002 07:54:26 GMT
View Forum Message <> Reply to Message

On 20 May 2002 20:41:44 +0100
wmc@bas.ac.uk wrote:

> Hello group. I want to do what I think of as "bootstrap" statistics, viz
> given a timeseries I take a random subsample (with, say, half the number
> of elements), compute some statistic (say, then mean); then take another

> random subsample; then again lots of times (say 1000 or 10000) and end
> up with a distribution of the statistic concerned.
>
> So: to do this I need a means to generate n/2 random (non-repeating)
> indices from 0...n-1. At the moment I do this by:
> once I have m indices I generate one more at
> random; see if its in the list of m; if not, good; if it is, generate another
> one. This is hideously inefficient and slow: there *must* be a better way.
>
Hello,

It is some sort of jacknife technique. I can propose you one solution:

```
ndata = float(n_elements(myarray)) ; myarray is the input array

for i=0, ngen-1 do begin
 x = randomu(s,ndata)
 flag = nint(x) ; nearest integer, so should have half times zeros and half times ones
 ixx = where(flag EQ 0)

 tmp = myarray ;
 remove,ixx,tmp ; using Wayne Landsman astrlib REMOVE.PRO

 mx[i] = mean(tmp)
endfor
```

Hope this helps.

Ivan
--
== ============
Ivan Valtchanov   e-mail:ivaltchanov@cea.fr
DSM/DAPNIA/Service d'Astrophysique
CEA/Saclay   tel: +33(0)1.69.08.34.92
Orme des Merisiers, bat.709   fax: +33(0)1.69.08.65.77
91191 Gif-sur-Yvette, France
== ============

Subject: Re: "bootstrap" statistics
Posted by ejensen1 on Tue, 21 May 2002 21:39:24 GMT
View Forum Message <> Reply to Message

Wayne Landsman <landsman@mpb.gsfc.nasa.gov> wrote
>
> Interesting question.   I had modified the above PERMUTE program to use a
> vector call to RANDOMN(/DOUBLE), (the /DOUBLE keyword has been available since
> V5.4).   My feeling was that with ~5e15 distinct double precision numbers

```
>  between 0 and 1, that probablity of RANDOMN returning two identical numbers was
>  vanishingly small, in a typical call of say less than 10,000 numbers
>
>  But I've never been comfortable enough with the modification to actually use
>  it.     I suppose I should add a check for any equal numbers in the
>  RANDOMN(/DOUBLE) call, and then randomize those numbers.
>
>  ;
>  ; Select M numbers at random, repeating none.
>
>      if N_elements(rseed) GT 0 then seed = rseed
>      return, sort( randomn(seed, m,/Double) )
```

This is the approach that I hit upon independently when I was doing resampling statistics. Note that the possible repetition, unlikely as it is, doesn't have much of an adverse effect on your resampling since SORT returns distinct indices for tied values, e.g.

```
IDL> print, sort([3,1,3,3,3])
           1           0           2           3           4
```

So I do something like this:

```
; Return indices for a random subsample of size m out of a full
; population of size n without replacement:

 (sort(randomu(seed,n)))[0:m-1]
```

If there are any ties in the random numbers returned, the SORT call will still return distinct indices, but it will be preserving a small amount of information about the ordering of the parent sample. If one really wanted to be paranoid about it, one could add an extra randomization step in there, so you don't draw the first m out of your final (mostly randomized) indices for the parent sample, but you use that set of m indices that are between 0 and n-1 to tell you which of *another* set of randomized 0 to n-1 indices to pick:

```
(sort(randomu(seed,n)))[(sort(randomu(seed,n)))[0:m-1]]
```

Now as long as the repeats in one randomu call don't match up with the repeats in the other, I think you should be fine. In theory one would need an infinite regression of such randomizations, I suppose, to be completely free of this problem, but I think this should work pretty well for most applications. This should especially be true if you use the /Double keyword as Wayne Landsman suggested to minimize the chance of repetions, though I don't know how that effects speed relative to not using it.

Also, RANDOMN may be somewhat less likely to return repeated
values than RANDOMU since the values aren't confined to 0-1,
but on the other hand they are more strongly peaked around
0, so I'm not sure which one wins out. You could test it,
which would also give you an idea of how often repeats
happen (though I'm not sure off the top of my head the most
efficient way to look for repeats in a big dataset).

Be sure to keep track of your SEED value if you have this in
a subroutine, or you'll really have problems with non-randomness
if you call RANDOMU repeatedly with an uninitialized SEED.

Hope this helps,

Eric