Subject: Re: Random selection
Posted by JD Smith on Mon, 13 Nov 2006 18:46:08 GMT
View Forum Message <> Reply to Message

On Sun, 12 Nov 2006 05:07:19 -0800, greg michael wrote:

> I'm not sure doubles are going to help - you'll get the same problem at
> 3 digits. And if you use the sort method, it doesn't really matter if
> you get two the same - they'll still map to unique indices.

If you only want to pick 10 random elements from a list 100,000 long,
it's very inefficient to generate a set of 100,000 random numbers, sort
them all, and then take the first 10 indices. There are all sorts of
iterative higher-order algorithms for selection without replacement, but
they don't match to IDL well. One simple trick would be to start by
generating M random numbers, check for duplicates, and generate M-n more,
accumulating until you have enough.

```
M=10
len=100000L
inds=lonarr(n,/NOZERO)
n=M
while n gt 0 do begin
   inds[M-n]=long(randomu(sd,n)*len)
   inds=inds[sort(inds)]
   u=uniq(inds)
   n=M-n_elements(u)
   inds[0]=inds[u]
end
```

For this case, the speedup is immense, on average about 3500x faster.
What about a case with more duplicates likely? How about len=100000,
M=25000?

```
Sort All randoms:           0.13349121
Brute force replacement:    0.091892505
```

Still about 1.5x faster.

Obviously, if you wanted len-1 random indices, this won't scale, but in
that case, you could just invert the problem, choose the random indices
to be *discarded*, and use HISTOGRAM to generate the "real" list.
Here's a general function which does this for you.

```
function random_indices, len, n_in
  swap=n_in gt len/2
  if swap then n=len-n_in else n=n_in
  inds=lonarr(n,/NOZERO)
```

```
  M=n
  while n gt 0 do begin
    inds[M-n]=long(randomu(sd,n)*len)
    inds=inds[sort(inds)]
    u=uniq(inds)
    n=M-n_elements(u)
    inds[0]=inds[u]
  endwhile

  if swap then inds=where(histogram(inds,MIN=0,MAX=len-1) eq 0)
  return,inds
end
```

It is outperformed by the simple sort method:

```
   r=randomu(sd,len)
   inds=(sort(r))[0:M-1]
```

only when M is close to len/2. For example, I found that selecting from length 100000 fewer than 30000 or more than 70000 elements favored RANDOM_INDICES. At worst case (M=len/2), it's roughly 3x slower. The RANDOM_INDICES method also returns the indices in sorted order (which you may or may not care about). You could obviously also make a hybrid approach which switches from one method to the other for

```
 abs(M-len/2) lt len/5
```

or so, but the tuning would be machine-specific.

JD
