Subject: Can this be vectorized?
Posted by davis on Tue, 26 Oct 1999 07:00:00 GMT
View Forum Message <> Reply to Message

I am looking for either a matlab or IDL solution to this problem. Suppose that I have two 1-d arrays, `I' and `X', where `I' is an integer array and `X' is a floating point array. `I' is assumed to be sorted in ascending order. I would like to produce a third array `Y' that is formed from the elements of `X' as follows (pseudocode):

```
   len = length (X);       #number of elements of X

   i = 0;
   j = 0;

   while (i < len)
     {
        last_I = I[i];
        sum = X[i];
 i = i + 1;
 while ((i < len)
        AND (I[i] == last_I))
   {
      sum = sum + X[i];
      i = i + 1;
   }
 Y[j] = sum;
 j = j + 1;
     }
```

For example, suppose

```
    I = [ 1 2 3 3 4 4 4 5]
    X = [ a b c d e f g h]
```

Then, Y would be 5 element array:

```
   Y = [a b (c+d) (e+f+g) h]
```

One partially vectorized pseudocode solution would be:

```
   jj = 0
   for (i = min(I) to max(I))
     {
        J = WHERE (I == i);
 Y[jj] = sum_elements (X[J])
 jj = jj + 1
     }
```

What is the best way to vectorize this? In reality, X consists of
about one million elements, so I would prefer a solution that is
memory efficient. I apologize for posting to both newsgroups, but I
am looking for a solution in either language.

Thanks,
--John

Subject: Re: Can this be vectorized?
Posted by Richard G. French on Tue, 02 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

I forgot to include the printed results!

```
>
>  ; print the results
>
>  print,'i=',i
>  print,'x=',x
>  print,'y=',result
>  end
```

```
i= 0      1      1      2      3      4      4      4      5
x=-3.0   5.00    2.50    7.00   12.00   -4.00    10.00   2.30    7.00
y=-3.00      7.50      7.00    12.00     8.30     7.00
```

Dick French
Astronomy Dept.
Wellesley College

Subject: Re: Can this be vectorized?
Posted by Richard G. French on Tue, 02 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

"John E. Davis" wrote:

```
>
>  I am looking for either a matlab or IDL solution to this problem.
>  Suppose that I have two 1-d arrays, `I' and `X', where `I' is an integer
>  array and `X' is a floating point array.  `I' is assumed to be sorted in
>  ascending order.  I would like to produce a third array `Y' that is
>  formed from the elements of `X' as follows (pseudocode):
>
>     len = length (X);      #number of elements of X
>
>     i = 0;
>    j = 0;
>
>    while (i < len)
>      {
>         last_I = I[i];
>         sum = X[i];
>         i = i + 1;
>         while ((i < len)
>                AND (I[i] == last_I))
>           {
>              sum = sum + X[i];
>              i = i + 1;
>           }
>         Y[j] = sum;
>         j = j + 1;
>      }
>
```

> For example, suppose
>

```
>     I = [ 1 2 3 3 4 4 4 5]
>     X = [ a b c d e f g h]
>
```

> Then, Y would be 5 element array:
>

```
>    Y = [a b (c+d) (e+f+g) h]
>
```

> One partially vectorized pseudocode solution would be:
>

```
>    jj = 0
>    for (i = min(I) to max(I))
>      {
>         J = WHERE (I == i);
>         Y[jj] = sum_elements (X[J])
>         jj = jj + 1
>      }
>
```

> What is the best way to vectorize this? In reality, X consists of
> about one million elements, so I would prefer a solution that is
> memory efficient. I apologize for posting to both newsgroups, but I
> am looking for a solution in either language.
>
> Thanks,
> --John

John - I have an IDL solution that is not completely vectorize but which at least does vectorize filling the cases in which there is only one contributor to the sum. I have not tried it out extensively but I'd be interested in knowing if it saves you any time on your million-point runs:

```
i=[0,1,1,2,3,4,4,4,5]
x=[-3,5,2.5,7.,12.,-4.,10.,2.3,7]
; find indices in I array for which neighbors differ
; do this for upper and lower end
ishift=shift(i,1)
jshift=shift(i,-1)
li=where(i ne ishift,nli)
lj=where(i ne jshift)

result=fltarr(nli) ; save storage for final answer

; fill elements that have only one contributor

ll=where(li eq lj,nll)
if nll gt 0 then result(ll)=x[li[ll]]

; sum up elements where there are more than one

lm=where(li ne lj,nlm)
if nlm gt 0 then $
 for n=0,nlm-1 do begin
  k=lm[n]
  result[k]=total(x[li[k]:lj[k]])
 endfor

; print the results

print,i
print,x
print,result
end
```

Subject: Re: Can this be vectorized?
Posted by Gautam Sethi on Wed, 03 Nov 1999 08:00:00 GMT

View Forum Message <> Reply to Message

here is another loop version. it is considerably smaller than dick's and your pseudo-code.

```
 ------------------------------------------------------------- ------
function Y = davis(I,X)

LI = length(I); UI = unique(I); LUI = length(UI); Y = zeros(LI,LUI);
for i = 1:LUI
   Y(find(I/i == 1),i) = 1;
```

```
end

Y = X*Y;
```

--- -----

```
: "John E. Davis" wrote:
:>
:> I am looking for either a matlab or IDL solution to this problem.
:> Suppose that I have two 1-d arrays, `I' and `X', where `I' is an integer
:> array and `X' is a floating point array.  `I' is assumed to be sorted in
:> ascending order.  I would like to produce a third array `Y' that is
:> formed from the elements of `X' as follows (pseudocode):
:>
:>    len = length (X);      #number of elements of X
:>
:>    i = 0;
:>    j = 0;
:>
:>    while (i < len)
:>      {
:>         last_I = I[i];
:>         sum = X[i];
:>         i = i + 1;
:>         while ((i < len)
:>                AND (I[i] == last_I))
:>           {
:>              sum = sum + X[i];
:>              i = i + 1;
:>           }
:>         Y[j] = sum;
:>         j = j + 1;
:>      }
:>
:> For example, suppose
:>
:>     I = [ 1 2 3 3 4 4 4 5]
:>     X = [ a b c d e f g h]
:>
:> Then, Y would be 5 element array:
:>
:>    Y = [a b (c+d) (e+f+g) h]
:>
:> One partially vectorized pseudocode solution would be:
:>
:>    jj = 0
:>    for (i = min(I) to max(I))
:>      {
:>         J = WHERE (I == i);
:>         Y[jj] = sum_elements (X[J])
:>         jj = jj + 1
:>      }
:>
```

:> What is the best way to vectorize this? In reality, X consists of
:> about one million elements, so I would prefer a solution that is
:> memory efficient. I apologize for posting to both newsgroups, but I
:> am looking for a solution in either language.
:>
:> Thanks,
:> --John

: John - I have an IDL solution that is not completely vectorize but which
: at least does vectorize filling the cases in which there is only one
: contributor to the sum. I have not tried it out extensively but I'd be
: interested in knowing if it saves you any time on your million-point
: runs:

```
: i=[0,1,1,2,3,4,4,4,5]
: x=[-3,5,2.5,7.,12.,-4.,10.,2.3,7]
: ; find indices in I array for which neighbors differ
: ; do this for upper and lower end
: ishift=shift(i,1)
: jshift=shift(i,-1)
: li=where(i ne ishift,nli)
: lj=where(i ne jshift)

: result=fltarr(nli) ; save storage for final answer

: ; fill elements that have only one contributor

: ll=where(li eq lj,nll)
: if nll gt 0 then result(ll)=x[li[ll]]

: ; sum up elements where there are more than one

: lm=where(li ne lj,nlm)
: if nlm gt 0 then $
:  for n=0,nlm-1 do begin
:   k=lm[n]
:   result[k]=total(x[li[k]:lj[k]])
:  endfor

: ; print the results

: print,i
: print,x
: print,result
```

: end