Subject: Re: Zero vector detection in IDL
Posted by Pavel Romashkin on Mon, 28 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Using WHERE does search through the entire array. I tried it: on my Power Mac I could only get a FINDGEN(10,500,000) in memory, and locating randomly incerted zeros in it using WHERE took 1.6 s. Unless your arrays are much larger, I'd stick with WHERE for simplicity sake. Saves about 8 lines of code, which I am not sure would speeds things up much: what if your non-zero is at the end? You will loose an hour to loop through 10,500,000 elements.
Good luck,
Pavel

Frank Morgan wrote:

> Given a big byte or integer array (actually a 1-D vector), is there a
> fast IDL way to check whether any non-zero elements exist?
>
> Something like "if (total(x) EQ 0)..." or a where() construct will work,
> but scans the whole vector when the first non-zero element is enough to
> answer the question. max(x) is even worse.

Subject: Re: Zero vector detection in IDL
Posted by J.D. Smith on Wed, 30 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Frank Morgan wrote:
>
> Given a big byte or integer array (actually a 1-D vector), is there a
> fast IDL way to check whether any non-zero elements exist?
>
> Something like "if (total(x) EQ 0)..." or a where() construct will work,
> but scans the whole vector when the first non-zero element is enough to
> answer the question. max(x) is even worse.
>
> On the other hand:
> for i=0,n-1 do begin
> if (x(i) NE 0) then begin
> (it's not all zero)
> goto, BREAK
> endif
> endif
> BREAK:
> will stop early if possible, but looping a
> conditional is maybe not the
> fastest structure in IDL.

>
> An internal command that implements the loop concept would be what I'm
> looking for I think but I don't know if it exists.
>
> Any better ideas?
>
> Thanks,
> Frank
> frank.morgan@jhuapl.edu

I was curious just how slow loops are. I had to revert to IDL 5.1 on my linux machine to get systime(1) doesn't work... hint hint. Anyway, I investigated three methods:

1. the loop above
2. (where(a ne 0.0d))[0] ne -1
3. a wrapped call_external to a C program similar to the above loop

I used a double vector of length 1 million:
a=[dblarr(500000),dindgen(500000)]

Results:

1. Average Time: 0.9944s
2. Average Time: 0.1172s
3. Average Time: 0.04447s

Two things to notice: IDL loops are *very* slow, and where isn't *too* bad for having had to search the full vector. The external program was just a little snippet like:

```
arr = (double*)argv[0];
    for(i=0;i<*(IDL_LONG*)argv[1];i++)
      if(arr[i]!=0.0) return 1;
return 0;
```

and I used a wrapper routine like:

```
function non_zero,a
  return, call_external('non_zero.so','non_zero',a,n_elements(a))
end
```

This only really wants doubles, but it shouldn't be hard to add the type also, and cast the array pointer accordingly. See the external examples for details on compiling for your system.

JD

--

J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|