Subject: Final Solution of Convol with Radius dependency
Posted by bente on Wed, 07 Nov 2001 15:41:33 GMT
View Forum Message <> Reply to Message

Hi, I just want to present my final solution of the Convolution ;-)
I takes only 50seconds on a 128x128x64 array but round about 10minutes
on a 256x256x128 array.

You can determine if you want no convolution, "normal", or with
r-dependency for each dimension (only with 3D-arrays, had no time to
care about 1 or 2 dimensions ;-)

Wrote it for Gaussian Kernel, but i build the kernel in sub-function
so it sshould be easy to replace it with any other kernel (in the case
that someone should need such stuff, too.

And I donï¿½t understand, why the time is so much larger than with the
small array. I found out that it already takes ages to copy the array
in a bit larger array (so i have space at the edges for the size of
the kernel).

So have fun, if someone needs it, or answer if someone has a faster
way ;-)

Kay

```
 ------------------------------------------------------------- ------------------
;----------------------3Dimensional Convolution with Dependency Of The
Radius----------------------------------
;------Kay Bente-------
;----------------------

 ;------------------------------------------------------------ --------
;--------------------Functions Needed------------------------------
 ;------------------------------------------------------------ --------


FUNCTION Kernel_Size, scale, fac, fw

;Determines the Size of The Kernel from FWHM and a Faktor and the
Scaling of the Dataset (PIXEL <-> mm)
;e.g. scale = [0.9,0.9,1.25]
;      FWHM = 6mm
;       fac = 3  (Kernel should be 3times larger then the FWHM
;Kernel_Size= [21,21,15]

; written by Kay Bente
; 11.9.01
```

```
 s=Ceil(fac*fw/scale)

 ;Round Up to next largest Odd integer
 s=s+1-(s MOD 2)

Return, s
END

FUNCTION Gauss, s, sig, scale=scale, fwhm=fwhm
;Create 3D Gaussian
;Kay Bente, Wuppertal: 06.09.01 - 11.09.01

;s -> [nx,ny,nz]
;sigma -> Sigma for Gauss
;scale -> [sx,sy,sz] for example for MRI images e.g. : [0.9,0.9,1.25]
;fwhm=1 -> Caluculate with FWHM instead of sigma,
FWHM=2*sigma*Sqrt(2*ALog(2)) => sigma~FWHM/2.35
;          sigma input in FWHM

;Determine Dimensions
ss=Size(s, /Dimensions)


CASE ss(0) OF
          0 : GOTO, label_1D
          1 : GOTO, label_1D
        2 : GOTO, label_2D
        3 : GOTO, label_3D
     ELSE : BEGIN
                Print, 'Wrong Dimensions!'
                Print, 'Size(s, /Dimensions) Has To Be Less Or Equal
3'
                Print, 'You Entered :',s
                Print, String('That has ',StrTrim(String(ss),1), '
Dimensions!')
                Print, '"0" Returned!'
                Return, 0
                GOTO, label_END
           ENDELSE
ENDCASE

;1D
label_1D:

;Check Keywords
IF Keyword_Set(scale) EQ 0 THEN scale = [1]
IF Keyword_Set(fwhm) THEN sigma = sig/2.35 ELSE sigma = sig
```

```
;Create Indizes
arr  = (FindGen(s(0)) - s(0)/2)*scale(0)
;Create Gaussian on Indizes
arr2 = Exp(-(Temporary(arr))^2/(2.*sigma^2))/(Sqrt(2.*!PI)*sigma)
Return,arr2
GOTO, label_END

;2D Adapted from 3D
label_2D:

;Check Keywords
IF Keyword_Set(scale) EQ 0 THEN scale = [1,1]
IF Keyword_Set(fwhm) THEN sigma = sig/2.35 ELSE sigma = sig

;Faktor 1 for odd size
f=[s(0) mod 2, s(1) mod 2]

mid=[s(0)/2 + f(0),s(1)/2 + f(1)]
arr=FltArr(mid(0),mid(1))

fak=2.*sigma^2
fak2=2.*!pi*sigma^2

;Create 1/4  Circle
FOR i=0,mid(0)-1 DO BEGIN
 FOR j=0,mid(1)-1 DO BEGIN
  arr(i,j) = Exp(-((i*scale(0))^2+(j*scale(1))^2)/fak)/fak2
 ENDFOR
ENDFOR

arr2=FltArr(s(0),s(1))

;Copy & Mirror To Fill The Rest Of Circle
arr2(mid(0)-f(0):s(0)-1,mid(1)-f(1):s(1)-1)=arr
arr2(mid(0)-f(0):s(0)-1,0:mid(1)-1)=Reverse(arr,2)
 arr2(0:mid(0)-1,0:s(1)-1)=Reverse(arr2(mid(0)-f(0):s(0)-1,0: s(1)-1),1)

Return, arr2
GOTO, label_END

;3D
label_3D:

;Check Keywords
IF Keyword_Set(scale) EQ 0 THEN scale = [1,1,1]
IF Keyword_Set(fwhm) THEN sigma = sig/2.35 ELSE sigma = sig
```

```
;Faktor 1 for odd size
f=[s(0) mod 2, s(1) mod 2, s(2) mod 2]

mid=[s(0)/2 + f(0),s(1)/2 + f(1),s(2)/2 + f(2)]
arr=FltArr(mid(0),mid(1),mid(2))

fak=2.*sigma^2
fak2=(2.*!pi)^(3./2.)*sigma^3

;Create 1/8  Sphere
FOR i=0,mid(0)-1 DO BEGIN
 FOR j=0,mid(1)-1 DO BEGIN
  FOR k=0,mid(2)-1 DO BEGIN
   arr(i,j,k)=  Exp(-((i*scale(0))^2+(j*scale(1))^2+(k*scale(2))^2)/fak)/fak 2
  ENDFOR
 ENDFOR
ENDFOR

arr2=FltArr(s(0),s(1),s(2))

;Mirror Subarrays to create whole Sphere
;   3 2 1                1 2 3                0 0 3
;a= 0 0 2 -> Reverse(a,1)= 2 0 0     Reverse(a,2)= 0 0 2
;   0 0 3                3 0 0                3 2 1

 arr2(mid(0)-f(0):s(0)-1,mid(1)-f(1):s(1)-1,mid(2)-f(2):s(2)- 1)=arr
 arr2(mid(0)-f(0):s(0)-1,0:mid(1)-1,mid(2)-f(2):s(2)-1)=Rever se(arr,2)
 arr2(0:mid(0)-1,0:s(1)-1,mid(2)-f(1):s(2)-1)=Reverse(arr2(mi
d(0)-f(0):s(0)-1,0:s(1)-1,mid(2)-f(1):s(2)-1),1)
 arr2(0:s(0)-1,0:s(1)-1,0:mid(2)-1)=Reverse(arr2(0:s(0)-1,0:s (1)-1,mid(2)-f(2):s(2)-1),3)
Return, arr2

label_END:
END


 ;------------------------------------------------------------ ----------
;-----------------------Main Program--------------------------------
 ;------------------------------------------------------------ ----------


FUNCTION R_Wisch,im,scale,minn=minn,maxx=maxx
;Convolutes an array with Gaussian Kernel with Dependency To The
Radius
;
;Kay Bente, 7.11.01
;bente@uni-wuppertal.de
```

```
;needs: Kernel_Size.pro
;       Gauss.pro

;Syntax:
; image -> image to be smoothed
; scale -> factors from MRImage e.g. [0.9,0.9,1.25]
;  minn -> [min_x,min_y,min_z], minn has to be GE 1
;  maxx -> [max_x,max_y,max_z]

time=Systime(1)
array=Float(im)
ndims = Size(array, /n_dimensions)
s=Size(array, /Dimensions)
xx=s(0)
yy=s(1)
zz=s(2)

IF ndims GT 3 THEN BEGIN
   Print,'Pleasy only arrays with 3 or less dimensions'
   Print,'0 returned'
   Return, 0
ENDIF

kerneldims = Replicate(1L, ndims)


FOR dimcnt=0L, ndims-1 DO BEGIN

;------------------------Convol in
x-----------------------------------------------
 IF dimcnt EQ 0 THEN BEGIN
    IF maxx(0) EQ 0 THEN GOTO, stepout
    Print,'Convol in x'
    IF minn(0) EQ maxx(0) THEN GOTO, old_fashion
    ss_max=Kernel_Size(scale(dimcnt),10,maxx(0))
    l_arr=FltArr(xx+ss_max+1,yy,zz)
    l_arr((ss_max+1)/2,0,0)=array

      FOR t=0,xx-1 DO BEGIN
     r=Abs(xx/2.-t)         ;Get Radius
     fwhmm=minn(1)+((maxx(0)-minn(0))*r/(xx/2.))  ;Calculate FWHM
     ss=Kernel_Size(scale(dimcnt),10,fwhmm)   ;Calculate KernelSize
     kernel=Gauss(ss,fwhmm,scale=scale(dimcnt),/FWHM);Build Kernel
(1D)
     large_kernel=Rebin(kernel,ss,yy,zz)    ;Duplicate Kernel in y & z
      array(t,*,*)=Rebin(large_kernel*l_arr(t+(ss_max+1)/2-1-ss/2:
t+(ss_max+1)/2-1+ss/2,*,*),1,yy,zz)*ss
     ;Rebin calculates mean -> Multiplicate*ss
```

```
     ENDFOR
     GOTO, stepout
    ENDIF

;-------------------------Convol in
y-------------------------------------------
 IF dimcnt EQ 1 THEN BEGIN
 IF maxx(1) EQ 0 THEN GOTO, stepout
    Print,'Convol in y'
    IF minn(1) EQ maxx(1) THEN GOTO, old_fashion
    ss_max=Kernel_Size(scale(dimcnt),10,maxx(1))
    l_arr=FltArr(xx,yy+ss_max+1,zz)
    l_arr(0,(ss_max+1)/2,0)=array

       FOR t=0,yy-1 DO BEGIN
     r=Abs(yy/2.-t)         ;Get Radius
     fwhmm=minn(1)+((maxx(1)-minn(1))*r/(yy/2.))  ;Calculate FWHM
     ss=Kernel_Size(scale(dimcnt),10,fwhmm)   ;Calculate KernelSize
     kernel=Gauss(ss,fwhmm,scale=scale(dimcnt),/FWHM);Build Kernel
(1D)
     kernel=Reform(kernel,[1,ss,1])     ;Erect Kernel along y-axis
     large_kernel=Rebin(kernel,xx,ss,zz)    ;Duplicate Kernel in x & z
      array(*,t,*)=Rebin(large_kernel*l_arr(*,t+(ss_max+1)/2-1-ss/
2:t+(ss_max+1)/2-1+ss/2,*),xx,1,zz)*ss
     ;Rebin calculates mean -> Multiplicate*ss
     ENDFOR
     GOTO, stepout
    ENDIF


;-------------------------Convol in
z-------------------------------------------
    IF dimcnt EQ 2 THEN BEGIN
    IF maxx(2) EQ 0 THEN GOTO, stepout
       Print,'Convol in z'
       IF minn(2) EQ maxx(2) THEN GOTO, old_fashion
       ss_max=Kernel_Size(scale(dimcnt),10,maxx(2))
    l_arr=FltArr(xx,yy,zz+ss_max+1)
    l_arr(0,0,(ss_max+1)/2)=array

       FOR t=0,zz-1 DO BEGIN
     r=Abs(zz/2.-t)         ;Get Radius
     fwhmm=minn(2)+((maxx(2)-minn(2))*r/(zz/2.))  ;Calculate FWHM
     ss=Kernel_Size(scale(dimcnt),10,fwhmm)   ;Calculate KernelSize
     kernel=Gauss(ss,fwhmm,scale=scale(dimcnt),/FWHM);Build Kernel
(1D)
     kernel=Reform(kernel,[1,1,ss])     ;Erect Kernel along z-axis
     large_kernel=Rebin(kernel,xx,yy,ss)    ;Duplicate Kernel in x & y
```

```
      array(*,*,t)=Rebin(large_kernel*l_arr(*,*,t+(ss_max+1)/2-1-s
s/2:t+(ss_max+1)/2-1+ss/2),xx,yy,1)*ss
     ;Rebin calculates mean -> Multiplicate*ss
       ENDFOR
       GOTO, stepout
    ENDIF


old_fashion:
s=Kernel_Size(scale(dimcnt),3,maxx(dimcnt))
kernel=Gauss(s,maxx(dimcnt),scale=scale(dimcnt),/FWHM)

;Make 1D-Kernel 3D
kerneldims(dimcnt)=s
kernel = Reform(kernel, kerneldims, /overwrite)

; convolve the input array (in the first step) or the intermediate
; result with kernel:

array = Convol(Float(Temporary(array)), kernel)
kerneldims[dimcnt] = 1L ;Reset KernelDims to [1,1,1]

stepout:
ENDFOR

l_arr=0
Print,'Time needed: ',Systime(1)-time
Return, array
END
```