Subject: Re: Fast Implementation
Posted by David Fanning on Fri, 05 Jul 2002 16:22:56 GMT
View Forum Message <> Reply to Message

Isa Usman (I.Usman@rl.ac.uk) writes:

> I have the bit of code below which calculates the number of points in all
> four quandrants of a 2d space. Unfortunately my arrays are very large and it
> takes quite a while to run.Is there a way of making the code faster.

If you normalize your quadrant to 1, and take a bin size
of 0.5, then a Histogram would give you the answer in, oh,
0.00005 seconds. :-)

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Fast Implementation
Posted by Pavel A. Romashkin on Fri, 05 Jul 2002 17:06:02 GMT
View Forum Message <> Reply to Message

David, I envy you. Your busy time ended in 44 minutes (10:22 first post,
11:06 the second). Obviously, you need to help me with that foundation
now :-)
Cheers,
Pavel

David Fanning wrote:
>
> David Fanning (david@dfanning.com) writes:
>
>> If you normalize your quadrant to 1, and take a bin size
>> of 0.5, then a Histogram would give you the answer in, oh,
>> 0.00005 seconds. :-)
>
> Sorry, I was busy before. Here is an example of what I
> mean:
>
> ***
> PRO Example

```
>
> ; Find out how many points are in each quadrant
> ; of a 2D space given by the max and min of the
> ; data.
>
> x = (randomu(seed, 1000) - 0.5) * 100
> y = (randomu(seed, 1000) - 0.5) * 100
> Plot, x, y, /NoData
> Oplot, x, y, Psym=4
>
> xx = Scale_Vector(x, 0, 0.9999999)
> yy = Scale_Vector(y, 0, 0.9999999)
> result = Hist_2D(xx, yy, Bin1 = 0.5, Bin2=0.5, $
>    Min1=0, Max1=1.0, Min2=0, Max2=1.0)
> print, result[0:1, 0:1]
> END
> ************************************************
>
> You can find the Scale_Vector program on my web page:
>
>    http://www.dfanning.com/programs/scale_vector.pro
>
> Cheers,
>
> David
>
> --
> David W. Fanning, Ph.D.
> Fanning Software Consulting, Inc.
> Phone: 970-221-0438, E-mail: david@dfanning.com
> Coyote's Guide to IDL Programming: http://www.dfanning.com/
> Toll-Free IDL Book Orders: 1-888-461-0155
```

Subject: Re: Fast Implementation
Posted by David Fanning on Fri, 05 Jul 2002 17:06:42 GMT
View Forum Message <> Reply to Message

David Fanning (david@dfanning.com) writes:

> If you normalize your quadrant to 1, and take a bin size
> of 0.5, then a Histogram would give you the answer in, oh,
> 0.00005 seconds. :-)

Sorry, I was busy before. Here is an example of what I
mean:

```
************************************************
```

```
PRO Example

; Find out how many points are in each quadrant
; of a 2D space given by the max and min of the
; data.

x = (randomu(seed, 1000) - 0.5) * 100
y = (randomu(seed, 1000) - 0.5) * 100
Plot, x, y, /NoData
Oplot, x, y, Psym=4

xx = Scale_Vector(x, 0, 0.9999999)
yy = Scale_Vector(y, 0, 0.9999999)
result = Hist_2D(xx, yy, Bin1 = 0.5, Bin2=0.5, $
   Min1=0, Max1=1.0, Min2=0, Max2=1.0)
print, result[0:1, 0:1]
END
************************************************
```

You can find the Scale_Vector program on my web page:

   http://www.dfanning.com/programs/scale_vector.pro

Cheers,

David

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Fast Implementation
Posted by Dick Jackson on Fri, 05 Jul 2002 17:48:30 GMT

View Forum Message <> Reply to Message

"Isa Usman" <I.Usman@rl.ac.uk> wrote in message
news:ag4fjt$j6a@newton.cc.rl.ac.uk...
> Hi,
>
> I have the bit of code below which calculates the number of points in
all
> four quandrants of a 2d space. Unfortunately my arrays are very large
and it
> takes quite a while to run.Is there a way of making the code faster.

David's approach may be just what you're looking for, but if you need
the results to be identical to what you gave (using GT and LT will not
count points that are *equal* to the test value), the code below shows
the time taken and checks that the result is still correct! I get better
than 2x speedup over your original on arrays up to 5000 items.

I used two tricks here:
- you don't need the indexes returned by Where, so just use Total
- create partial results (binary arrays for X gt x0, etc.) and reuse
them

=====

```
PRO QuadrantCount

n1 = 1000
n2 = 1000
x = RandomU(seed, n1)
y = RandomU(seed, n1)

;;    Method 1

points = FltArr(n1, 4)

Print, 'Starting method 1...'
t0 = SysTime(1)

for j=0L,n1-1 do begin
 x0=X(j)
 y0=Y(j)

 index=where(X gt x0 and Y gt y0,count1)
 index=where(X lt x0 and Y gt y0,count2)
 index=where(X lt x0 and Y lt y0,count3)
 index=where(X gt x0 and Y lt y0,count4)

 na=count1
 nb=count2
 nc=count3
 nd=count4

 points(j,0:3)=float([na,nb,nc,nd])/n2

endfor

Print, SysTime(1)-t0, ' seconds'
```

```
;;    Method 2 - don't use Where, use Total

points2 = FltArr(n1, 4)

Print, 'Starting method 2...'
t0 = SysTime(1)

for j=0L,n1-1 do begin
 x0=X(j)
 y0=Y(j)

 na=Total(X gt x0 and Y gt y0)
 nb=Total(X lt x0 and Y gt y0)
 nc=Total(X lt x0 and Y lt y0)
 nd=Total(X gt x0 and Y lt y0)

 points2(j,0:3)=float([na,nb,nc,nd])/n2

endfor

Print, SysTime(1)-t0, ' seconds'
Print, Total(points2 NE points), ' errors'


;;    Method 3 - create partial results and reuse them

points3 = FltArr(n1, 4)

Print, 'Starting method 3...'
t0 = SysTime(1)

for j=0L,n1-1 do begin
 x0=X(j)
 y0=Y(j)

 xl = X lt x0
 xg = X gt x0
 yl = Y lt y0
 yg = Y gt y0

 na=Total(xg AND yg)
 nb=Total(xl AND yg)
 nc=Total(xl AND yl)
 nd=Total(xg AND yl)

 points3(j,0:3)=float([na,nb,nc,nd])/n2
```

endfor

Print, SysTime(1)-t0, ' seconds'
Print, Total(points3 NE points), ' errors'

END

=====

Hope this helps!

Cheers,
--
-Dick

Dick Jackson / dick@d-jackson.com
D-Jackson Software Consulting / http://www.d-jackson.com
Calgary, Alberta, Canada / +1-403-242-7398 / Fax: 241-7392

Subject: Re: Fast Implementation
Posted by Sven Geier on Tue, 09 Jul 2002 21:45:43 GMT
View Forum Message <> Reply to Message

Isa Usman wrote:

```
> Hi,
>
> I have the bit of code below which calculates the number of points in all
> four quandrants of a 2d space. Unfortunately my arrays are very large and
> it takes quite a while to run.Is there a way of making the code faster.
>
> Thanks in advance!
>
> Isa
> @@@@@@@@@@@@
> ;Data samples
> for j=0L,n1-1 do begin
>  x0=X(j)
>  y0=Y(j)
>
>  index=where(X gt x0 and Y gt y0,count1)
>  index=where(X lt x0 and Y gt y0,count2)
>  index=where(X lt x0 and Y lt y0,count3)
>  index=where(X gt x0 and Y lt y0,count4)
>
>  na=count1
>  nb=count2
>   nc=count3
>   nd=count4
>
>
>   points(j,0:3)=float([na,nb,nc,nd])/n2
>
>  endfor
```

Just to get that straight: You have n1 points, where n1 is large and for each point you want to know how many of the *other* points are above/below/left/right of it, right?

I am tacitly assuming here that you are using reals or doubles, i.e. there is rarely ever an event with the exact same X[] or Y[] as another (right? wrong?)

Step one: If you sort your arrays according to one of the coordinates, the answer for that coordinate will just be the index:

```
 Xs = sort(X)
 Xn = x[Xs]
 Yn = Y[Ys]
```

Then X[i] has (i-1) other points with X < X[i]

Step two: for each point you only need to know how many other points are greater in Y, since you know the total number of points (and whatever isn't greater would have to be less or equal). I.e. if

```
 na = where(Yn[0:i-1] lt Yn[i]) ; don't need to check the [i+1:*] elements
                                ; because the sorting put those in the
                                ; other X half-plane
```

Then

```
 nb = i - na
```

IFF you can tolerate a "le" for one half of your quadrants.

Subject: Re: Fast Implementation
Posted by Sven Geier on Tue, 09 Jul 2002 21:58:09 GMT
View Forum Message <> Reply to Message

[Humm - resending. My new newsreader seems to have small troubles...]

Isa Usman wrote:

> Hi,
>
> I have the bit of code below which calculates the number of points in all
> four quandrants of a 2d space. Unfortunately my arrays are very large and
> it takes quite a while to run.Is there a way of making the code faster.
>
> Thanks in advance!
>
> Isa

```
> @@@@@@@@@@@@@
> ;Data samples
> for j=0L,n1-1 do begin
>  x0=X(j)
>  y0=Y(j)
>
>  index=where(X gt x0 and Y gt y0,count1)
>  index=where(X lt x0 and Y gt y0,count2)
>  index=where(X lt x0 and Y lt y0,count3)
>  index=where(X gt x0 and Y lt y0,count4)
>
>  na=count1
>  nb=count2
>  nc=count3
>  nd=count4
>
>
>  points(j,0:3)=float([na,nb,nc,nd])/n2
>
> endfor
```

Just to get that straight: You have n1 points, where n1 is large and for
each point you want to know how many of the other points are
above/below/left/right of it, right?

I am tacitly assuming here that you are using reals or doubles, i.e. there
is rarely ever an event with the exact same X[] or Y[] as another (right?
wrong?)

Step one: If you sort your arrays according to one of the coordinates, the
answer for that coordinate will just be the index:

```
 Xs = sort(X)
 Xn = x[Xs]
 Yn = Y[Ys]
```

Then X[i] has (i-1) other points with X < X[i]

Step two: for each point you only need to know how many other points are greater in Y, since you know the total number of points (and whatever isn't greater would have to be less or equal). I.e. if

```
 na = where(Yn[0:i-1] lt Yn[i]) ; don't need to check the [i+1:*] elements
                              ; because the sorting put those in the
                              ; other X half-plane
```

Then

```
 nb = i - na
```

IFF you can tolerate a "le" for one half of your quadrants.