Subject: Bug/feature in matrix multiply
Posted by Mark Fardal on Fri, 12 Mar 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Hi,

is the following a bug or feature? I don't understand why changing the
type of the array changes the dimensions of the result. Then again,
it's late on Friday, so my brain might just be mush.

```
IDL> junk=fltarr(3)
IDL> junk=reform(junk,3,1)
IDL> help,junk
JUNK            FLOAT     = Array[3, 1]
IDL> help,[3.,2.,1.]#junk
<Expression>    FLOAT     = Array[1]
IDL> help,[3.d0,2.d0,1.d0]#junk
<Expression>    DOUBLE    = Array[3, 3]
```

this is with
IDL Version 5.2 (sunos sparc). Research Systems, Inc.

thanks,
Mark Fardal
UMass
fardal@weka.phast.umass.edu

Subject: Re: Bug/feature in matrix multiply
Posted by steinhh on Sat, 13 Mar 1999 08:00:00 GMT
View Forum Message <> Reply to Message

> Mark Fardal (fardal@weka.phast.umass.edu) writes:
>
>> is the following a bug or feature? I don't understand why changing the
>> type of the array changes the dimensions of the result. Then again,
>> it's late on Friday, so my brain might just be mush.
>>
>> IDL> junk=fltarr(3)
>> IDL> junk=reform(junk,3,1)
>> IDL> help,junk
>> JUNK   FLOAT   = Array[3, 1]
>> IDL> help,[3.,2.,1.]#junk
>> <Expression>   FLOAT   = Array[1]
>> IDL> help,[3.d0,2.d0,1.d0]#junk
>> <Expression>   DOUBLE   = Array[3, 3]
>
> I don't know if it is a bug or a feature, but I

> agree that it is strange. But so is this command:
>
> junk = reform(junk, 3, 1)
>
> Do you mean this:
>
> junk = reform(junk, 1, 3)
>
> The latter will make a column vector, which makes more
> sense when multiplied by a row vector. What kind of result
> were you expecting? From my reading of the # operator
> I think the result with the floating array is correct.
> I don't have a clue why the double expression does what
> it does. :-(

Hint:

```
junk=reform(junk,3,1)
help,junk,double(junk)
```

8-)

Subject: Re: Bug/feature in matrix multiply
Posted by Mark Fardal on Sun, 14 Mar 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Hi,

> Hint:
>
> junk=reform(junk,3,1)
> help,junk,double(junk)
>
> 8-)

Stein is quite right. The problem occurs because converting junk to double removes the trailing dimension.

So my question becomes, why does this happen? If IDL is going to treat an 3x1 array differently than a 3-element vector, it shouldn't just cavalierly remove the trailing dimension in my opinion. The behavior is probably documented somewhere but I couldn't find it in the hyperhelp. There is this one sentence in "Combining Array Subscripts with Others": "As with other subscript operations, trailing degenerate dimensions (those with a size of 1) are eliminated."

I also notice that the behavior is somewhat inconsistent, in that

converting an expression to one of the same type does _not_ remove
the trailing dimension:

```
IDL> junk=intarr(3)
IDL> junk=reform(junk,3,1)
IDL> help,junk
JUNK            INT       = Array[3, 1]
IDL> help,junk,float(junk),fix(junk),double(junk)
JUNK            INT       = Array[3, 1]
<Expression>    FLOAT     = Array[3]
JUNK            INT       = Array[3, 1]
<Expression>    DOUBLE    = Array[3]
```

Also, conversion of an array of length 1 does not produce a scalar.
It seems like this would be the analogous behavior.

The initial problem I had may clarify why this is important. I was
trying to do a nonlinear, 1-parameter fit, and chose to use CURVEFIT.
This is clearly killing a fly with a machine gun, but hey, the machine
gun was close at hand. Also, the code for CURVEFIT does indicate some
thought about the 1-parameter case, i.e.

```
        IF nterms EQ 1 THEN pder = reform(pder, n_elements(y), 1)
```

(Does this line answer your question David?) I used single precision
for most variables, but returned the fitting function as a double
array. This caused CURVEFIT to crash. Here's a simple program that
demonstrates the same behavior:

```
pro expdecay, x, rate, yfit

yfit = exp(-rate(0) * x)
yfit = double(yfit)

return
end


pro testcurvefit

x = [0., 1., 2.]
y = exp(-x)
params=[1.1]
weights = x*0 + 1.
fit = curvefit(x, y, weights, params, function_name='expdecay', /noderivative)

print, 'Rate constant:', params(0)
```

end

When running this routine I get

```
IDL> testcurvefit
% Operands of matrix multiply have incompatible dimensions: <FLOAT
  Array[1]>, <DOUBLE    Array[3, 3]>.
% Error occurred at:  CURVEFIT         279
  /usr/local/rsi/idl/lib/curvefit.pro
```

This happens because in the statement

```
        beta = (y-yfit)*Weights # pder
```

pder is a floating 3x1 array, since the parameter "a" (= params) was a float. It is getting multiplied by ((y-yfit)*Weights) which is a double, so pder get promoted to double and loses its trailing dimension in the process. Then beta winds up as the #-product of two 3-element vectors, or a 3x3 array. It should be a 1x1 array.

I believe this demonstrates a bug in either CURVEFIT or in type conversion in general. My vote is for the latter. A workaround to using CURVEFIT is to make all parameters the same type.

Mark Fardal
UMass

Subject: Re: Bug/feature in matrix multiply
Posted by Mark Fardal on Mon, 15 Mar 1999 08:00:00 GMT
View Forum Message <> Reply to Message

> I agree (somewhat). Generally, I found that IDL is quite "smart" in
> removing trailing dimensions so that one doesn't have to worry too much
> about them.

When is this "feature" actually useful? I'm sure I use it in some form but I can't think of it.

> Anyway: you can always make sure you get what you want with
>
> a = transpose(reform(a))
>
> These statements are not very costly in terms of execution time, because
> it's only messing around with the array descriptor (at least I believe
> so).

Could you be more specific about what this statement is supposed to

do? It will give a 1xN array, not a Nx1 array which is what I wanted originally. If you do a second transpose it drops the trailing dimension again.

On the subject of speed, try it with out with 10M elements. I think it's altering more than a single descriptor.

thanks,
Mark Fardal
UMass

Subject: Re: Bug/feature in matrix multiply
Posted by Martin Schultz on Mon, 15 Mar 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Mark Fardal wrote:
> [...]

> So my question becomes, why does this happen? If IDL is going to
> treat an 3x1 array differently than a 3-element vector, it shouldn't
> just cavalierly remove the trailing dimension in my opinion. The
> behavior is probably documented somewhere but I couldn't find it in
> the hyperhelp. There is this one sentence in "Combining Array
> Subscripts with Others": "As with other subscript operations, trailing
> degenerate dimensions (those with a size of 1) are eliminated."
>
> I also notice that the behavior is somewhat inconsistent, in that
> converting an expression to one of the same type does _not_ remove
> the trailing dimension:
> [...]

I agree (somewhat). Generally, I found that IDL is quite "smart" in removing trailing dimensions so that one doesn't have to worry too much about them. But when you do encounter a case where you need to (and this is frequent for any type of matrix manipulation), IDL is just too smart and you have to think twice to outsmart it ;-)

Anyway: you can always make sure you get what you want with

a = transpose(reform(a))

These statements are not very costly in terms of execution time, because it's only messing around with the array descriptor (at least I believe so).

Regards,
Martin.

--
-- -------
Dr. Martin Schultz
Department for Engineering&Applied Sciences, Harvard University
109 Pierce Hall, 29 Oxford St., Cambridge, MA-02138, USA

phone: (617)-496-8318
fax : (617)-495-4551

e-mail: mgs@io.harvard.edu
Internet-homepage: http://www-as.harvard.edu/people/staff/mgs/
-- -------