Subject: Approximate convolution - for loop problem
Posted by samuel.leach on Sun, 21 Dec 2008 17:32:40 GMT

View Forum Message <> Reply to Message

Hello everyone, I'm trying to execute a 1-d convolution of an array, signal.

Using an analytic approximation, obtaining the convolved bolometer signal, bolo_signal, at time step ii, is given by the following:

```
nsamp=n_elements(signal)
const1 = exp(-tsamp/taubolo)
const2 = 1.-const1



bolo_signal = const2*signal
for ii= 1L,nsamp-1L do begin
   bolo_signal[ii] += const1*bolo_signal[ii-1]
endfor
```

where tsamp and taubolo are scalars. Is there any way to avoid the for loop in this case? The hope is to speed up the execution.

Many thanks for your help

Sam Leach

Subject: Re: Approximate convolution - for loop problem
Posted by Jeremy Bailin on Wed, 24 Dec 2008 21:20:16 GMT

View Forum Message <> Reply to Message

On Dec 23, 9:27 pm, Jeremy Bailin <astroco...@gmail.com> wrote:

```
>  On Dec 21, 3:01 pm, Sam <samuel.le...@gmail.com> wrote:
>
>
>
>>  Hi David, unfortunately shift() does not do the business for me, as
>>  these two examples below show. So I'm still a bit stumped here.
>
>>  ; Array operation I'm trying to execute.
>>  a=[1.,2.,3.,4.]
>>  for ii=1,3 do a[ii] += 0.5*a[ii-1]
>>  print,a
>>   1.00000      2.50000      4.25000      6.12500
>
>>  ; Attempt to perform this operation with shift()
>>  a=[1.,2.,3.,4.]
```

>> a += 0.5*shift(a,-1)
>> print,a
>> 2.00000 3.50000 5.00000 4.50000
>
>> On Dec 21, 7:03 pm, David Fanning <n...@dfanning.com> wrote:
>
>>> samuel.le...@gmail.com writes:
>>>> Hello everyone, I'm trying to execute a 1-d convolution of an array,
>>>> signal.
>
>>>> Using an analytic approximation, obtaining the convolved bolometer
>>>> signal, bolo_signal, at time step ii, is given by the following:
>
>>>> nsamp=n_elements(signal)
>>>> const1 = exp(-tsamp/taubolo)
>>>> const2 = 1.-const1
>
>>>> bolo_signal = const2*signal
>>>> for ii= 1L,nsamp-1L do begin
>>>>   bolo_signal[ii] += const1*bolo_signal[ii-1]
>>>> endfor
>
>>>> where tsamp and taubolo are scalars. Is there any way to avoid the for
>>>> loop in this case? The hope is to speed up the execution.
>
>>> I think this gives you the same results:
>
>>>   bolo_signal += const1 * shift(bolo_signal,-1)
>
>>> Cheers,
>
>>> David
>>> --
>>> David Fanning, Ph.D.
>>> Fanning Software Consulting, Inc.
>>> Coyote's Guide to IDL Programming:http://www.dfanning.com/
>>> Sepore ma de ni thui. ("Perhaps thou speakest truth.")
>
> How about this:
>
> a = [1.,2.,3.,4.]
> n = n_elements(a)
> c = 0.5^reverse(indgen(n))
> new_a = total(a*c, /cumulative) / c
>
> -Jeremy.

Of course, there are issues. Here is a test that shows that it works and is faster than the for loop:

```
pro test

n = 500l
seed = 2

c = double(randomu(seed))
a = randomu(seed, n)
b = a

t1 = systime(/sec)
for ii=1l,n-1 do a[ii] += c * a[ii-1]
t2 = systime(/sec)
print, 'For loop', t2-t1

t3 = systime(/sec)
carray = c^reverse(indgen(n))
new_a = total(b*carray, /cumulative) / carray
t4 = systime(/sec)
print, 'total(/cumulative)',t4-t3

print, 'Max deviation',max(abs(a-new_a))

end
```

And here's what I get:

```
For loop   0.00079083443
total(/cumulative)   0.00019907951
Max deviation   7.1814603e-08
```

So a factor of 4 speed improvement. Of course, n=500 isn't that big, and therein lies the problem. The code precomputes c^(n-1) and divides by it... so as soon as you get a floating underflow in c^(n-1), the algorithm returns NaNs. If your n is so large that Wox's method (which mine is obviously based on, to some degree) runs you out of memory, then it's probably also so large that my method causes an underflow. Anyone have any suggestions to get around that?

-Jeremy.

Subject: Re: Approximate convolution - for loop problem
Posted by David Gell on Fri, 02 Jan 2009 19:12:11 GMT
View Forum Message <> Reply to Message

On Dec 21 2008, 11:32 am, samuel.le...@gmail.com wrote:

> Hello everyone, I'm trying to execute a 1-d convolution of an array,
> signal.
>
> Using an analytic approximation, obtaining the convolved bolometer
> signal, bolo_signal, at time step ii, is given by the following:
>
> ```
> nsamp=n_elements(signal)
> const1 = exp(-tsamp/taubolo)
> const2 = 1.-const1
>
> bolo_signal = const2*signal
> for ii= 1L,nsamp-1L do begin
>    bolo_signal[ii] += const1*bolo_signal[ii-1]
> endfor
> ```
>
> where tsamp and taubolo are scalars. Is there any way to avoid the for
> loop in this case? The hope is to speed up the execution.
>
> Many thanks for your help
>
> Sam Leach

An alternative way requires building an upper triangular matrix of
powers of the second constant

```
anA=[1.,2.,3.,4]
nConst1 = 1.0
nConst2 = 0.5

nEle=n_elements(anA)

;bulid a upper diagonal matrix from the constants
anMatrix = fltarr(nEle,nEle)
for nI=0,nEle-1 do anMatrix[nI:*,nI] = nConst2^indgen(nEle-nI)

;compute result
anResult = anA ## anMatrix
help, anResult
print, anResult
end

% Compiled module: $MAIN$.
ANRESULT        FLOAT     = Array[4]
```

```
    1.00000      2.50000      4.25000      6.12500
```

The problem is building the upper triangular matrix.If that could be done efficiently, then this might speed things up.

David Gell
Southwest Research Institute
San Antonio, TX