
Subject: mean() function

Posted by [biocpu](#) on Tue, 10 Jan 2006 22:34:09 GMT

[View Forum Message](#) <> [Reply to Message](#)

The following looks very odd. Have any clues?

```
IDL> y = fltarr(1008879)+35
IDL> id = where(y ne 35, cc)
IDL> print, cc
      0
```

; so y is strictly 35.0 BUT

```
IDL> print, mean(y)
      35.5249
```

```
IDL> print, mean(y(0:400000))
      35.0000
```

Thanks,

Subject: Re: mean() function

Posted by [Kenneth P. Bowman](#) on Thu, 12 Jan 2006 17:07:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

In article <ywkuymm117bv.fsf@snowblower.colorado.edu>, savoie@nsidc.org wrote:

> "Maarten" <maarten.sneep@knmi.nl> writes:

>

>> And the reason you need that page, is in part because IDL uses the
>> moment routine described in Numerical Recipes (take total first, divide
>> later), instead of a proper running average, like the GNU scientific
>> library does.

>>

>> However, since looping is slow in IDL, you don't want to implement that
>> in IDL, so the next best thing is to have that page.

The situation to avoid is adding values with different magnitudes. That results in a loss of precision.

For example, summing a large number of values with similar magnitudes will eventually result in adding small values to large values.

There are several tricks one can use to avoid this, particularly in the case where all the values have similar magnitudes. The simplest is to assume that the mean is close to the first value. Subtract the first value from each value before accumulating, compute the mean, then add the first value back into the total. Alternatively, compute a first

approximation to the mean the naive way, then recompute the mean by subtracting the approximate mean from each value before summing. This requires two passes through the data, so the tradeoff is computational time and precision.

Ken Bowman

Subject: Re: mean() function

Posted by [Maarten\[1\]](#) on Fri, 13 Jan 2006 08:57:40 GMT

[View Forum Message](#) <> [Reply to Message](#)

savoie@nsidc.org wrote:

> Would you mind explaining this a bit for me? What's a proper running
> average? And why is it better in general?

It assumes that values are reasonably close to the average,
mathematically it is equivalent to taking the total, and then dividing,
but it avoids some precision problems.

In IDL code it requires an explicit loop over the data, which is slow.
An alternative is to do everything in double precision, but that is
just postponing the inevitable.

Code to calculate the mean in an array X:

```
mean = 0.  
for ii = 0, n_elements(X)-1 do $  
    mean += (X[ii] - mean) / (ii + 1)
```

Maarten

Subject: Re: mean() function

Posted by [Foldy Lajos](#) on Sun, 15 Jan 2006 13:53:17 GMT

[View Forum Message](#) <> [Reply to Message](#)

Hi,

sometimes there is a trade-off between speed and correctness.
Here is another example:

```
IDL> print, !version  
{ x86 linux unix linux 6.2 Jun 20 2005    32    64}
```

```
IDL> print, smooth([1, 1.0e20, 1, 1, 1, 1], 3)
```

1.00000 3.33333e+19 3.33333e+19 0.00000 0.00000 1.00000

Instead of calculating the average for each 3-element window, IDL uses the first average only, then slides the window by adding/subtracting elements from the window edge. For width w it uses only 2 add/sub per window instead of w additions, which is a huge speed-up for large widths, but can give incorrect result for rare cases.

regards,
lajos

On Sun, 15 Jan 2006, Reimar Bauer wrote:

```
> mean is not useable if it results in this
>
> IDL> print,mean( make_array(500000,val=35,/float) )
>    35.0413
> IDL> print,mean( make_array(400000,val=35,/float) )
>    35.0000
>
> I prefer a slower routine if this is right.
>
> no one would accept 1.0 + 1.0 result = 1.5
>
```

Subject: Re: mean() function

Posted by [Maarten\[1\]](#) on Mon, 16 Jan 2006 08:01:05 GMT

[View Forum Message](#) <> [Reply to Message](#)

I think that this comes close. I ignores infinite numbers on request.
Is it fast: no. But implementing the thing in C should be near trivial
if you have dealt with that before (I haven't, at least not in IDL).

```
function alt_mean, D, nan=nan
  compile_opt defint32, strictarr, logical_predicate, strictarrsubs
```

```
M = 0.0
```

```
if keyword_set(nan) then begin
  idx = where(finite(D), cnt)
  if cnt gt 0 then begin
    for ii=0,n_elements(idx) do $
      M += (D[idx[ii]] - M)/(ii+1)
  endif else begin
    M = !values.d_nan
  endelse
```

```
endif else begin
  for ii=0,n_elements(D) do $
    M += (D[ii] - M)/(ii+1)
  endelse
```

```
return, M
end
```

Subject: Re: mean() function

Posted by [Foldy Lajos](#) on Mon, 16 Jan 2006 10:27:34 GMT

[View Forum Message](#) <> [Reply to Message](#)

Hi,

one small correction: instead of `n_elements(...)` you should use `n_elements(...)-1`.

This approach has an other problem: the elements at the beginning of the array are divided `~n_elements()` times, which can introduce large rounding errors.

Example:

```
IDL> print, !version
print, !version
{ x86 linux unix linux 6.2 Jun 20 2005    32    64}
```

```
IDL> a=fltarr(100000000l)
IDL> a[0]=1.0e7
IDL> print, alt_mean(a)
    1.02190
```

vs.

```
IDL> print, mean(a)
% Compiled module: MEAN.
% Compiled module: MOMENT.
    1.00000
```

there is no golden way :-)))

regards,
lajos

On Mon, 16 Jan 2006, Maarten wrote:

```
> I think that this comes close. I ignores infinite numbers on request.
> Is it fast: no. But implementing the thing in C should be near trivial
> if you have dealt with that before (I haven't, at least not in IDL).
>
> function alt_mean, D, nan=nan
> compile_opt defint32, strictarr, logical_predicate, strictarrsubs
>
> M = 0.0
>
> if keyword_set(nan) then begin
>   idx = where(finite(D), cnt)
>   if cnt gt 0 then begin
>     for ii=0,n_elements(idx) do $
>       M += (D[idx[ii]] - M)/(ii+1)
>   endif else begin
>     M = !values.d_nan
>   endelse
> endif else begin
>   for ii=0,n_elements(D) do $
>     M += (D[ii] - M)/(ii+1)
>   endelse
>
> return, M
> end
>
>
```
