Subject: Re: Combinatorial
Posted by Paul Van Delst[1] on Fri, 27 Aug 2004 14:11:01 GMT
View Forum Message <> Reply to Message

Adilson wrote:
> Dear all,
> Would like to know as I make to effect combinations in the IDL I have
> a problem where I want to execute a fixed combination of elements
> contained in a vector. EX: A=[0,1,2,3,4,5] --> six elements I want
> to make combinations 3x3 of the elements contained in. The formed
> vector is of 6!/3!*(6-3)! = 20 elements. Example of the vector to be
> formed->[0,1,2],[0,1,3]... [3,4,5]. In the total of 20 combinations.
> Which the best form to execute this operation? If you to be able to
> help me would be grateful.
> I subscribe myself with the highest consideration.
> Thanks in advance for your help.

This is how I do it in Fortran. If IDL doesn't already have something like this built in,
the kludge below should be relatively easy to implement. I think. Does IDL allow recursion?

```
  RECURSIVE SUBROUTINE predictor_combination( initialize,      &  ! In/Output
                                              n_in_set,        &  ! Input
                                              n_in_subset,     &  ! Input
                                              i,               &  ! In/Output
                                              predictor_index, &  ! Output
                                              done             )  ! Output
    ! -- Input
    LOGICAL,                    INTENT( IN OUT ) :: initialize
    INTEGER,                    INTENT( IN )     :: n_in_set     ! == n
    INTEGER,                    INTENT( IN )     :: n_in_subset  ! == k
    INTEGER,                    INTENT( IN OUT ) :: i
    ! -- Output
    INTEGER, DIMENSION( n_in_subset ), INTENT( OUT )    :: predictor_index
    LOGICAL,                    INTENT( OUT )    :: done
    ! -- Local variables
    INTEGER :: j
    INTEGER :: max_index
    INTEGER :: i_minus_1

    IF ( initialize ) THEN
      initialize = .FALSE.
      done = .FALSE.
      i = n_in_subset
      predictor_index(:) = (/ ( j, j = 1, n_in_subset ) /)
      RETURN
    END IF

    max_index = n_in_set - n_in_subset + i
```

```
    IF ( predictor_index( i ) < max_index ) THEN
      predictor_index( i ) = predictor_index( i ) + 1
      IF ( i == 1 .AND. predictor_index( i ) == max_index ) THEN
        done = .TRUE.
      END IF
    ELSE
      i_minus_1 = i - 1
      CALL predictor_combination( initialize,      &  ! Input
                                  n_in_set,        &  ! Input
                                  n_in_subset,     &  ! Input
                                  i_minus_1,       &  ! Input
                                  predictor_index, &  ! Output
                                  done             )  ! Output
      predictor_index( i ) = predictor_index( i - 1 ) + 1
    END IF
  END SUBROUTINE predictor_combination
```

And the documentation:

```
! CALLING SEQUENCE:
!       CALL predictor_combination( initialize,      &  ! In/Output
!                                   n, k,            &  ! Input
!                                   i,               &  ! In/Output
!                                   predictor_index, &  ! Output
!                                   done             )  ! Output
!
! INPUT ARGUMENTS:
!       initialize:       A logical variable used to initialize the predictor
!                         sequence generation. Must be set to .TRUE. on first
!                         call. It is set to .FALSE. after first call. If at
!                         any stage this variable is reset to .TRUE. for
!                         subsequent calls, the predictor sequence generation
!                         starts from scratch.
!                         UNITS:      N/A
!                         TYPE:       LOGICAL
!                         DIMENSION:  Scalar
!                         ATTRIBUTES: INTENT( IN OUT )
!
!       n:                The size of the predictor set. This is the total
!                         number of predictors available for selection.
!                         UNITS:      N/A
!                         TYPE:       INTEGER
!                         DIMENSION:  Scalar
!                         ATTRIBUTES: INTENT( IN )
!
!       k:                The size of the predictor subset. This is the
```

```
!                   number of predictors used in the absorber coefficient
!                   regression. The total number of predictor combinations
!                   generated by this routine is the binomial coefficient
!                   n(C)k. See PROCEDURE below.
!                   UNITS:      N/A
!                   TYPE:       INTEGER
!                   DIMENSION:  Scalar
!                   ATTRIBUTES: INTENT( IN )
!
!      i:           The predictor index subset element to modify. This
!                   value is initialized on the first call to this
!                   routine so the calling program does not have to
!                   (indeed it SHOULDN'T) define or modify this value.
!                   UNITS:      N/A
!                   TYPE:       INTEGER
!                   DIMENSION:  Scalar
!                   ATTRIBUTES: INTENT( IN OUT )
!
! OPTIONAL INPUT ARGUMENTS:
!      None.
!
! OUTPUT ARGUMENTS:
!      predictor_index:  The index list for a predictor combination sequence.
!                   This array is updated each call with the next
!                   possible combination of predictor indices.
!                   UNITS:      N/A
!                   TYPE:       INTEGER
!                   DIMENSION:  Rank-1, DIMENSION( k )
!                   ATTRIBUTES: INTENT( OUT )
!
!      done:           A logical variable used to indicate to the CALLING
!                   routine that all of the possible combinatorial
!                   sequences have been generated.  This value is
!                   initialized to .FALSE. on the first call to this
!                   routine and set to .TRUE. when all predictor index
!                   sequences have been exhausted.
!                   UNITS:      N/A
!                   TYPE:       LOGICAL
!                   DIMENSION:  Scalar
!                   ATTRIBUTES: INTENT( OUT )
```

A simple test program below shows how it is called in practice. Each trip through the DO loop spits out the next combination based on on the "n" and "k" inputs.

```
PROGRAM predictor_combination_test
  ! $Id: predictor_combination_test.f90,v 1.1 2001/11/30 18:07:27 paulv Exp $
```

```
  ! -- Use the required module
  USE coefficient_generation

  ! -- Type declarations
  LOGICAL :: initialise, done
  INTEGER :: n, k, i, j
  INTEGER, DIMENSION(:), ALLOCATABLE :: p

  ! -- Get some n, k values
  WRITE( *, '( /5x, "Enter values for n and k : " )', &
           ADVANCE = 'NO' )
  READ( *, * ) n, k

  ! -- Allocate the predictor index array
  ALLOCATE( p(k) )

  ! -- Initialisation
  j = 0
  initialise = .TRUE.

  ! -- Output the combinations
  DO
    CALL predictor_combination( initialise, n, k, i, p, done )
    PRINT *, p
    j = j + 1
    IF (done) EXIT
  END DO
  WRITE( *, '( 5x, "Total combinations = ", i5, / )' ) j

  ! -- Deallocate the predictor index array
  DEALLOCATE( p )

END PROGRAM predictor_combination_test
```