Subject: Re: Resampling data with irregular time base
Posted by Richard G. French on Sat, 05 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Karl Krieger wrote:

```
>
> I have data with an irregular time base, which I would like to resample
> in a regular spaced time base. How can I average over all original data
> points in each interval of the new time vector without resorting to a
> FOR loop?
> Currently I am using this horrible kludge:
>
> deltat = newtime[1] - newtime[0]
> FOR n=0, n_elements(newtime)-1 DO BEGIN
>   index = where((oldtime GT (newtime[n]-deltat/2.)) AND $
>                (oldtime LE (newtime[n]+deltat/2.)), $
>                count)
>   IF count GT 0 THEN newdata[n] = total(olddata[index]) / count
> ENDFOR
>
```

Another idea! Make sure that oldtime is a sorted array.
Then do a simple interpolation into the oldtime array to find the bin
location boundaries that correspond to the newtime locations. This gives
you the indices in the oldtime[] array that correspond to the points you
are after for each newtime[]
data point. You still have to do a loop, but you are saving a LOT of
time by not having to use WHERE over the full range of oldtime[] - you
already know the approximate range of data in oldtime[] that fall
withing the newtime[] bin, and you can interrogate this MUCH smaller set
of points inside your loop to see if there are actually any data points
within the deltat range of each newdata[] point.
Dick French
rfrench@mediaone.net

Subject: Re: Resampling data with irregular time base
Posted by Richard G. French on Sat, 05 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Karl Krieger wrote:

```
>
> I have data with an irregular time base, which I would like to resample
> in a regular spaced time base. How can I average over all original data
> points in each interval of the new time vector without resorting to a
> FOR loop?
> Currently I am using this horrible kludge:
>
```

```
>  deltat = newtime[1] - newtime[0]
>  FOR n=0, n_elements(newtime)-1 DO BEGIN
>    index = where((oldtime GT (newtime[n]-deltat/2.)) AND $
>                (oldtime LE (newtime[n]+deltat/2.)), $
>                count)
>    IF count GT 0 THEN newdata[n] = total(olddata[index]) / count
>  ENDFOR
>
```

It appears to me that you are not after an interpolated smooth function
here, since if you have a very small deltat, you will get lots of
newdata[] values that are zero in cases when no olddata[] fall within
the time range of newdata[]. If you have a densely populated set of
olddata and you are trying to average, this may not be a problem. There
may be a way to use the histogram routine to get the information you are
after. I have not checked into that, but others may have suggestions. It
seems to me that the HISTOGRAM routine can give you the elements that go
into each bin, and if so, then you can retrieve the points that
contribute to each time bin and do the normalization yourself.

I think the REVERSE_INDICES keyword may be what you want, but you still
may end up having to do a loop.
>
> Set this keyword to a named variable in which the list of reverse indices is returned. This list is returned as a longword vector whose number of elements is the sum of the number of elements in the histogram, N, and the number of array elements included in the histogram, plus one.
>
> The subscripts of the original array elements falling in the ith bin, 0 ï¿½ i < N, are given by the expression: R(R[i] : R(i+1)-1), where R is the reverse index list. If R[i] is equal to R[i+1], no elements are present in the ith bin.
>
> Example Make the histogram of array A:
>
> H = HISTOGRAM(A, REVERSE_INDICES = R)
>
> IF R(i) NE R(i+1) THEN A(R(R(I) : R(i+1)-1)) = 0
> ;Set all elements of A that are in the ith bin of H to 0.
>
> The above is usually more efficient than the following:
>
> bini = WHERE(A EQ i, count)
>
> IF count NE 0 THEN A(bini) = 0
>

If you don't mind interpolating between neigboring points, you might try
doing a linear interpolation onto a very fine time grid that is

oversampled by some
integer multiple of the deltat you want in the end - choose this
multiplier
to be approximately the value of the closest spacing of your data
points.
Then you can use REBIN to average the regularly interpolated result to
deltat.
I've used this approach quite often when I have a relatively smooth but
irregularly spaced set of points - if it is REALLY smooth, then you can
use cubic splines and bypass the interpolation step altogether. But my
hunch is that you have some data that may be quite variable over short
time scales and that where there are no data points, you really want a
zero in the newdata[] array, not an iterpolation. In this case, I would
do a histogram of the oldtime array at the newtime spacing, pick out the
zero elements in the histogram and set the corresponding elements of the
interpolated array to zero. I guess it all depends on the nature of your
data and the accuracy you are after.

Dick French
rfrench@mediaone.net

Subject: Re: Resampling data with irregular time base
Posted by steinhh on Sat, 05 Jun 1999 07:00:00 GMT

View Forum Message <> Reply to Message

In article <7jatua$sau$1@kiosk.rzg.mpg.de>
krieger@ipp.mpg.NOSPAM (Karl Krieger) writes:

```
>  I have data with an irregular time base, which I would like to resample
>  in a regular spaced time base. How can I average over all original data
>  points in each interval of the new time vector without resorting to a
>  FOR loop?
>  Currently I am using this horrible kludge:
>
>  deltat = newtime[1] - newtime[0]
>  FOR n=0, n_elements(newtime)-1 DO BEGIN
>    index = where((oldtime GT (newtime[n]-deltat/2.)) AND $
>      (oldtime LE (newtime[n]+deltat/2.)), $
>      count)
>    IF count GT 0 THEN newdata[n] = total(olddata[index]) / count
>  ENDFOR
>
>  Any idea how to transform this in vectorized IDL code? At the moment I
>  see no way apart from writing the function in C and calling it by
>  linkimage.
```

I really can't see any other way than to use a temporary weighting

array with size n_elements(newtime) x n_elements(oldtime),
containing the differences between all points in newtime and all
points in oldtime.

If your data sets are large, this would still be very inefficient,
and this problem is a very good example of why IDL will never be
quite optimal compared to fully compiled languages. I C or
Fortran, you'd never muck about with any temporary arrays at all
in this case, you'd calculate the whole thing directly.

I would strongly suggest going for a linkimage (or DLM) function,
something like

```
  newdata = irreg_raverage(oldtime,olddata,newtime,deltat/2.,deltat/2.)
```

(irregularly sampled running average) allowing you to specify both
(ordered, but possibly irregular!) new sampling times as well as the
forward/backward time windows to average over.

Regards,

Stein Vidar

Subject: Re: Resampling data with irregular time base
Posted by Martin Schultz on Mon, 07 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Karl Krieger wrote:

```
>
> I have data with an irregular time base, which I would like to resample
> in a regular spaced time base. How can I average over all original data
> points in each interval of the new time vector without resorting to a
> FOR loop?
> Currently I am using this horrible kludge:
>
> deltat = newtime[1] - newtime[0]
> FOR n=0, n_elements(newtime)-1 DO BEGIN
>   index = where((oldtime GT (newtime[n]-deltat/2.)) AND $
>               (oldtime LE (newtime[n]+deltat/2.)), $
>               count)
>   IF count GT 0 THEN newdata[n] = total(olddata[index]) / count
> ENDFOR
>
> Any idea how to transform this in vectorized IDL code? At the moment I
> see no way apart from writing the function in C and calling it by
> linkimage.
>
```

> Best
>
> Karl
>
> --
> To reply by email please replace domain .NOSPAM by .de in reply address

Hi Karl,

I would also do it with a for loop, but I wouldn't call WHERE. Rather
I would go for the old "FORTRANY" approach and loop over each element,
get the sum and count the number of time steps you pass, and compute the
average as soon as you reach the next regular time step. As an example
you could take a look at my attached run_av.pro which computes running
averages and can handle irregular series (just "regridding" is even
easier). One further big advantage of this method is that you can easily
exclude "missing" data on the fly. If you have large gaps in your data,
you could think of testing for the next regular time step after each
averaging step and then fill the gaps with one fltarr() command instead
of looping through 1000 void steps.

One hint: make sure that you use LONG integers in your FOR loops! I just
realized I didn't adhere to that in the attached version of run_av.pro
...

Regards,
Martin.

Never trust anyone less than yourself!

--

|||||||||||||||\\\\\\\\\\\\\-------------------///////////// //||||||||||||||
Martin Schultz, DEAS, Harvard University, 29 Oxford St., Pierce 109,
Cambridge, MA 02138   phone (617) 496 8318   fax (617) 495 4551
e-mail mgs@io.harvard.edu   web http://www-as/people/staff/mgs/

```
;------------------------------------------------------------- --
; $Id: run_av.pro,v 1.10 1999/01/22 20:12:17 mgs Stab $
;+
; NAME:
;       RUN_AV (function)
;
; PURPOSE:
;       Compute running average or running total of a
;       data vector. Compared to the IDL function TS_SMOOTH,
;       this function takes into account missing values or
;       gaps in an optional x vector, and it allows for
;       even bandwidths. It can also be used to compute cumulative
;       totals.
;
; CATEGORY:
;       math
;
; CALLING SEQUENCE:
;       result = RUN_AV(Y [,X] [,keywords] )
;
; INPUTS:
;       Y -> the data vector (a 2-D array will be treated as a vector)
;
;       X -> an optional X vector defining e.g. the sample times.
;           This only has an effect when the DELTAX keyword is specified.
;           X must be monotonically increasing and have the same
;           number of elements as Y.
;
; KEYWORD PARAMETERS:
;       WIDTH -> The number of points to use for the average or total
;           Default is 1, i.e. Y is returned unchanged.
;
;       MINWIDTH -> The minimum number of points that must be valid
;           in order to return a average or total for the given point.
;           Default is MINWIDTH=WIDTH, i.e. all points must be valid
;           (and if X and DELTAX are specified, all points must lie
;           within WIDTH*DELTAX).
;
;       MIN_VALID -> The minimum value for valid data. Data with less than
;           MIN_VALID will be considered missing. MIN_VALID is also used
;           to indicate invalid totals or averages (1% is subtracted).
;
;       DELTAX -> The maximum gap between two consecutive x values.
;           Only effective when X is given.
;
;       COUNT -> A named variable will return the number of points used
;           in each average or total.
;
;       /TOTAL -> Set this keyword to compute running totals instead
;           of running averages.
;
; OUTPUTS:
;       The function returns a vector with running averages or totals.
;       The number of elements in the result vector always equals the
;       number of elements in Y (unless an error occurs).
;
; SUBROUTINES:
;
; REQUIREMENTS:
```

```
;
; NOTES:
;       This function can also be used to compute accumulative totals.
;       Simply set WIDTH to n_elements(Y) and MINWIDTH to 1 and use
;       the /TOTAL keyword. However, this is very uneffective for large
;       data vectors!
;
; EXAMPLE:
;       y = findgen(20)
;       print,run_av(y,width=4)
;       ; IDL prints: -1E31 -1E31 -1E31  1.5  2.5  3.5  4.5 ...
;
;       print,run_av(y,width=4,/TOTAL)
;       ; IDL prints: -1E31 -1E31 -1E31  6  10  14  18 ...
;
;       ; (cumulative total)
;       print,run_av(y,width=n_elements(y),minwidth=1,/TOTAL)
;       ; IDL prints:  0  1  3  ...  190
;
;       x = [ 0, 2, 4, 6, 16, 20, 24, 25, 26, 27, 28, 29, 30, 32, 33 ]
;       y = fltarr(n_elements(x)) + 1.
;       print,run_av(y,x,width=4,count=c)
;       ; IDL prints: -1E31  -1E31  -1E31  1  1  1  1  ...
;       print,c
;       ; IDL prints:  1  2  3  4  4  4  4  4  4  4  4  4  4  4  4
;
;       print,run_av(y,x,deltax=2,width=4,count=c)
;       ; IDL prints: -1E31  -1E31  -1E31  1  -1E31  -1E31  -1E31
;       ;             -1E31  -1E31  -1E31  1   1   1   1   1
;       print,c
;       ; IDL prints:  1  2  3  4  3  2  1  1  2  3  4  4  4  4  4
;
; MODIFICATION HISTORY:
;       mgs, 21 Oct 1998: VERSION 1.00
;
;-
; Copyright (C) 1998, Martin Schultz, Harvard University
; This software is provided as is without any warranty
; whatsoever. It may be freely used, copied or distributed
; for non-commercial purposes. This copyright notice must be
; kept with any copy of this software. If this software shall
; be used commercially or sold as part of a larger package,
; please contact the author to arrange payment.
; Bugs and comments should be directed to mgs@io.harvard.edu
; with subject "IDL routine run_av"
 ;----------------------------------------------------------- --
```

```
function run_av,y,x,width=width,min_valid=min_valid,deltax=deltax, $
      minwidth=minwidth,count=rcount,total=ctotal


   result = 0.
   if (n_elements(y) eq 0) then return,result

   ;  ============================================================ ====
   ; set up result array and temporary storage
   ;  ============================================================ ====

   average = not keyword_set(ctotal)

   if (n_elements(width) eq 0) then width = 1  $
   else width = fix(abs(width[0]))

   if (n_elements(minwidth) eq 0) then minwidth = width $
   else minwidth = minwidth < width      ; no larger than width!

   if (width eq 0) then begin
      message,'WIDTH must be greater or equal 1!',/Cont
      return,result
   endif

   accu = fltarr(width)
   count = intarr(width)
   result = fltarr(n_elements(y))
   rcount = intarr(n_elements(y))
   ic = 0

   if (n_elements(min_valid) eq 0) then min_valid = -9.99E30

   ;  ============================================================ ====
   ; VERSION 1: no x array given
   ;  ============================================================ ====

   if (n_elements(x) eq 0) then begin
      ; loop through y vector and accumulate
      for i = 0,n_elements(y)-1 do begin

         if ( (i-ic) ge width ) then ic = ic + width

         ; add current y value to all buffer elements
         ; if greater min_valid
         ; and increment counter
         if (y[i] gt min_valid) then begin
            accu[*] = accu[*] + y[i]
            count[*] = count[*] + 1
         endif

         ; read out ith buffer value and reset ith buffer
         rcount[i] = count[i-ic]
         if (count[i-ic] ge minwidth) then begin
            result[i] = accu[i-ic]
            if (average) then result[i] = result[i]/rcount[i]
         endif else begin
            result[i] = min_valid
         endelse

         accu[i-ic] = 0.
         count[i-ic] = 0


      endfor

      return,result
   endif



   ;  ============================================================= ====
   ; VERSION 2: with x array
   ; same as above, but needs to take care of min x steps
   ;  ============================================================= ====

   if (n_elements(x) ne n_elements(y)) then begin
      message,'X and Y must have same number of elements!',/Cont
      return,0.
   endif

   if (n_elements(deltax) eq 0) then begin
       xdiff = x - shift(x,1)
       deltax = max(xdiff[1:*])
   endif

   ; loop through y vector and accumulate
   for i = 0,n_elements(y)-1 do begin

       if ( (i-ic) ge width ) then ic = ic + width

       ; add current y value to all buffer elements
       ; if greater min_valid
       ; and increment counter
       if (y[i] gt min_valid and x[i]-x[(i-1)>0] le deltax) then begin
         accu[*] = accu[*] + y[i]
         count[*] = count[*] + 1
       endif
```

```
        ; read out ith buffer value and reset ith buffer
        rcount[i] = count[i-ic]
        if (count[i-ic] ge minwidth) then begin
          result[i] = accu[i-ic]
          if (average) then result[i] = result[i]/rcount[i]
        endif else begin
          result[i] = min_valid
        endelse

        accu[i-ic] = 0.
        count[i-ic] = 0

    endfor

    return,result



end
```

File Attachments

1) run_av.pro, downloaded 113 times

Subject: Re: Resampling data with irregular time base
Posted by Struan Gray on Tue, 08 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Karl Krieger, krieger@ipp.mpg.NOSPAM writes:

> I have data with an irregular time base, which I
> would like to resample in a regular spaced time
> base. How can I average over all original data
> points in each interval of the new time vector
> without resorting to a FOR loop?

Use the HISTOGRAM function with the REVERSE_INDICES keyword on your
array of time values. You can use the MAX, MIN and BINSIZE keywords
to define the start, stop and interval times of the new timebase.
Then, for each of those intervals the array returned by
REVERSE_INDICES will tell you the elements of your original data which
lie in that interval, so it's easy to add them up. You can extract the
normalisation divisor from the number of elements pointed to by the
reverse indices array, or from the value of the relevant bin of the
histogram itself.

Struan