Subject: Convolution of Stick Spectra
Posted by tclement on Sat, 09 Sep 2000 22:28:49 GMT
View Forum Message <> Reply to Message

Hi all...

In my on-going effort to speed up the code in our lab, I have another 'challenge' for you (I have to put 'challenge' in quotes because it seems that no matter what I ask, someone knows the ansewr off (or at least nearly off) the top of their head!)

We have a situation where we need to convolute (with energy dependent gaussians) a number of stick spectra on a well-defined energy axis. The stick spectra are read in from another program as a 2-dimensional array, using ddread. The [0,*] elements are the energies of the sticks, and the [1,*] values are the intensities. These have no inherent spacing, they are just calculated intensities at whatever energy the calculation returns.

We want to convolute these with gaussians on a regular energy axis, so we declare our energy array from 0 to 4, as shown below.

The only way I've found to do the convolution, since the x axes don't match on the two arrays, is to do the following (slow) for loop:

```
convoluted = fltarr( 2, 2048 )
convoluted[0,*] = findgen( 2048 ) / 2047. * 4.

;; Let's fake a stick spectrum, we usually have at least this many elements
stick = abs(randomn( systime(1), 2, 5000 ))
stick[0,*] = stick[0,*] * 4.
stick[1,*] = stick[1,*] * 1000.

for i=0L, n_elements( stick ) / 2 -1 do $
   convoluted[1,*] = stick[1,i]*exp(-((convoluted[0,*] - stick[0,i])^2)/ $
            (((.12*sqrt(stick[0,i]/1000))/1.6651)*1000)^2) $
            + convoluted[1,*]
```

So the question becomes, is there any way to speed this up? I thought of using a larger convoluted array with multiple dimensions and using total() somehow, but I couldn't think of how to do that.

Thanks in advance (and in retrospect) for all the help!
Todd

Subject: Re: Convolution of Stick Spectra

Posted by Craig Markwardt on Sun, 17 Sep 2000 07:00:00 GMT
View Forum Message <> Reply to Message

mole6e23@hotmail.com (Todd Clements) writes:

> craigmnet@cow.physics.wisc.edu wrote:
>
>> After looking at your problem it looks like the widths of the lines
>> are as wide as your energy range. I guess this would be appropriate
>> [snip..]
>> so many exponentiations. You could have gotten a pretty big savings
>> if the lines were narrow, and you could restrict the computation to a
>> narrow region around the line center (say +/- 10 sigma with WHERE). I
>> recommend that anyway to get rid of underflow errors.
>> [snip..]
>> If for example your sigma term were
>> (((.12*sqrt(stick[0,i]/1000))/1.6651)*10) [note the last factor is
>> smaller] then things start to look interesting.
>
> That's because there's always a danger in taking code x and modifying it
> into simpler example y! As you suggested, I actually DO have a sigma term
> with a *10 instead of a *1000.
>
> I did what you suggested with the +/- 10*sigma with where (code below),
> and it drastically improved running time. For the 5000 element array, the
> running time went from 62.3 seconds to 4.9 seconds! I plot the error
> associated with the method, and I was actually able to go down to +/-
> 4*sigma before there was any noticable error.
>
> Thanks!
> Todd

You're welcome. But call me a pedant too. You can save yet more
computations by precomputing the gaussian argument.

Craig

```
xx = convoluted(0,*)
yy = convoluted(1,*)
for i = 0L, n_elements(stick)/2 -1 do begin
  z1 = (xx-stick(0,i))/(((.12*sqrt(stick[0,i]/1000))/1.6651)*10)^2
  wh = where(z1 LT (2.*4^2), ct)  ;; Four sigma (remember the 1/2!)
  if ct GT 0 then yy(wh) = yy(wh) + stick(1,i)*exp(-z1)
endfor
convoluted(1,*) = yy
```

I hope I got it all right, but you get the idea. By the way, as I
note above, the definition of the gaussian is exp(-x^2/(2*sigma^2)).
Is the 1/2 buried somewhere in your definition?

--
 -- --------------
Craig B. Markwardt, Ph.D.   EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 -- --------------

Subject: Re: Convolution of Stick Spectra
Posted by mole6e23 on Sun, 17 Sep 2000 07:00:00 GMT
View Forum Message <> Reply to Message

craigmnet@cow.physics.wisc.edu wrote:

> After looking at your problem it looks like the widths of the lines
> are as wide as your energy range. I guess this would be appropriate
> [snip..]
> so many exponentiations. You could have gotten a pretty big savings
> if the lines were narrow, and you could restrict the computation to a
> narrow region around the line center (say +/- 10 sigma with WHERE). I
> recommend that anyway to get rid of underflow errors.
> [snip..]
> If for example your sigma term were
> (((.12*sqrt(stick[0,i]/1000))/1.6651)*10) [note the last factor is
> smaller] then things start to look interesting.

That's because there's always a danger in taking code x and modifying it
into simpler example y! As you suggested, I actually DO have a sigma term
with a *10 instead of a *1000.

I did what you suggested with the +/- 10*sigma with where (code below),
and it drastically improved running time. For the 5000 element array, the
running time went from 62.3 seconds to 4.9 seconds! I plot the error
associated with the method, and I was actually able to go down to +/-
4*sigma before there was any noticable error.

Thanks!
Todd

--

```
pro test

 convoluted = fltarr( 2, 2048 )
 convoluted[0,*] = findgen( 2048 ) / 2047. * 4.
 convoluted2 = convoluted

 ;; Let's fake a stick spectrum, we usually have at least this many elements
 stick = abs(randomn(10, 2, 5000 ))
 stick[0,*] = stick[0,*] * 4.
 stick[1,*] = stick[1,*] * 1000.

 time=systime(1)
 for i=0L, n_elements( stick ) / 2 -1 do $
      convoluted[1,*] = stick[1,i]*exp(-((convoluted[0,*] - stick[0,i])^2)/ $
                  (((.12*sqrt(stick[0,i]/1000))/1.6651)*10)^2) $
                  + convoluted[1,*]

 time2 = systime(1)
 for i=0L, n_elements( stick ) / 2 -1 do begin
     ;; Find range
     sigma = (((.12*sqrt(stick[0,i]/1000))/1.6651)*10)
     range = where( convoluted2[0,*] ge (stick[0,i]-4*sigma) and $
                    convoluted2[0,*] le (stick[0,i]+4*sigma) )

     if( range[0] ne -1 ) then $
       convoluted2[1,range] = stick[1,i]*exp(-((convoluted2[0,range] -  $
                              stick[0,i])^2)/sigma^2) $
                              + convoluted2[1,range]
 endfor

 print, 'time old: ', time2-time
 print, 'time new: ', systime(1)-time2

 new = convoluted
 new[1,*] = new[1,*] - convoluted2[1,*]

 plot,new[0,*], new[1,*],yrange=[-10,10]

end ;; test
```

Subject: Re: Convolution of Stick Spectra
Posted by Craig Markwardt on Sun, 17 Sep 2000 07:00:00 GMT

View Forum Message <> Reply to Message

Hi Todd--

After looking at your problem it looks like the widths of the lines
are as wide as your energy range. I guess this would be appropriate
if you were using proportional counters to detect X-ray lines :-)
Because of this it doesn't seem like you need a super accurate
representation of the gaussian.

The biggest cost appears to be associated with the computation of the
EXP() function, so you will need to reduce the number of computations.

Vectorizing further probably won't help since you are compute-bound by
so many exponentiations. You could have gotten a pretty big savings
if the lines were narrow, and you could restrict the computation to a
narrow region around the line center (say +/- 10 sigma with WHERE). I
recommend that anyway to get rid of underflow errors.

I tried a spline interpolation, based on a precomputed gaussian
temlate, but again given the broadness of your lines, it doesn't
really save anything.

If for example your sigma term were
(((.12*sqrt(stick[0,i]/1000))/1.6651)*10) [note the last factor is
smaller] then things start to look interesting.

Why isn't your energy grid coarser? Why aren't you using an honest to
goodness response matrix?

Cheers,
Craig

tclement@ucsd.edu (Todd Clements) writes:

```
>  Hi all...
>
>  In my on-going effort to speed up the code in our lab, I have another
>  'challenge' for you (I have to put 'challenge' in quotes because it seems
>  that no matter what I ask, someone knows the ansewr off (or at least
>  nearly off) the top of their head!)
>
>  We have a situation where we need to convolute (with energy dependent
>  gaussians) a number of stick spectra on a well-defined energy axis. The
>  stick spectra are read in from another program as a 2-dimensional array,
>  using ddread. The [0,*] elements are the energies of the sticks, and the
>  [1,*] values are the intensities. These have no inherent spacing, they are
>  just calculated intensities at whatever energy the calculation returns.
>
>  We want to convolute these with gaussians on a regular energy axis, so we
>  declare our energy array from 0 to 4, as shown below.
>
>  The only way I've found to do the convolution, since the x axes don't
>  match on the two arrays, is to do the following (slow) for loop:
>
>  convoluted = fltarr( 2, 2048 )
>  convoluted[0,*] = findgen( 2048 ) / 2047. * 4.
>
>  ;; Let's fake a stick spectrum, we usually have at least this many elements
>  stick = abs(randomn( systime(1), 2, 5000 ))
>  stick[0,*] = stick[0,*] * 4.
>  stick[1,*] = stick[1,*] * 1000.
>
>  for i=0L, n_elements( stick ) / 2 -1 do $
>      convoluted[1,*] = stick[1,i]*exp(-((convoluted[0,*] - stick[0,i])^2)/ $
>                   (((.12*sqrt(stick[0,i]/1000))/1.6651)*1000)^2) $
>                   + convoluted[1,*]
>
>
```

> So the question becomes, is there any way to speed this up? I thought of
> using a larger convoluted array with multiple dimensions and using total()
> somehow, but I couldn't think of how to do that.
>
> Thanks in advance (and in retrospect) for all the help!
> Todd

--
 --- --------------
Craig B. Markwardt, Ph.D.   EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 --- --------------

Subject: Re: Convolution
Posted by Alex Schuster on Wed, 12 Sep 2001 14:20:38 GMT
View Forum Message <> Reply to Message

Kay Bente writes:

> I have to convolute a 256x256x128 Floating Point array with a 3D Gaussian
> Kernel of ~ 30x30x30, this lasts round about 45Minutes. So my question is,
> if there is any way how i can speed this up. I tried to separate this in
> each dimension with a 1D Kernel, but I donï¿½t know if I have done this
> correct (cause the procedure hangs up after a few loops)
>
> I know that the Convolution of two functions is a Multiplication in Fourier
> Space, but how can I do this with discrete arrays, do I have to enlarge my
> kernel to the size of the array i want to smooth? If so, the creation of the
> kernel with the dimensions of my array nearly lasts as long as the normal
> convolution :-(

I use the routine PSF_GAUSSIAN() to create these kernels, speed is no
problem there. The kernel has the same size as the orignal image, but
that's no problem in fourier space.
Here is some code I ripped from one of my programs. Computation takes
some seconds, not 45 minutes :)

```
pix = [ aat.x_pixel_size, aat.y_pixel_size, aat.z_pixel_size ]
r = float( radius ) / 10.0
```

```
xywidth = long( 0.5 + r/pix[0] )
zwidth = long( 0.5 + r/pix[2])

; dim[0] and dim[1] are powers of 2, make make dimz a power of 2, too,
; and use it instead of dim[2]
dimz = 4
while ( dim[2] ge dimz ) do dimz = dimz * 2
startz = (dimz-dim[2]) / 2

filter_kernel = complexarr( dim[0], dim[1], dimz )
filter_kernel[0,0,0] = psf_gaussian( $
   npixel=[dim[0]-1,dim[1]-1,dimz-1], $
   ndimen=3, $
   fwhm=[xywidth, xywidth, zwidth], /normalize )
filter_kernel = fft( shift( temporary( filter_kernel ), $
dim[0]/2+1, dim[1]/2+1, dimz/2+1 ) )

filt_image = complexarr( dim[0], dim[1], dimz )
filt_image[dim[0]*dim[1]*startz] = image
filt_image = fft( fft( temporary( filt_image ) ) * filter_kernel,
/inverse ) $
    * filter_mask
```

        Alex
--
  Alex Schuster   Wonko@planet-interkom.de
                    alex@pet.mpin-koeln.mpg.de

Subject: Re: Convolution
Posted by Jaco van Gorkom on Wed, 12 Sep 2001 14:22:38 GMT
View Forum Message <> Reply to Message

Kay Bente wrote:
> I have to convolute a 256x256x128 Floating Point array with a 3D Gaussian
> Kernel of ~ 30x30x30, this lasts round about 45Minutes. So my question is,
> if there is any way how i can speed this up. I tried to separate this in
> each dimension with a 1D Kernel, but I donï¿½t know if I have done this
> correct (cause the procedure hangs up after a few loops)

Manually looping through slices of the array shouldn't be necessary. It is
possible to use 3D kernel arrays which extend over only one dimension:
```
  kernel_x = fltarr(30,1,1)
  kernel_y = fltarr(1,30,1)
  kernel_z = fltarr(1,1,30)
```
and then just apply three convol statements like
```
  iresult = convol(             array, kernel_x)
  iresult = convol(temporary(iresult), kernel_y)
  result  = convol(temporary(iresult), kernel_z)
```
The only tricky bit is that IDL tends to remove the trailing dimension from kernel_y at inconvenient times, most notably inside CONVOL if its type needs to be converted.

I coded up a general function for the application of a 1D kernel to each dimension of an n-dimensional array, see below. It cracks your array size in 52 seconds on my system, versus longer-than-coffee for the normal 3D CONVOL.

The multiplication in Fourier space sounds promising as well, maybe someone else can comment?

Regards,
  Jaco

```
function symconvol, array, kernel, scale_factor, _ref_extra=extra
  ; Similar to CONVOL, can be of use for fully circularly symmetric,
  ; cubic kernels (e.g. Gauss). Applies 1D Kernel along each of the
  ; dimensions of Array. This should be faster than a direct 3D
  ; convolution for all but the smallest kernel sizes.
  ; Note: if scale_factor is specified, it will be applied multiple times
  ; (once for each dimension).
  ; Other arguments as for convol.
  ; written 12 Sept. 2001 by Jaco van Gorkom (gorkom@rijnh.nl),
  ;                    based on code by Kay Bente.

    if size(kernel, /n_dimensions) ne 1 then $
      message, 'One-dimensional vector expected for input parameter Kernel.'

    ; fix kernel data type to avoid losing trailing unit dimensions in
    ; later conversions:
    ikernel = fix(kernel, type=size(array,/type))
    ndims = size(array, /n_dimensions)
    kernelsize = n_elements(ikernel)
    kerneldims = replicate(1L, ndims)

    for dimcnt=0L, ndims-1 do begin
        ; make the ikernel extend along the current dimension:
        kerneldims[dimcnt] = kernelsize
        ikernel = reform(ikernel, kerneldims, /overwrite)
        ; convolve the input array (in the first step) or the intermediate
        ; result with ikernel:
        if dimcnt eq 0L then $
          if n_params() eq 3 then $
            result = convol(array, ikernel, scale_factor, _extra=extra) $
          else $
            result = convol(array, ikernel, _extra=extra) $
        else $
```

```
          if n_params() eq 3 then $
            result = convol(temporary(result), ikernel, scale_factor, $
              _extra=extra) $
          else $
            result = convol(temporary(result), ikernel, _extra=extra)
        kerneldims[dimcnt] = 1L
    endfor

   return, result
end
```