Subject: Convolution with non-constant Kernel?
Posted by SonicKenking on Fri, 12 Nov 2010 00:56:18 GMT
View Forum Message <> Reply to Message

Hi, I wonder if there is an easy way to perform convolution on an
array with non-constant kernel.

The IDL built-in CONVOL function requires the kernel to be a fixed
array, e.g.
[-1,2,-1]. I want to have a dynamic kernel that changes based on the
position of the array. Something like

array = [8,6,7,9,1,3,4,5], kernel=[sin(index_i-1), 2, sin(index_i+1)]

Is there any other built-in IDL function that can do this or is there
someone who has already coded this up? If the answer is no, I'll go
ahead and code my own program. Just checking it here beforehand to
avoid re-inventing wheels.

Thanks!

Subject: Re: Convolution with non-constant Kernel?
Posted by rogass on Fri, 12 Nov 2010 22:03:25 GMT
View Forum Message <> Reply to Message

On 12 Nov., 01:56, SonicKenking <ywa...@gmail.com> wrote:
> Hi, I wonder if there is an easy way to perform convolution on an
> array with non-constant kernel.
>
> The IDL built-in CONVOL function requires the kernel to be a fixed
> array, e.g.
> [-1,2,-1]. I want to have a dynamic kernel that changes based on the
> position of the array. Something like
>
> array = [8,6,7,9,1,3,4,5], kernel=[sin(index_i-1), 2, sin(index_i+1)]
>
> Is there any other built-in IDL function that can do this or is there
> someone who has already coded this up? If the answer is no, I'll go
> ahead and code my own program. Just checking it here beforehand to
> avoid re-inventing wheels.
>
> Thanks!

If you are interested. I have a routine which strictly performs
discrete convolutions for a 2d array and a 3d kernel without zero
padding, without loops and for small kernels faster than fft. Just
send me an email.

Cheers

CR

---

## Subject: Re: Convolution with non-constant Kernel?
Posted by Juggernaut on Sat, 13 Nov 2010 16:23:42 GMT
View Forum Message <> Reply to Message

On Nov 12, 5:03 pm, chris <rog...@googlemail.com> wrote:
> On 12 Nov., 01:56, SonicKenking <ywa...@gmail.com> wrote:
>
>
>
>> Hi, I wonder if there is an easy way to perform convolution on an
>> array with non-constant kernel.
>
>> The IDL built-in CONVOL function requires the kernel to be a fixed
>> array, e.g.
>> [-1,2,-1]. I want to have a dynamic kernel that changes based on the
>> position of the array. Something like
>
>> array = [8,6,7,9,1,3,4,5], kernel=[sin(index_i-1), 2, sin(index_i+1)]
>
>> Is there any other built-in IDL function that can do this or is there
>> someone who has already coded this up? If the answer is no, I'll go
>> ahead and code my own program. Just checking it here beforehand to
>> avoid re-inventing wheels.
>
>> Thanks!
>
> If you are interested. I have a routine which strictly performs
> discrete convolutions for a 2d array and a 3d kernel without zero
> padding, without loops and for small kernels faster than fft. Just
> send me an email.
>
> Cheers
>
> CR

These types of algorithms would be useful for implementing a tilt-
shift photography on a standard image.

---

## Subject: Re: Convolution with non-constant Kernel?
Posted by Gray on Wed, 17 Nov 2010 20:52:03 GMT

I'm trying to implement this sort of thing right now... the
approximation I'm using is to subdivide the image into pieces that are
small enough that the approximation of a constant kernel over a
subdivision is reasonably good (which for my images is around 128px
square in a grid of 16x16 subdivisions), then reconstruct into the
full image. This seems like it could be good enough for my purposes
(matching spatially-varying PSFs between two images), but I don't know
if it works for what you want.

Subject: Re: Convolution with non-constant Kernel?
Posted by rogass on Fri, 19 Nov 2010 10:07:02 GMT

Hi, try the following routines for 2D images and 2D mask or mask sized
[maskx,masky,n_elements(image)],whereas boundary pixels are not
considered, large images and small masks will cause heavy memory
usage:

```
function cr_discrete_convolve,mask,im
szm = ulong(size(mask,/dimensions))
szi = ulong(size(im,/dimensions))
 return, reform(/
 over,total(total(im[cr_window(szi[0],szi[1],szm[0],szm[1])]* $
   n_elements(szm) eq 2? rebin(/
sample,mask,szm[0],szm[1],szi[0]*szi[1]):$
                mask,2),1),szi[0],szi[1])
end

function cr_window,sx,sy,wx,wy,norot=norot
wx = ulong(wx)
wy = ulong(wy)
sx = ulong(sx)
sy = ulong(sy)
base = ((ll=ulindgen(wx,wy) mod wx)) + transpose(ll)*sx
return, rebin(/sample,(keyword_set(norot)? temporary(base) : $
  rot(reform(/over,temporary(base),wx,wy),180)),wx,wy,sx*sy) + $
  rebin(/sample,ulindgen(1,1,sx*sy),wx,wy,sx*sy)
end
```

Cheers

CR