
Subject: Re: How to speed up kernel density smoothing for many data points

Posted by [Moritz Fischer](#) on Thu, 10 Oct 2013 12:55:21 GMT

[View Forum Message](#) <> [Reply to Message](#)

Hi Jacob!

Just a quick shot:

I've got a routine that returns pairwise distances (below).

I think you want

```
mat = matrix_euclidean_distance(x,x)
f1 = 1./float(N) * total( 1. (hopt^2)*K(mat,hopt),2)
```

Just remove the first line of your K function (mat already contains distances!) and make sure it respects the dimensions of mat (where returns one dimensional indices, but output should be N by N). note that mat is - in this case - symmetric, so the dimension argument of total(, dim) could as well be 1.

hope this helps
cheers

Am 10.10.2013 14:22, schrieb jacobsvensmark@gmail.com:

```
> Okay so I have a long array of N 2D points: x = fltarr(N,2) (as
> exemplified below). For each point x(i,*) I need to input the
> distance to all other points into my K function: K(xi-x), sum
> together K for each point, square root and put into my final function
> f1(i).
>
> I guess I need help both making the input xi-x faster, as well as
> making K run faster. I tried with 'REPLICATE_INPLACE' which helped
> some, but I am out of ideas....
>
> PRO test_kernel
>
> N = 1000000 r = 2*randomn(seed,N) v = 2*randomu(seed,N) x =
> [[r],[v]]
>
> ; Just a smoothing parameter, unimportant... hopt =
> 6.24/(N^(1./6.))*sqrt((stddev(x(*,0))^2+stddev(x(*,1))^2)/2. )
>
> ; Slow loop, where I need help f1 = fltarr(N) xi = fltarr(N,2) for
> i=0L,N-1 do begin REPLICATE_INPLACE, xi, x(i,0), 1, [1,0]
> REPLICATE_INPLACE, xi, x(i,1), 1, [0,1] f1(i) = 1./float(N) *
> total(1./(hopt^2)*K(xi-x,hopt)) endfor
>
> END
>
> FUNCTION K,tvec,h t = vec_norm(tvec)/h aa = where(t ge 1.,n0) bb
```

```

> = where(t lt 1.,nt) RES = fltarr(n_elements(t)) if n0 ne 0 then
> RES(aa) = 0. if nt ne 0 then RES(bb) = 4./!Pi*(1.-t(bb)^2)^3
> RETURN,res END
>
> Thanks, Jacob
>

```

```

;+
; NAME:
;   MATRIX_EUCLIDEAN_DISTANCE
;
; AUTHOR:
;   Fischer
;
; PURPOSE:
;   Returns a two dimensional array of distances.
;
; CALLING SEQUENCE:
;
;   result = matrix_euclidean_distance( p1, p2 )
;
; DESCRIPTION:
;
;   MATRIX_EUCLIDEAN_DISTANCE implements Euclidean metric for k
dimensional space.
;   The input format fits the output of my position handling routines, i.e.
;   p1 = dblarr( n, k ) and p2 = dblarr( m, k ),
;   where n and m are the number of k dimensional points in p1 and p2,
respectively.
;
; INPUTS:
;
;   p1, p2  arrays of IR^k points, nr_of_points x dimension
;           e.g. p1 = dblarr(n,3), p2 = dblarr(m,3)
;
; KEYWORDS:
;
;   SQUARED   if set, the square root is omitted (to avoid redundant
operations.)
;
; RETURNS:
;
;   arr( n , m )  Matrix of pairwise distances, where result[i,j] = ||
p1[i,*] - p2[j,*] ||
;

```

FUNCTION MATRIX_EUCLIDEAN_DISTANCE, p1, p2, SQUARED = SQUARED

```

s1 = size(p1) & s2 = size(p2)
IF s1[0] EQ 1 THEN p1 = REFORM(p1, 1, s1[1], /OVERWRITE )
IF s2[0] EQ 1 THEN p2 = REFORM(p2, 1, s2[1], /OVERWRITE )
s1 = size(p1) & s2 = size(p2)

dm = make_array(s1[1], s2[1], TYPE=s1[3])

; this loops over dimensions k, not elements!
FOR dim = 0, s1[2]-1 DO dm += $
(      rebin( reform(p1[*],dim)), s1[1], s2[1], /S ) - $
  transpose(rebin( reform(p2[*],dim)), s2[1], s1[1], /S )))^2

RETURN, keyword_set( SQUARED ) ? reform( dm ) : reform( sqrt(dm) )
END

```
