Subject: Euclidean distance
Posted by Imanol Echave on Thu, 21 May 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Hi:

I'd like to program the following routine: giving two set of vectors with n and m vectors respectively (a matrix of size pxn and another of size pxm), I want to obtain a matrix of size nxm where the element (i,j) is the euclidean distance from first set's vector i to second set's vector j. I know the solution using loops, but exists it a different way without loops?

Subject: Re: Euclidean distance
Posted by mirko_vukovic on Fri, 22 May 1998 07:00:00 GMT
View Forum Message <> Reply to Message

In article <3563F2D3.59A48553@sc.ehu.es>,
Imanol Echave <ccaeccai@sc.ehu.es> wrote:
>
> Hi:
>
> I'd like to program the following routine: giving two set of vectors with n and
> m vectors respectively (a matrix of size pxn and another of size pxm), I want to
> obtain a matrix of size nxm where the element (i,j) is the euclidean distance
> from first set's vector i to second set's vector j. I know the solution using
> loops, but exists it a different way without loops?
>
The thing to do is to express each component of each vector as a (nxm) matrix.
For the first set of vectors, the values would vary along the first index, and
for the second they would vary along the second index.
Then you use each matrix as a variable in the formula for the euclidian
distance (A concept I am no longer familiar with)

One example on how to do this is the following routine that I find invaluable.

```
;+ NAME:
; TWO_D_ARR
;
; PURPOSE:
; Creates two 2-D arrays AX, AY from the input two 1-D vectors.  Both
; arrays have dimensions N_ELEMENTS(VX) * N_ELEMENTS(VY).  AX contains
; VX in all of its rows, AY contains TRANSPOSE(VY) in all the columns.
; These two arrays can be used for surface or countour plotting of
```

```
; functions of two variables.  The variable values are given in VX and
VY,
; AX and AY are created and can be used to calculate the 2-D array of
; the function values.
;
; CATEGORY:
; Vector processing
;
; CALLING SEQUENCE:
; TWO_D_ARR, VX,VY,AX,AY
;
; INPUTS:
; VX, VY -- 1-D vectors
;
; OPTIONAL INPUT PARAMETERS:
; None
;
; OUTPUTS:
; AX, AY, array of dimensions N_ELEMENTS(VX) * N_ELEMENTS(VY)
;
; OPTIONAL OUTPUT PARAMETERS:
;       DIM_ARR -- vector containing the dimension of the output arrays
;  in common block ARRAY_STUFF
;
; COMMON BLOCKS:
;       ARRAY_STUFF -- contains DIM_ARR, see above
;
; SIDE EFFECTS:
; None
;
; RESTRICTIONS:
; Unknown
;
; PROCEDURE:
; Straight forward
;
; MODIFICATION HISTORY:
; Conceived, written and performed by Mirko, Sept. 89
;-

PRO TWO_D_ARR, vx, vy, AX, AY
common ARRAY_STUFF, dim_arr

tvy = transpose(vy)
vx_size=size(vx)
vy_size=size(vy)
nx = vx_size(vx_size(0)+2)
ny = vy_size(vy_size(0)+2)
```

```
dim_arr = [nx,ny]

; pick type of array of higher order
type=vx_size(vx_size(0)+1)>vy_size(vy_size(0)+1)
ax_size=[2,dim_arr,type,nx*ny]
ax = make_array(size=ax_size)
ay = make_array(size=ax_size)

for i = 0, ny-1 do ax(0,i) = vx
for i = 0, nx-1 do ay(i,0) = tvy

end
```

***

the following two were part of a set of multi-dimensional array filling routines that helped me explore equations by not worrying about what is a scalar and what is not. I basically converted everything that was a vector into a set of compatible matrices (that conversion program would check what variables were vectors, and of what type, and call appropriate routines.)

Here they go

****

```
;+
; NAME:
;       ONE_TO_2_D_R_R
; PURPOSE:
;       routine that transforms a row vector into a matrix where the
;       vector is stacked by rows (that's what the R in the routine
;       name stands for), i.e., the elements of the matrix vary with the
;       first index, they are same for any second index, as long as the
;       first index is the same.   Based on the routine TWO_D_ARRAY
;
; CATEGORY:
;       VARIABLE MANIPULATION
;
; CALLING SEQUENCE:
; PRO ONE_TO_2_D_R_R, arg [,DIM=DIM_ARR]
;
; INPUTS:
;       DIM_ARR -- vector containing two elements.  The first indicates
;       how many columns should the array have, the second should have
;       the value of the n_elements(arg).  It can be passed either
;       throgh the keyword DIM or through the ARRAY_STUFF common block
;
;       ARG -- The vector to be transformed into the array.  Its
;       contetns are destroyed
```

```
; KEYWORD PARAMETERS:
;       DIM -- May be used to pass the dimensions fo of the array
;
; OPTIONAL INPUT PARAMETERS:
;       None
;
; OUTPUTS:
;       ARG -- the Array
;
; OPTIONAL OUTPUT PARAMETERS:
;       None
;
; COMMON BLOCKS:
;       ARRAY_STUFF -- contains DIM_ARR, see above
;
; SIDE EFFECTS:
;       The contents of ARG are destroyed, and the array is put in its
;       place.
;
; RESTRICTIONS:
;       None
;
; PROCEDURE:
;       The transpose of ARG is placed into the array
;
; MODIFICATION HISTORY:
;       Conceived, written and performed by Mirko, June 1990
;
;       Modified Dec.4.1991, by Mirko so that matrix dimensions can be
;       passed either by keyword or through the common block
;-
PRO ONE_TO_2_D_R_R,  arg,dim=dim
common ARRAY_STUFF, dim_arr1

if keyword_set(dim) then dim_arr=dim else dim_arr=dim_arr1

nx = dim_arr(0)
ny = dim_arr(1)

type=size(arg)
ay = make_array(size=[2,nx,ny,type(2),nx*ny])
for i = 0, ny-1 do ay(0,i) = arg

arg = ay

end
```

*** and the second one

```
;+
; NAME:
;       ONE_TO_2_D_R_C
; PURPOSE:
;       routine that transforms a row vector into a matrix where the
;       vector is stacked by columns (that's what the C in the routine
;       name stands for), i.e., the elements of the matrix vary with the
;       second index, they are same for any first index, as long as the
;       second index is the same.   Based on the routine TWO_D_ARRAY
;
; CATEGORY:
;       VARIABLE MANIPULATION
;
; CALLING SEQUENCE:
; PRO ONE_TO_2_D_R_C, arg
;
; INPUTS:
;       DIM_ARR -- vector containing two elements.  The first indicates
;       how many columns should the array have, the second should have
;       the value of the n_elements(arg).  It is located in the
;    ARRAY_STUFF common block
;       ARG -- The vector to be transformed into the array.  Its
;       contetns are destroyed
; OPTIONAL INPUT PARAMETERS:
;       None
;
; OUTPUTS:
;       ARG -- the Array
;
; OPTIONAL OUTPUT PARAMETERS:
;       None
;
; COMMON BLOCKS:
;       ARRAY_STUFF -- contains DIM_ARR, see above
;
; SIDE EFFECTS:
;       The contents of ARG are destroyed, and the array is put in its
;       place.
;
; RESTRICTIONS:
;       None
;
; PROCEDURE:
;       The transpose of ARG is placed into the array
;
; MODIFICATION HISTORY:
;       Conceived, written and performed by Mirko, June 1990
;-
```

```
PRO ONE_TO_2_D_R_C,  arg,dim=dim
common ARRAY_STUFF, dim_arr1

if keyword_set(dim) then dim_arr=dim else dim_arr=dim_arr1

nx = dim_arr(0)
ny = dim_arr(1)
tvy = transpose(arg)

type=(size(arg))(2)
ay = make_array(size=[2,nx,ny,type,nx*ny])
for i = 0, nx-1 do ay(i,0) = tvy

arg = ay

end
```

***

cheers and good luck,

mirko