Subject: Re: Cumulative total
Posted by Thomas A. McGlynn on Thu, 24 Sep 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Kenneth P. Bowman wrote:
>
> In article <3602B1F3.210@cdc.noaa.gov>, Andrew Loughe <afl@cdc.noaa.gov> wrote:
>
>> Nice, elegant solution, Eddie.
>
>>> here is the way i do it:
>>>
>>> IDL> A = findgen(10)
>>> IDL> N = n_elements(A)
>>> IDL> result = A # (lindgen(N,N) ge transpose(lindgen(N,N)))
...
> In my opinion this is a baroque construction that reveals a shortcoming in
> IDL. (Hey, not a major shortcoming. I'm not committing heresy here!)
>
> To compute the cumulative sum of a vector of length N, i.e.,
>
> x_cum[0] = x[0]
> FOR i = 1, N-1 DO x_cum[i] = x_cum[i-1] + x[i]
>
> should require N loads, N flops (adds), and N stores. This may not
> optimize well, since each result depends on the previous one, but I
> suspect most modern Fortran or C compilers would do pretty well.
>
> The IDL approach above requires creating an N^2 matrix filled with
> integers, performing an if test on every element of that array and its
> transpose, and then performing a matrix-vector multiply (N^2 multiplies
> and N^2 adds). What do you do when N = 100,000? It seem a silly way to
> do a simple task.
>
> I'm not trying to pick on Eddie. It works for him. I wonder which method
> is faster?
>
...
> So, note to RSI: Add CUMULATIVE function to next release of IDL and make
> it a good IDL function so that one can specify which dimension of a
> possibly multidimensional array to accumulate over, etc. It should be
> quite useful with HISTOGRAM.
>
> Ken
>

Here's an idea for a more general solution. The problem arises
because when IDL evaluates a vector expression it acts as if all

elements of the expression are evaluated simulataneously. Suppose IDL had a new equality operator which requires vector expressions to be evaluated in subscript order.

E.g. currently,

```
a(1:9) = a(0:8) + a(1:9)
```

translates to

```
b = a
a(1) = b(0) + b(1)
a(2) = b(1) + b(2)
...
```

So we can't easily calculate a cumulative distribution.

Suppose we have a new equality operator, say :=, which says that each vector expression will be evaluated in turn so

```
a(1:9) := a(0:8) + a(1:9)
```

translates to

```
a(1) = a(0) + a(1)
a(2) = a(1) + a(2)
```

which gives us a cumulative distribution and solves the original problem efficiently and elegantly. For a non-vector expression, := and = would be identical.

Another example of where this would be useful is an application where you have the bin numbers for a number of events and you want to reconstruct the histogram. E.g., suppose I have the measured x and y pixels for a number of photons measured in some camera, and I'd like to construct an image. There may be many photons that hit the same pixel. Let the arrays x and y hold the pixel indices Then

```
image(*,*) = 0
image(x,y) = image(x,y) + 1
```

would be a nice way to construct the image, but it doesn't work. Regardless of how many photons were detected in a given pixel the maximum value for image will be 1. This particular example that I ran into -- and it hurt!

Using the := operator describe above we'd get the right answer. I think there are a lot of applications where this would be helpful and it probably would obviate the need for a lot of special keyword in various functions.

There might be a penalty to pay in real vector machines for these kinds of non-vectorizable loops, but most of us aren't using IDL on Crays. My -- likely flawed -- understanding is that the problem with explicit loops is that the interpreter is called for each iteration. There would be no reason to do that for the := operator so that they could be quite fast. Indeed, they might occasionally be faster than the current loops since there would be no need for the 'b = a' (or equivalent) statement that we had in the first example.

I'll likely be seeing Dave Stern in a few weeks. Is this idea off the wall or a reasonable extension to ask for?

Tom McGlynn
tam@silk.gsfc.nasa.gov
