Pergamon

Computers & Geosciences Vol. 22, No. 10, pp. 1175–1186, 1996
Copyright © 1996 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0098-3004/96 $15.00 + 0.00

PII: S0098-3004(96)00026-X

FAST VARIOGRAM COMPUTATION WITH FFT

DENIS MARCOTTE

Department Génie Minéral, Ecole Polytechnique, C.P. 6079, Succ. Centre-ville, Montréal, Canada H3C 3A7 (*e-mail*: marcotte@geo.polymtl.ca)

(Received 11 August 1995; accepted 20 January 1996)

Abstract—Two programs are presented to compute direct- and cross-variograms, direct and cross-covariograms, and pseudo-cross-variograms. The programs are written in MATLAB and are based on the Fast Fourier Transform algorithm (FFT). The programs accept complete, or incomplete, regular grid data. The FFT appoach is shown to be faster than the spatial approach for this type of data. It gives exactly the same numerical variogram values as programs operating in the spatial domain. These programs could be most useful in image analysis, where images are usually 256×256 pixels, 512×512 pixels, or larger. For such large images, FFT is many orders of magnitude faster than the spatial approach. Copyright © 1996 Elsevier Science Ltd

Key Words: Variogram, Cross-variogram, Covariogram, Fast Fourier Transform.

INTRODUCTION

It is straightforward to write a program for computing variograms: it is more difficult to make it efficient in computing time. In mining or environmental studies, this is usually not a problem as the number of data values rarely exceeds a few thousand, most often being a few hundred.

An application where computing time could be a deciding factor is remote sensing analysis. Typically images are 256×256 pixels, 512×512 pixels, or larger. This large number of data values prohibits computing variograms using a standard program for general usage. Even with small images or sub-images, the time factor could be important if many images are to be analyzed. It is, therefore, important to obtain ways to accelerate the computation to make the variogram a useful tool for this type of application. A first approach is to custom design the variogram program to gridded data. A second, and faster approach, utilizes the Fast Fourier Transform (FFT) algorithm.

This paper presents how the variogram computation can be accelerated by working in the frequency domain using the FFT algorithm. A comparison is made for these computations with those of the conventional spatial domain software. Advantages of the FFT approach are emphasized. Two programs written in the language MATLAB are presented. The programs enable computing variograms, cross-variograms, covariograms, cross-covariograms and pseudo-cross-variograms to be computed for a 2D regular grid. The grid need not be complete. It is shown that the gain in computing time can reach many orders of magnitude, depending on the data. On the other hand, FFT requires more available RAM.

METHODOLOGY

Two spatial domain approaches

The most classical approach for variogram computation is to consider, in turn, every possible pair of data values. For each pair, the distance, the orientation, and the squared difference between the data values are computed. When the data are on a regular grid, a better method is to consider only a few directions and lag steps. The computation can be performed at a lower cost by simply shifting the data matrix. The pairs are formed by the overlapping points of the original and the shifted matrix (Fig. 1). This approach has, assuming a regular data array of size $N \times M$, a computing complexity of:

$$\sum_k \sum_l (N-k) \bullet (M-l) \tag{1}$$

where k and l represent the different shifts in the x and y directions. The domain of definition of k and l depends on the number of lags and the directions to be computed. Of course, if all directions and lag steps are required or if an omnidirectional variogram is wanted, all the possible pairs have to be considered and the complexity is then: $(N \bullet M) \bullet (N \bullet M - 1)/2$, that is, the same complexity as for a general-purpose variogram program. However, typically four to eight different directions and lag spacings up to $N/2$ and $M/2$ are sufficient for variogram modelling. In that situation, the gain of the regular-grid approach is significant (Table 1).

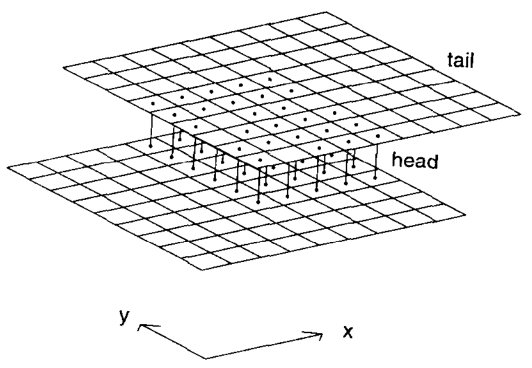

Figure 1. Original data matrix and shifted matrix necessary to compute variogram at lag spacing $Dx = 4$, $Dy = 3$. Displacement vector joins tail variable to head variable.

The spectral domain approach

Consider two random functions $f(x,y)$ and $g(x,y)$. In the continuous instance, the noncentered cross-covariance between $f(x,y)$ and $g(x,y)$ at a distance h_x and h_y is given by:

$$C(h_x, h_y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)g(x + h_x, y + h_y)dxdy. \tag{2}$$

Taking the Fourier transform, we obtain (Sneddon, 1951):

$$TF[C] = F^*(u, v)G(u, v) \tag{3}$$

where $F^*(u,v)$ is the complex conjugate of $F(u,v)$, $F(u,v)$ is the Fourier transform of $f(x,y)$ and $G(u,v)$ is the Fourier transform of $g(x,y)$.

The noncentered covariance can thus be obtained by computing the Fourier transform of $f(x,y)$ and $g(x,y)$, multiplying both and then taking the inverse Fourier transform. These results transpose directly in the discrete instance, provided that $f(x,y)$ and $g(x,y)$ are enlarged with zeros up to arrays of size $2N - 1$ by $2M - 1$. The enlargement must be performed to avoid the wraparound because of the assumed periodic nature of "f" and "g" (Fig. 2). Enlarging with zeros enables the FFT program to produce exactly the same numerical results as those obtained with spatial-domain programs.

FFT is an efficient algorithm to compute the discrete Fourier transform. Starting from a rectangular array of size $N \times M$, the array is enlarged with zeros to $(2N - 1) \times (2M - 1)$. Assuming $M < N$, the computational complexity is then:

$$(2M - 1)(2N - 1)\log_2(2N - 1). \tag{4}$$

Table 1 presents values obtained for a square array of size $N \times N$ using the standard approach, custom-grid spatial approach and FFT. For Equation (1), eight directions are considered and the maximum lag spacing is $N/2$. From Table 1, FFT is faster than the two spatial methods for a moderate N. For example, when N is 128, FFT is eight times faster than the (eight directions) spatial approach. For a large image (512 × 512), FFT can be 30–3000 times faster than the spatial approach depending on the number of directions and lags required.

Computing variograms with FFT

To compute the cross-variogram of $f(x,y)$ with $g(x,y)$, two indicator matrices, say I_f and I_g, are defined which are the same size as the extended data matrices f and g, and have ones at each data point and zeros elsewhere. Note that the data matrices need not be full and that missing values are replaced by zeros. Then, the following equation computes the cross-variogram:

$$\begin{aligned}\gamma_{fg} = (TF^{-1}\{TF[fg]^* TF[I_f I_g] + TF[I_f I_g]^* TF[fg] \\ -TF[fI_g]^* TF[gI_f] - TF[gI_f]^* TF[fI_g]\})/ \\ 2TF^{-1}\{TF[I_f I_g]^* TF[I_f I_g]\}\end{aligned} \tag{5}$$

where the division operator represents a division performed element by element for the matrices on the numerator and denominator (i.e $A/B = C$ with $[c_{ij}] = [a_{ij}/b_{ij}]$) and * is for the complex conjugate.

To compute the variogram of $f(x,y)$, one needs simply to replace "g" by "f" and I_g by I_f in the above formula.

Table 1. Complexity of operations

N ($M = N$)	Spatial approach all pairs $(N \bullet M)\bullet(N \bullet M - 1)/2$	Spatial approach eight directions, maximal spacing of $N/2$ lags$\sum_k\sum_l(N - k)\bullet(M - l)$	FFT approach $(2M - 1)(2N - 1)\log_2(2N - 1)$
2	6	6	14
4	120	90	138
8	2016	868	879
16	32,640	7560	4761
32	523,776	62,992	23,724
64	8,386,560	514,080	112,720
128	134,209,536	4,153,408	519,833
256	2,147,450,880	33,390,720	2,349,353
512	34,359,607,296	267,780,352	10,463,815
1024	549,755,289,600	2,144,862,720	46,089,347

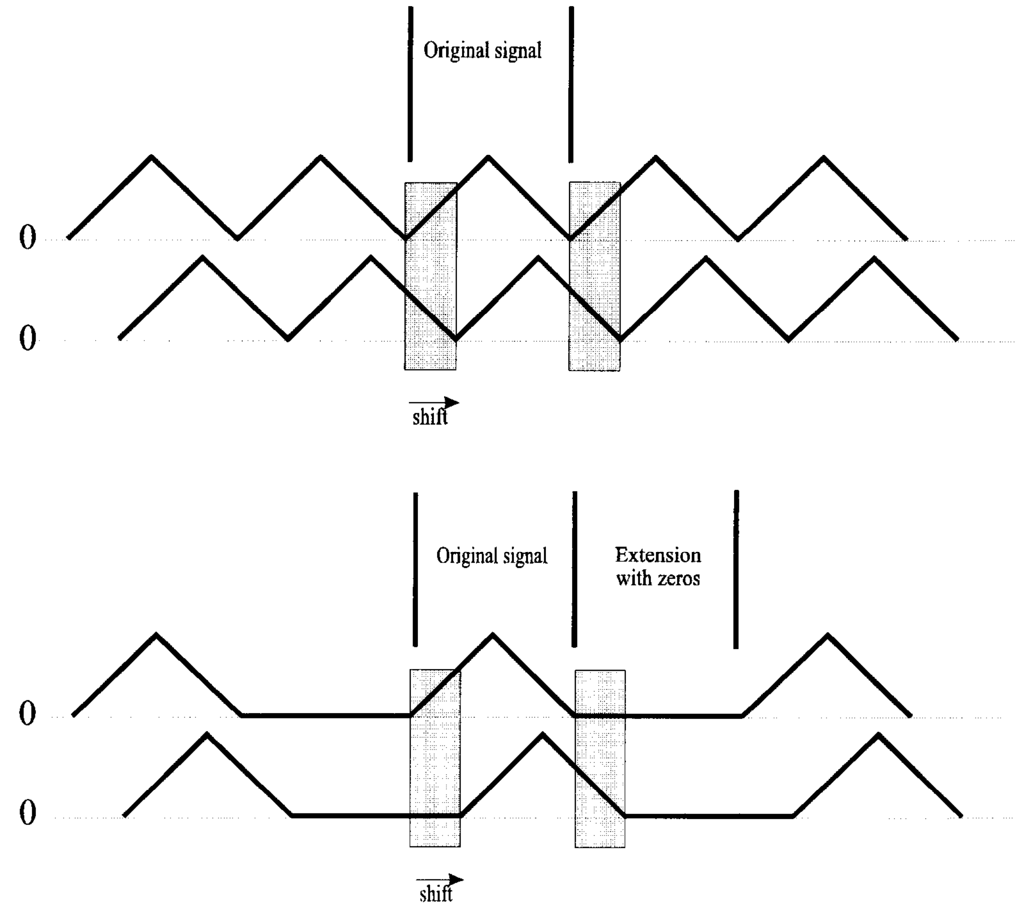

Figure 2. Illustration (in 1-D) of wraparound as result of assumed periodic nature of f(x) and g(x) in Fourier analysis. Enlarging signal with zeros eliminates contributions from outside spatial window observed.

Other structural functions such as the covariograms and cross-covariograms (Isaaks and Srivastava, 1989, p.59) and the pseudo-cross-variograms (Clark, Basinger, and Harper, 1989; Myers, 1991) can be computed also by FFT. The formula for the cross-covariogram is:

$$C_{fg} = \frac{TF^{-1}\{TF[f]^{*}\,TF[g]\}}{TF^{-1}\{TF[I_f]^{*}\,TF[I_g]\}} - \frac{TF^{-1}\{TF[f]^{*}\,TF[I_g]\}\,TF^{-1}\{TF[I_f]^{*}\,TF[g]\}}{\{TF^{-1}\{TF[I_f]^{*}\,TF[I_g]\}\}^{2}} \quad (6)$$

where the divisions and multiplications between matrices are made, as before, element by element. The displacement vector "h" is measured positively from "f" to "g" in C_{fg}. For example, the value at cell (3,2) (cell (0,0) is located at the center of C_{fg}) represents the covariances with "f" measured at (x,y) and "g" at (x + 3 lags, y + 2 lags). The value at cell $(-3,-2)$ is the covariance for "f" at (x,y) and "g" at ($x-$ 3 lags, $y-$ 2 lags). Of course, C_{fg} is not symmetric in "h" as opposed to γ_{fg}. In the above equation, the conjugate variable is the tail of the displacement vector.

The pseudo-cross-variogram is computed by:

$$p_{fg} = \frac{TF^{-1}\{TF[f^{2}]^{*}\,TF[I_g] + TF[I_f]^{*}\,TF[g^{2}] - 2TF[f]^{*}\,TF[g]\}}{2TF^{-1}\{TF[I_f]^{*}\,TF[I_g]\}} \quad (7)$$

with all arithmetic operations made element by element. The same convention used for the cross-covariograms is used for the displacement vector. Note also that the number of pairs for C_{fg} and p_{fg} [the denominators in Eqs (6) and (7)] is not symmetric as a function of "h".

MATLAB PROGRAMS FOR VARIOGRAM COMPUTATIONS

MATLAB is a software package combined with a command interpreter that allows fast program development. It has many attributes of a true programming language. Programs should be run within the MATLAB environment. However it is possible to use MATLAB functions from C (or FORTRAN) programs or call existing programs from MATLAB. (It is distributed by The Mathworks Inc. and is available at an educational cost of approximately US$400).

Two programs are presented in Appendices 1 and 2. The program variof1.m computes the variogram and covariogram for one variable. The program variof2.m performs the above computations for up to three variables (and can be modified easily to accomodate more variables). For both programs, the variogram (or other structural functions) and the number of pairs are computed for each lag separation. Results are presented in a matrix of size $2N-1 \times 2M-1$. The central point of this matrix corresponds to the lag (0,0). From the center, variogram values are positioned in the matrix according to the corresponding lag spacings measured positively from the tail variable to the head variable. Thus, the output directly gives the variogram map (Isaaks and Srivastava, 1989, p. 151). For variof2.m, the tail variable is the first one appearing in the input stream.

Of course, the output matrices can be processed further within MATLAB either for display of a variogram map (using the MATLAB "image" function, see Fig. 3) or for further processing such as the averaging of cells or the extraction of directional variograms for modelling.

A FEW EXAMPLES

The first example concerns two small data matrices (size 3×3) with missing values. It is used to check the different computations involved.

```
Let                        and
m1 be                      m2 be:
   3    6    5             10    NaN  5
   7    2    2             NaN   8    7
   4    NaN  0             5     9    11
```

where NaN is a code to indicate a missing value.

Calling variof2.m with:

$$[gh11, nh11, gh12, nh12, gh22, nh22] = variof2(m1, m2, [\,], 1)$$

results in:

```
gh11                                                nh11
     4.500  18.000   6.500   2.000   0.500            1    1    2    1    1
    12.500   3.500   5.400   2.333   2.000            2    3    5    3    2
     7.500   4.375   0.000   4.375   7.500            3    4    8    4    3
     2.000   2.333   5.400   3.500  12.500            2    3    5    3    2
     0.500   2.000   6.500  18.000   4.500            1    1    2    1    1
gh12                                                nh12
    -1.500   0.000  -8.750   0.000   0.000            1    0    2    0    1
     1.500  -1.000  -3.500  -3.750  -2.000            1    2    2    2    1
    -8.500   0.000   0.000   0.000  -8.500            2    1    6    1    2
    -2.000  -3.750  -3.500  -1.000   1.500            1    2    2    2    1
     0.000   0.000  -8.750   0.000  -1.500            1    0    2    0    1
gh22                                                nh22
     0.500   0.500  15.250   8.000   0.000            1    1    2    1    1
     4.500   3.250   3.500   3.667   2.000            1    2    3    3    1
    15.250   3.500   0.000   3.500  15.250            2    3    7    3    2
     2.000   3.667   3.500   3.250   4.500            1    3    3    2    1
     0.000   8.000  15.250   0.500   0.500            1    1    2    1    1
```

Note that zero is returned when no pairs are obtained at a given lag distance. "NaN" would also have been a sensible choice.

Changing the option to 2 (direct- and cross-covariograms)

$$[gh11, nh11, gh12, nh12, gh22, nh22] = variof2(m1, m2, [\ ], 2)$$

results in

```
gh11                                                  nh11
        0.000   0.000  -2.000   0.000   0.000            1      1      2      1      1
       -2.000   1.111   0.400   1.222   2.250            2      3      5      3      2
       -1.222  -0.375   4.734  -0.375  -1.222            3      4      8      4      3
        2.250   1.222   0.400   1.111  -2.000            2      3      5      3      2
        0.000   0.000  -2.000   0.000   0.000            1      1      2      1      1
gh12                                                  nh12
        0.000   0.000   5.000   0.000   0.000            1      0      2      0      1
        0.000   1.000   4.667   1.500  -1.500            1      2      3      2      2
        6.250   1.333  -2.944  -0.438   0.222            2      3      6      4      3
        0.000   0.667  -3.600  -1.625   4.000            1      3      5      4      2
        0.000  -1.000   2.444   1.500   0.000            1      2      3      2      1
gh22                                                  nh22
        0.000   0.000  -7.500   0.000   0.000            1      1      2      1      1
        0.000  -1.500   1.333  -1.222   0.000            1      2      3      3      1
       -7.500   0.667   4.694   0.667  -7.500            2      3      7      3      2
        0.000  -1.222   1.333  -1.500   0.000            1      3      3      2      1
        0.000   0.000  -7.500   0.000   0.000            1      1      2      1      1
```

And finally, with 3 as the option,

$$[gh11, nh11, gh12, nh12, gh22, nh22] = variof2(m1, m2, [\ ], 3)$$

gives the direct-variograms and the pseudo-cross-variogram:

```
gh11                                                  nh11
        4.500  18.000   6.500   2.000   0.500            1      1      2      1      1
       12.500   3.500   5.400   2.333   2.000            2      3      5      3      2
        7.500   4.375   0.000   4.375   7.500            3      4      8      4      3
        2.000   2.333   5.400   3.500  12.500            2      3      5      3      2
        0.500   2.000   6.500  18.000   4.500            1      1      2      1      1
gh12                                                  nh12
       50.000   0.000  15.250   0.000   0.500            1      0      2      0      1
       32.000  32.000  11.167   6.250   3.250            1      2      3      2      2
       12.500  22.167  19.333   6.500   8.833            2      3      6      4      3
        4.500  11.167  14.200  13.875   8.000            1      3      5      4      2
        0.000   4.250   8.167  15.250  32.000            1      2      3      2      1
gh22                                                  nh22
        0.500   0.500  15.250   8.000   0.000            1      1      2      1      1
        4.500   3.250   3.500   3.667   2.000            1      2      3      3      1
       15.250   3.500   0.000   3.500  15.250            2      3      7      3      2
        2.000   3.667   3.500   3.250   4.500            1      3      3      2      1
        0.000   8.000  15.250   0.500   0.500            1      1      2      1      1
```

The second example shows the gain in computer time resulting from the use of the FFT variogram computation. Different images of various sizes are used in the variogram computation. Computing times obtained with variof1 and gam2 from the library GSLIB (Deutsch and Journel, 1992) are shown in Table 2. Runs were performed on a Pentium 100 computer (with 32 MB RAM). Four directions and all possible lags were specified for gam2. Note that the program gam2 is a compiled FORTRAN program whereas variof1 is interpreted within MATLAB (under WINDOWS). The differences shown in Table 2 are expected to be greater for a compiled version of variof1 written in FORTRAN or C.

As stated already, the proposed FFT approach gives the same numerical result as the spatial approach. To check this, data of the preceeding

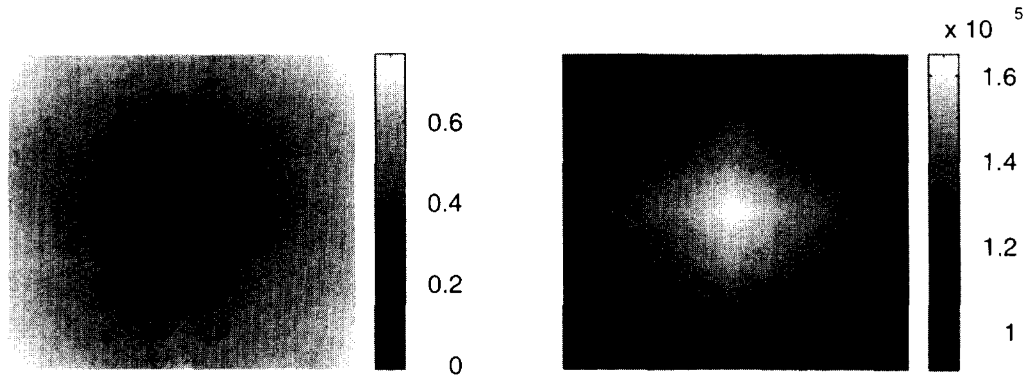

Figure 3. Variogram map (left) and corresponding image for number of pairs (right) for lags up to 32 units. Data matrix used in computation was an image with 128 x 128 pixels.

Table 2. Computing times for square matrices ($N \times N$) on Pentium – 100

N	Spatial approach four directions, all lags gam2 time (s)	FFT approach all directions all lags variofl time (s)
8	0.2	<0.1
16	0.8	0.1
32	4.2	0.3
64	6.7	1.6
128	49.8	6.9
200	178.5	26.4
256	683.5	58.2*

*Swapping to hard disk occurred. Without swapping execution time would be approximately 30 s.

256×256 matrix was obtained from a lognormal distribution with a coefficient of variation of 2. The numerical relative difference between the FFT approach and the spatial approach is measured by $\max_h\left[|\gamma_{FFT}(h) - \gamma_{\text{Spatial}}(h)|/\gamma_{\text{Spatial}}(h)\right]$ considering all lags h in the four main directions. The maximum value obtained was 1.7×10^{-12}, a value close to 0.

The main drawback of the frequency approach is the memory space required which is greater than with the spatial approach. On the computer used, 16 MB RAM were available, and it was possible to compute variograms for data matrices of sizes up to 150×150 before MATLAB started to swap to the hard-disk. A compiled version could enable larger data matrices to be run using only RAM.

CONCLUSION

The time required to compute variograms or similar structural functions is important for large images. Using FFT dramatically reduces the computing time. Even for small- to medium-sized images, the reduction in computing time warrants using the frequency approach, especially when repetitive variogram computations are required as is the situation for textural analysis of images based on variogram parameters (St-Onge, 1994). The code is available by anonymous FTP from the server at IAMG.ORG.

Acknowledgments—This work was partly financed by the National Science and Engineering Research Council of Canada.

REFERENCES

Clark, I., Basinger, K., and Harper, W., 1989, MUCK—a novel approach to co-kriging, *in* Buxton, B. E., ed., Proc. Conference on Geostatistical, Sensitivity and Uncertainty: Methods for Ground-Water Flow and Radionuclide Transport Modeling: Batelle Press, Columbus, p. 473–494.

Deutsch, C. V., and Journel, A. G., 1992, GSLIB: geostatistical software library and user's guide: Oxford Univ. Press, New York, 340 p.

Isaaks, E., and Srivastava, R., 1989, An introduction to applied geostatistics: Oxford Univ. Press, New York, 561 p.

Myers, D. E., 1991, Pseudo-cross-variograms, positive-definiteness and cokriging: Math. Geology, v. 23, no. 6, p. 806–816.

Sneddon, I. N., 1951, Fourier transforms: McGraw-Hill, New York, 542 p.

St-Onge, B., 1994, L'apport de la texture des images numériques de haute résolution à la cartographie forestière automatisée: unpubl. Ph.D dissertation, Univ. Montreal, Montreal, Canada, 251 p.

APPENDIX 1

Listing of program Variof1.m

```
function [gh11,nh11]=variof1(x1,icode);
%
% function [gh11,nh11]=variof1(x1,icode);
%
% function to compute variograms or covariograms, in 1D or 2D
% the data are on a (possibly incomplete) regular grid.
% the program computes variograms in the frequency domain by
% using 2D-FFT.
%
% input: x1:  data matrix. Missing values are indicated by NaN
%            icode: a code to indicate which function to compute
%               =1 : variogram
%               =2 : covariogram
%
% output:      gh11: variogram or covariogram depending on icode.
%              nh11: number of pairs available
%
% this program uses the functions FFT2, IFFT2, FFT2SHIFT and CONJ which are
% standard MATLAB functions.


[n,p]=size(x1);                             % dimensions of data matrix
nrows=2*n-1;
ncols=2*p-1;


% find the closest multiple of 8 to obtain a good compromise between
% speed (a power of 2) and memory required


nr2=ceil(nrows/8)*8;
nc2=ceil(ncols/8)*8;

% form an indicator  matrix:         1's for all data values
%                                           0's for missing values
% in data matrix, replace missing values by 0;


x1id=~isnan(x1);                            % 1 for a data value; 0 for missing
x1(~x1id)=zeros(sum(sum(~x1id)),1);         % missing replaced by 0


fx1=fft2(x1,nr2,nc2);                       % fourier transform of x1


if icode==1
  fx1_x1=fft2(x1.*x1,nr2,nc2);              % fourier transform of x1*x1
end
clear x1;


fx1id=fft2(x1id,nr2,nc2);                   % fourier transform of the indicator matri
clear x1id


% compute number of pairs at all lags


nh11=round(real(ifft2(conj(fx1id).*fx1id)));


% compute the different structural functions according to icode
```

```
if icode==1;                          % variogram is computed

  gh11=real(ifft2(conj(fx1id).*fx1_x1+conj(fx1_x1).*fx1id-2*conj(fx1).*fx1));
  gh11=gh11./max(nh11,1)/2;

else                                  % covariogram is computed

  m1=real(ifft2(conj(fx1).*fx1id))./max(nh11,1);    % compute tail mean
  m2=real(ifft2(conj(fx1id).*fx1))./max(nh11,1);    % compute head mean
  clear fx1id
  gh11=real(ifft2(conj(fx1).*fx1));
  gh11=gh11./max(nh11,1)-m1.*m2;
end

clear fx1 fx1id fx1_fx1

% reduce matrix to required size and shift so that the 0 lag appears at the center of each
matrix

nh11=[nh11(1:n,1:p) nh11(1:n,nc2-p+2:nc2);nh11(nr2-n+2:nr2,1:p) nh11(nr2-n+2:nr2,nc2-p+2:nc2)
gh11=[gh11(1:n,1:p) gh11(1:n,nc2-p+2:nc2);gh11(nr2-n+2:nr2,1:p) gh11(nr2-n+2:nr2,nc2-p+2:nc2)

gh11=fftshift(gh11);
nh11=fftshift(nh11);
```

APPENDIX 2

Listing of program Variof2.m

```
function
[gh11,nh11,gh12,nh12,gh22,nh22,gh13,nh13,gh23,nh23,gh33,nh33]=variof2(x1,x2,x3,icode);
%
% function
[gh11,nh11,gh12,nh12,gh22,nh22,gh13,nh13,gh23,nh23,gh33,nh33]=variof2(x1,x2,x3);
%
% function to compute variograms, cross-variograms, covariograms,
% cross-covariograms and pseudo-cross-variograms in 1D or 2D for up to 3 variables.
% the data are on a (possibly incomplete) regular grid
% the program computes variograms in the frequency domain using 2D-FFT.
%
% input:        x1,x2,x3:     data matrices.        Each matrix is either empty or of
size n x m
%                             Missing values are indicated by NaN
%              icode: a code to indicate which function to compute
%                =1 : variograms and cross-variograms;
%                =2 : covariograms and cross-covariograms
%                =3 : variograms and pseudo-cross-variograms
%
% output:      ghij:  (direct- and cross-) variograms for variables i and j, or
covariograms or
%              pseudo-variograms depending on icode.
%              nhij:  number of pairs available to compute the structural function
%
% this program uses the functions FFT2, IFFT2, FFT2SHIFT and CONJ which are
% standard MATLAB functions.


[n,p]=size(x1);                            % dimensions of data matrix
[n2,p2]=size(x2);
[n3,p3]=size(x3);


% find the closest multiple of 8 to obtain a good compromise between
% speed (a power of 2) and memory required


nrows=2*n-1;
ncols=2*p-1;
nr2=ceil(nrows/8)*8;
nc2=ceil(ncols/8)*8;


% form an indicator  matrix:        1's for all data values
%                                      0's for missing values
% in data matrix, replace missing values by 0;


x1id=~isnan(x1);                                   % 1 for a data value; 0 for missing
x1(~x1id)=zeros(sum(sum(~x1id)),1);        % missing replaced by 0


% compute fourier transforms


fx1=fft2(x1,nr2,nc2);                              % fourier transform ox x1
fx1_x1=fft2(x1.*x1,nr2,nc2);               % fourier transform of x1*x1
fx1id=fft2(x1id,nr2,nc2);
nh11=round(real(ifft2(conj(fx1id).*fx1id))); % number of pairs for x1 variogram


% do the same with x2 (if defined)
% compute number of pairs for cross-structural functions selected (icode)
```

```
if n2>0;                                    % begin test on n2
  x2id=~isnan(x2);
  x2(~x2id)=zeros(sum(sum(~x2id)),1);


  fx2=fft2(x2,nr2,nc2);
  fx2_x2=fft2(x2.*x2,nr2,nc2);
  fx2id=fft2(x2id,nr2,nc2);
  nh22=round(real(ifft2(conj(fx2id).*fx2id)));


  % number of pairs for cross-structural function x1-x2


  if icode==1
    fx12id=fft2(x1id.*x2id,nr2,nc2);
    nh12=round(real(ifft2(conj(fx12id).*fx12id)));
  else
    nh12=round(real(ifft2(conj(fx1id).*fx2id)));
  end
end                                         % end test on n2

% do the same with x3 (if defined)
% compute number of pairs for cross-structural functions selected (icode)


if n3>0;                                    % begin test on n3
  x3id=~isnan(x3);
  x3(~x3id)=zeros(sum(sum(~x3id)),1);


  fx3=fft2(x3,nr2,nc2);
  fx3_x3=fft2(x3.*x3,nr2,nc2);
  fx3id=fft2(x3id,nr2,nc2);
  nh33=round(real(ifft2(conj(fx3id).*fx3id)));


  % number of pairs for cross-structural function x1-x3, x2-x3


  if icode==1
    fx13id=fft2(x1id.*x3id,nr2,nc2);
    nh13=round(real(ifft2(conj(fx13id).*fx13id)));
    fx23id=fft2(x2id.*x3id,nr2,nc2);
    nh23=round(real(ifft2(conj(fx23id).*fx23id)));
  else
    clear x1 x2 x3 x1id x2id x3id
    nh13=round(real(ifft2(conj(fx1id).*fx3id)));
    nh23=round(real(ifft2(conj(fx2id).*fx3id)));
  end

end                                  % end test on n3

% compute the different structural functions according to icode

if icode==1;                         % variograms and cross-variograms are computed
if n3>0;
 gh33=real(ifft2(conj(fx3id).*fx3_x3+conj(fx3_x3).*fx3id-2*conj(fx3).*fx3))./max(nh33,1)/2;
    clear fx3id fx3_x3 fx3


    t1=fft2(x1.*x3id,nr2,nc2);
    t2=fft2(x3.*x1id,nr2,nc2);
    t12=fft2(x1.*x3,nr2,nc2);

gh13=real(ifft2(conj(fx13id).*t12+conj(t12).*fx13id-conj(t1).*t2-t1.*conj(t2)))./max(nh13,1
clear fx13id


    t1=fft2(x2.*x3id,nr2,nc2);
    t2=fft2(x3.*x2id,nr2,nc2);
    t12=fft2(x2.*x3,nr2,nc2);
    clear x3 x3id

gh23=real(ifft2(conj(fx23id).*t12+conj(t12).*fx23id-conj(t1).*t2-t1.*conj(t2)))./max(n
h23,1)/2;
    clear fx23id
  end


 if n2>0;

gh22=real(ifft2(conj(fx2id).*fx2_x2+conj(fx2_x2).*fx2id-2*conj(fx2).*fx2))./max(nh22,1
)/2;
    clear fx2id fx2_x2 fx2


    t1=fft2(x1.*x2id,nr2,nc2);              % FFT on points available for
cross-variogram computation
    clear x2id
    t2=fft2(x2.*x1id,nr2,nc2);
    clear x1id
    t12=fft2(x1.*x2,nr2,nc2);
    clear x1 x2

gh12=real(ifft2(conj(fx12id).*t12+conj(t12).*(fx12id)-conj(t1).*t2-t1.*conj(t2)))./max
(nh12,1)/2;
    clear fx12id t1 t2 t12
  end



gh11=real(ifft2(conj(fx1id).*fx1_x1+conj(fx1_x1).*fx1id-2*conj(fx1).*fx1))./max(nh11,1
)/2;

elseif icode==2;                  % covariograms and cross-covariograms are
computed


  clear fx1_x1 fx2_x2 fx3_x3

  if n3>0;
    m1=real(ifft2(conj(fx3id).*fx3))./max(nh33,1);          % computes the tail x3
means
    m2=real(ifft2(fx3id.*conj(fx3)))./max(nh33,1);          % computes the head x3
means
    gh33=real((ifft2(conj(fx3).*fx3))./max(nh33,1)-m1.*m2);

    m1=real(ifft2(conj(fx1).*fx3id))./max(nh13,1);          % computes the tail x1
means
   m2=real(ifft2(conj(fx1id).*fx3))./max(nh13,1);    % computes the head x3 means
    gh13=real((ifft2(conj(fx1).*fx3))./max(nh13,1)-m1.*m2);

    m1=real(ifft2(conj(fx2).*fx3id))./max(nh23,1);          % computes the tail x2
means
    m2=real(ifft2(conj(fx2id).*fx3))./max(nh23,1);          % computes the head x3
means
    gh23=real((ifft2(conj(fx2).*fx3))./max(nh23,1)-m1.*m2);
    clear fx3 fx3id
  end


  if n2>0;
    m1=real(ifft2(conj(fx2).*fx2id))./max(nh22,1);          % computes the tail x2
means
    m2=real(ifft2(conj(fx2id).*fx2))./max(nh22,1);          % computes the head x2
means
    gh22=real((ifft2(conj(fx2).*fx2))./max(nh22,1)-m1.*m2);

    m1=real(ifft2(conj(fx1).*fx2id))./max(nh12,1);          % computes the tail x1
means
    m2=real(ifft2(conj(fx1id).*fx2))./max(nh12,1);          % computes the head x2
means
    gh12=real(ifft2(conj(fx1).*fx2))./max(nh12,1)-m1.*m2;
    clear fx2 fx2id
  end
  m1=real(ifft2(conj(fx1).*fx1id))./max(nh11,1);            % computes the tail x1
means
  m2=real(ifft2(conj(fx1id).*fx1))./max(nh11,1);          % computes the head x1 means
```

```
ghll=real((ifft2(conj(fxl).*fxl))./max(nhll,1)-ml.*m2);

elseif icode==3                           % variograms and pseudo-cross-variograms are
computed

  if n3>0;
 gh33=real(ifft2(conj(fx3id).*fx3_x3+conj(fx3_x3).*fx3id-2*conj(fx3).*fx3))./max(nh33,1)/2;
 gh13=real(ifft2(fx3id.*conj(fxl_xl)+conj(fxlid).*fx3_x3-2*conj(fxl).*fx3))./max(nh13,1)/2;
 gh23=real(ifft2(fx3id.*conj(fx2_x2)+conj(fx2id).*fx3_x3-2*conj(fx2).*fx3))./max(nh23,1)/2;
    clear fx3id fx3 fx3_x3
  end

  if n2>0;
 gh22=real(ifft2(conj(fx2id).*fx2_x2+conj(fx2_x2).*fx2id-2*conj(fx2).*fx2))./max(nh22,1)/2;
 gh12=real(ifft2(fx2id.*conj(fxl_xl)+conj(fxlid).*fx2_x2-2*conj(fxl).*fx2))./max(nh12,1)/2;
    clear fx2id fx2 fx2_x2
  end

 ghll=real(ifft2(conj(fxlid).*fxl_xl+conj(fxl_xl).*fxlid-2*conj(fxl).*fxl))./max(nhll,1)/2;

end                                       % end test on icode

% reduce matrices to required size

nhll=[nhll(1:n,1:p) nhll(1:n,nc2-p+2:nc2);nhll(nr2-n+2:nr2,1:p) nhll(nr2-n+2:nr2,nc2-p+2:nc2)
ghll=[ghll(1:n,1:p) ghll(1:n,nc2-p+2:nc2);ghll(nr2-n+2:nr2,1:p) ghll(nr2-n+2:nr2,nc2-p+2:nc2)

if n2>0
  nh22=[nh22(1:n,1:p) nh22(1:n,nc2-p+2:nc2);nh22(nr2-n+2:nr2,1:p) nh22(nr2-n+2:nr2,nc2-p+2:nc2)];
  gh22=[gh22(1:n,1:p) gh22(1:n,nc2-p+2:nc2);gh22(nr2-n+2:nr2,1:p) gh22(nr2-n+2:nr2,nc2-p+2:nc2)];

  nh12=[nh12(1:n,1:p) nh12(1:n,nc2-p+2:nc2);nh12(nr2-n+2:nr2,1:p) nh12(nr2-n+2:nr2,nc2-p+2:nc2)];
gh12=[gh12(1:n,1:p) gh12(1:n,nc2-p+2:nc2);gh12(nr2-n+2:nr2,1:p) gh12(nr2-n+2:nr2,nc2-p+2:nc2)

  if n3>0
                     nh33=[nh33(1:n,1:p)       nh33(1:n,nc2-p+2:nc2);nh33(nr2-n+2:nr2,1:p)
nh33(nr2-n+2:nr2,nc2-p+2:nc2)];
                     gh33=[gh33(1:n,1:p)       gh33(1:n,nc2-p+2:nc2);gh33(nr2-n+2:nr2,1:p)
gh33(nr2-n+2:nr2,nc2-p+2:nc2)];
                     nh13=[nh13(1:n,1:p)       nh13(1:n,nc2-p+2:nc2);nh13(nr2-n+2:nr2,1:p)
nh13(nr2-n+2:nr2,nc2-p+2:nc2)];
                     gh13=[gh13(1:n,1:p)       gh13(1:n,nc2-p+2:nc2);gh13(nr2-n+2:nr2,1:p)
gh13(nr2-n+2:nr2,nc2-p+2:nc2)];
                     nh23=[nh23(1:n,1:p)       nh23(1:n,nc2-p+2:nc2);nh23(nr2-n+2:nr2,1:p)
nh23(nr2-n+2:nr2,nc2-p+2:nc2)];
                     gh23=[gh23(1:n,1:p)       gh23(1:n,nc2-p+2:nc2);gh23(nr2-n+2:nr2,1:p)
gh23(nr2-n+2:nr2,nc2-p+2:nc2)];
  end
end

% shift all the matrices so that the 0 lag appears at the center of each matrix

nhll=fftshift(nhll);
ghll=fftshift(ghll);
nh22=fftshift(nh22);
gh22=fftshift(gh22);
nh12=fftshift(nh12);
gh12=fftshift(gh12);
nh33=fftshift(nh33);
gh33=fftshift(gh33);
nh13=fftshift(nh13);
gh13=fftshift(gh13);
nh23=fftshift(nh23);
gh23=fftshift(gh23);
```