Subject: Re: Find all points within a set of polygons
Posted by JD Smith on Wed, 30 Nov 2005 20:45:15 GMT
View Forum Message <> Reply to Message

On Wed, 30 Nov 2005 07:14:15 -0800, Maarten wrote:

> Hi,
>
> I'm trying to compare satellite images from two different polar
> sun-synchronous satellites. One dataset is high resolution (1 by 1 km,
> from MODIS/Aqua, if you're curious), the other has a lower spatial
> resolution (though still pretty high at about 15 by 15 km, from OMI/Aura).
>
> I'd like to attach a number to each MODIS pixel telling me in which "low
> resolution" box it is located - allowing me to continue with histogram and
> its reverse indices to extract relevant averages from the MODIS data.
>
> I have some code which tells me if a point falls within a specific box,
> but that would require a loop over all boxes and all pixels until a
> matching box is found - with ~ 99000 boxes in an orbit, things get tedious
> and slow very fast. I /think/ I can adapt this code to do either loop in
> one go, but not both loops at the same time - at least I have no idea
> where (no pun intended) to start.
>

Here's a good reference:

http://astronomy.swin.edu.au/~pbourke/geometry/insidepoly/

I believe Mark Hadfield has coded up one or more of these in his
Motley library, which at least vectorizes over the points to test.
Since all your polygons are the same length (4), it is possible to
simultaneously vectorize over both points and polygons (this would not
be true for arbitrary-sized polygons).

It sounds like you could use Solution 3 from the above reference for
convex polygons, i.e. see if a given point is to the same side of all
line segments in your polygons, by testing the sign of the quantity:

 (y-y0)(x1-x0)-(x-x0)(y1-y0)

for each polygon line segment, and concluding it's inside if all the
signs agree. How would this look vectorized? Something like the
function below. Notice it allows any arbitrary shaped array of x,y
input points, and just appends a dimension for the number of polygons.

However, similar to the "points inside a sphere" problem I just
commented on, you'll run out of memory fast using such a brute force

method. The algorithm takes several arrays of size
npts*npoly*mpoly*4, where npts is the number of points to test, npoly
is the number of polygons to test against, and mpoly is the length of
each polygon. You could probably test about 200 points at a time in
1GB memory against your 99000 polygons without hitting the memory
limit (such a run takes about 15s on my system). Since a loop which
processes 1GB/200 worth of memory at a time will not really feel the
IDL loop penalty, you could just as well loop over those 200 points
without a big loss of speed, or "chunk" them into 100 at a time or so.
If you have multiple processors, it's more efficient to cast things
into fairly large arrays.

For a smallish number of points and polygons, this brute force method
will definitely be the fastest, thanks to the overhead of loops.
However, for larger sized problems, just like for the spherical search
case, you would do well to pre-reject points which definitely cannot
be inside a given polygon. A similar method could be used: use
HIST_2D on the points with a binsize similar to the typical size of
each polygon. For each polygon, pre-compute the bounding-box in x and
y, and from that compute which histogram bins lie within each bounding
box. Then, perform inside/outside tests only on those points, and
don't bother testing the rest. How much pre-selection to apply will
depend on how many points vs. polygons, whether either set is fixed
from calculation to calculation, etc.

JD

```
;+
; NAME:
;
;  POINT_IN_POLY_SET
;
; PURPOSE:
;
;  Determine whether a group of points are inside a set of 2D, convex
;  polygons of the same length.
;
; CALLING SEQUENCE:
;
;  res=point_in_poly_set(x,y,px,py)
;
; INPUTS:
;
;  x,y: The points to test, arrays of any format.  The result will be
;    an array of the same size, with a final trailing dimension of
;    length the number of polygons.
;
```

```
;  px,py: The polygons to test against, arranged as arrays of size
;    nxm, where n is the number of polygons, and m is their length
;    (which must be the same).
;
; OUTPUTS:
;
;  res: A tri-valued array the same size as the input x,y vectors,
;    with an additional trailing dimension with length n, the number
;    of polygons passed.  It indicates, for each point and each
;    polygon, whether the point was inside (1), outside (-1), or on
;    (0) the given polygon.
;
; RESTRICTIONS:
;
;  The polygons must be convex.
;
; PROCEDURE:
;
;  From solution 3 of:
;  http://astronomy.swin.edu.au/~pbourke/geometry/insidepoly/
;
; MODIFICATION HISTORY:
;
;
;   2005-11-30 (J.D. Smith): Writen
;-
 ;#####################################################################
####################
;
; LICENSE
;
;  Copyright (C) 2005 J.D. Smith
;
;  This file is free software; you can redistribute it and/or modify
;  it under the terms of the GNU General Public License as published
;  by the Free Software Foundation; either version 2, or (at your
;  option) any later version.
;
;  This file is distributed in the hope that it will be useful, but
;  WITHOUT ANY WARRANTY; without even the implied warranty of
;  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
;  General Public License for more details.
;
;  You should have received a copy of the GNU General Public License
;  along with this file; see the file COPYING.  If not, write to the
;  Free Software Foundation, Inc., 59 Temple Place - Suite 330,
;  Boston, MA 02111-1307, USA.
;
```

```
 ;#####################################################################
####################

function point_in_poly_set,x,y,px,py
  spol=size(px,/DIMENSIONS)
  if n_elements(spol) ne 2 then message,'Polygons must be nxm array'
  spts=size(x,/DIMENSIONS)
  m=spol[0]

  pxdiff=shift(px,1,0)-px & pydiff=shift(py,1,0)-py

  rs=[1,1,spts]              ;reform size
  ts=[spol,spts]             ;working target size

   in_test=(rebin(reform(y,rs),ts,/SAMPLE)-rebin(py,ts,/SAMPLE) )* $
         rebin(pxdiff,ts,/SAMPLE) - $
         (rebin(reform(x,rs),ts,/SAMPLE)-rebin(px,ts,/SAMPLE))* $
         rebin(pydiff,ts,/SAMPLE)

  ;; For each point and polygon, test for all negative or all positive
  ;; (inside polygon), mixed (outside) or any exactly 0.0 (on polygon)
  sign=1 - 2*(in_test lt 0.0) - (temporary(in_test) eq 0.0)

  ;; Inside/Outside polygon test: -m, or m: inside, otherwise: outside or on
  out=long(total(sign,1,/INTEGER))

  ;; On polygon test: on one or more polygon line segments, and to the
  ;; same side of all the others
  on=(product(sign,1,/INTEGER) eq 0LL) AND $
     (abs(out) eq long(total(temporary(sign) ne 0,1,/INTEGER)))
  out=temporary(out)/m eq 0L

  if size(out,/N_DIMENSIONS) le 1 then return,1-2*out+on
  return,transpose(1-2*out+on,[1+indgen(n_elements(spts)),0])
end
```