
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 don't 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

```
