Sequential Gaussian Simulation for Large Grids (`sgsim_pw`)

Julián Ortiz C. (jmo1@ualberta.ca) and Clayton V. Deutsch (cdeutsch@ualberta.ca)
Department of Civil & Environmental Engineering, University of Alberta

Abstract

The GSLIB program `sgsim` *cannot handle large grids due to limitations in RAM memory. The algorithm has been modified in order to permit simulation of large grids. The new algorithm* `sgsim_pw` *simulates pieces of the larger model, so each piece can be handled with the available RAM memory. To preserve the spatial continuity, a coarse grid is simulated first and added to the conditioning data, so when simulating a piece, there are conditioning points to honor the long range correlation. Moreover, an overlapping zone is used to preserve the correct transition from one piece to the next. This avoids discontinuities and artifacts due to the separate generation of the pieces. Changes in the original code are presented in this short note. Examples are shown. To avoid large files, a binary output is implemented (*`sgsim_pwb`*). Two additional programs are presented:* `getsim` *extracts one or all realization of the binary output file and writes them as an ASCII file, and* `getslice` *permits to extract a slice from a large ASCII file, so that other GSLIB programs do not present allocation problems.*

Introduction

Sequential Gaussian Simulation is one of the most widely used geostatistical simulation techniques. Application to dense grids for large domains required the modification of the code to avoid RAM memory limitations.

`sgsim` requires roughly $4N$ bytes of RAM to simulate a grid of N nodes. A computer with 128MB of free RAM would be limited to approximately 30 million cells. One way to overcome the RAM limitation is to simulate piece-wise. The grid to be simulated, which has nz levels, is divided into $npie$ pieces with $nx \cdot ny$ nodes in the horizontal plane and $\frac{nz}{npie}$ levels each. The algorithm is presented schematically in **Figure 1**:

1. Set parameters to simulate a coarse grid: the new dimensions of the grid, number of nodes and coordinates of the first node have to be reset. The original parameters are kept. The dynamic allocation of memory is done using new parameters that account for the size of the coarse grid and the number of nodes to be simulated at one piece that includes overlapping.

2. Simulate the coarse grid: using the specified parameters for the coarse grid, the routine `sgsim_pw` is called. The output is kept in an array and transferred to the next step.

3. Add the coarse grid to the conditioning data array: based on the specifications for the coarse grid, the locations and corresponding coordinates are calculated for each simulated node. They are loaded into the data array to be considered as hard data during the dense simulation.

4. Set parameters to simulate a piece of the dense grid: the specifications for the dense grid simulation of a piece are made. Depending on the piece number the size of the sub-model

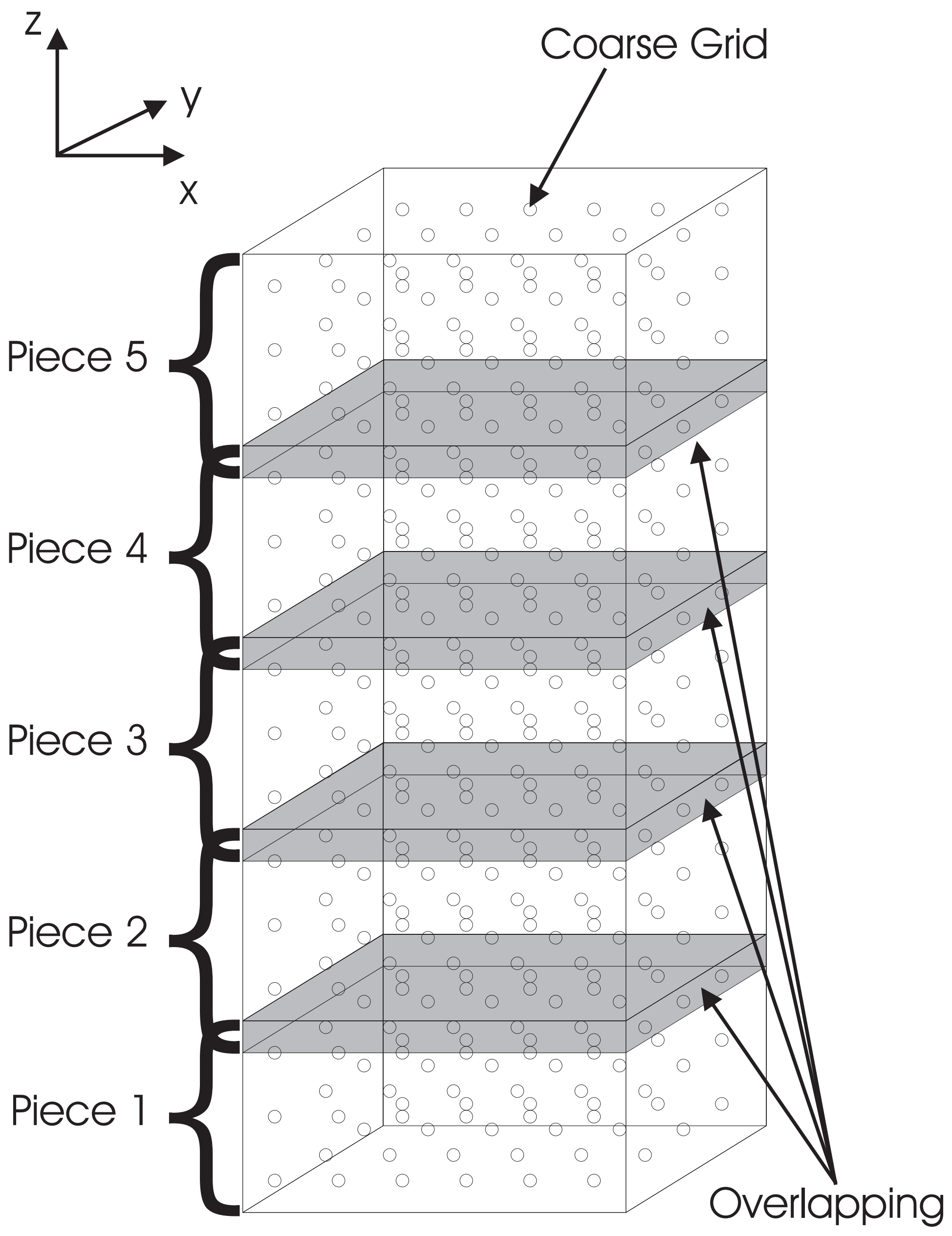

Figure 1: Schematic description of `sgsim_pw` algorithm.

will include the overlapped nodes or not. The size of the first piece is $nx \cdot ny \cdot \frac{nz}{npie}$, while the size of the other $npie - 1$ pieces is $nx \cdot ny \cdot \left(\frac{nz}{npie} + over\right)$.

5. Start piece-wise simulation: looping over the pieces, the following steps are repeated:

   (a) Set parameters depending on the piece: the dimension in the z direction depends on the piece number. The first piece does not include an overlapping zone from the previous piece, since there is none. The dimension in the z direction and consequently, the total number of nodes are reset.

   (b) Simulate the piece: given the current parameters, `sgsim_pw` is called to simulate the dense piece.

   (c) Keep the nodes in the overlapping zone: as defined by the user, a zone of *over* levels is used in the next piece. That makes a total number of *nover* nodes to keep. Those nodes are stored in a temporary array and loaded into the grid for the next piece before simulating, so they are considered informed, hence skipped by the algorithm.

Modifications to the code

Modifications in several units of the `sgsim` were required. The main program was modified to call several times `sgsim_pw`, to complete the pieces. The loop to simulate several realizations was also taken out the original `sgsim` routine, to the main program.

The definition of the variables was the most important modification, since this is where the problem of memory allocation is encountered. Each array was modified to handle the larger piece to be simulated. Parameters for maximum sizes of arrays were also changed.

Finally, a few parameters were added to the parameter file to give the user the flexibility to manipulate the sizes of the pieces, overlapping and coarse grid.

Changes to the main program

The code was modified to create different realizations in the main program, and perform the step-wise simulation. The variables *ipie* and *nover* were added to the parameters used when calling the routine `sgsim_pw`:

```
      call sgsim_pw(MAXNOD,MAXCXY,MAXCTX,MAXCTY,MAXCTZ,
     +              MAXSBX,MAXSBY,MAXSBZ,ipie,nover)
```

Details of changes made to the main program:

1. Keep original parameters

```
      nxold=nx
      nyold=ny
      nzold=nz
      xsizold=xsiz
      ysizold=ysiz
      zsizold=zsiz
      xmnold=xmn
      ymnold=ymn
      zmnold=zmn
      ndold=nd
```

2. Loop over realizations:

```
do isim=1,nsim
```

3. Set parameters to simulate a coarse grid:

```
nd=ndold            % Set number of data

ipie=0              % Set piece number

if(nx.ne.1) nx=int(nxold/ratio) %
if(ny.ne.1) ny=int(nyold/ratio) % Reset number of nodes
if(nz.ne.1) nz=int(nzold/ratio) %

if(nx.ne.1) xsiz=xsizold*ratio  %
if(ny.ne.1) ysiz=ysizold*ratio  % Reset spacing between nodes
if(nz.ne.1) zsiz=zsizold*ratio  %

nxy=nx*ny           % Reset number of nodes in horizontal plane

nxyz=nx*ny*nz       % Reset number of nodes overall

if(nx.ne.1) xmn=xmnold+(xsiz/2)-(xsizold/2) %
if(ny.ne.1) ymn=ymnold+(ysiz/2)-(ysizold/2) % Reset new origin
if(nz.ne.1) zmn=zmnold+(zsiz/2)-(zsizold/2) %
```

4. Simulate the coarse grid:

```
   call sgsim_pw(MAXNOD,MAXCXY,MAXCTX,MAXCTY,MAXCTZ,
+               MAXSBX,MAXSBY,MAXSBZ,ipie,nover)
```

5. Add the coarse grid to the conditioning data array:

```
ndnew = ndold + nxyz                % Temporary number of data

do in=nd+1,ndnew                    % Loop over simulated nodes
    in2=in-nd

    iz=1+int((in2-1)/nxy)           %
    iy=1+int((in2-1-(iz-1)*nxy)/nx) % Get index in large grid
    ix=in2-(iz-1)*nxy-(iy-1)*nx     %

    x(in)=xmn+real(ix-1)*xsiz       %
    y(in)=ymn+real(iy-1)*ysiz       % Assign coordinate
    z(in)=zmn+real(iz-1)*zsiz       %

    vr(in)=coarse(in2)              % Assign value

end do                              % Finish loop

nd=ndold+nxyz                       % Update number of data
```

6. Set parameters to simulate a dense grid of size $nx \cdot ny \cdot nz_{piece}$

```
nx=nxold            %
ny=nyold            % Reset number of nodes
nz=nzold/npie       %
```

```
xsiz=xsizold        %
ysiz=ysizold        % Reset spacing between nodes
zsiz=zsizold        %

nxy=nx*ny           % Reset number of nodes in horizontal plane

nxyz=nx*ny*nz       % Reset number of nodes overall

xmn=xmnold          % Reset new origin x and y only
ymn=ymnold          %

nover=nx*ny*over    % Set number of nodes to overlap
```

7. Start piece-wise simulation. Loop over pieces:

   ```
   do ipie=1,npie
   ```

   (a) Set parameters depending on the piece:

   ```
       zmn=zmnold                      % Reset z origin

       if (ipie.ne.1) then             %
           nz=nzold/npie+over          %
           zmn=zmnold+(ipie-1)*        % Reset number of nodes
   +         (nz-over)*zsiz-over*zsiz  % to account for overlapping
           nxyz=nx*ny*nz               %
       end if                          %
   ```

   (b) Simulate the piece:

   ```
       call sgsim_pw(MAXNOD,MAXCXY,MAXCTX,MAXCTY,MAXCTZ,
   +                 MAXSBX,MAXSBY,MAXSBZ,ipie,nover)
   ```

   (c) Keep the nodes in the overlapping zone:

   ```
       do iover=1,nover
          tover(iover)=sim(nxyz-nover+iover)
       end do
   ```

   (d) End loop over pieces.

8. End loop over realizations.

Changes in definition of variables and input parameters

The `readparm` routine reads the new parameters added to the parameter file (see below). Three new parameters were added:

- `ratio`: refers to the ratio of nodes in the dense grid over nodes in the coarse grid in one direction. That means that if the dense grid has nx nodes in x, then the coarse grid has $\frac{nx}{ratio}$ nodes. Overall, the coarse grid has $\frac{nx \cdot ny \cdot nz}{ratio^3}$ nodes.
- `npie`: defines the number of pieces that the large model is divided into. Each piece should be small enough so that it does not exceed the maximum size that the RAM memory can handle. The overlapping grid must be included in this calculation.
- `over`: Defines how many levels in z are to be overlapped, that is, to be used for the next piece.

A few variables were added to store the coarse grid and overlapping simulated nodes:

- *tover*(:): Stores the overlapping nodes.
- *coarse*(:): Stores the coarse grid to load it into the data array.

Maximum sizes of arrays were changed to account only for pieces and overlapping, and not the entire dense grid:

- Maximum size on the z direction:

```
if(nz/ratio.gt.nz/npie+over) then   %
    MAXZ   = nz/ratio               %
else                                % Set as maximum possible case
    MAXZ   = nz/npie+over           %
end if                              %

MXYZ   = MAXX * MAXY * MAXZ         % Reset size of simulated arrays
```

- Size of the overlapping array:

```
MXYZO  = MAXX * MAXY * over
```

- Size of data arrays:

```
MAXDAT = MAXDAT + nx*ny*nz/ratio**3
```

- Dynamic memory was assigned to the new variables:
  - $coarse(MAXDAT)$
  - $tover(MXYZO)$
- Define maximum size of coarse grid and data array:

```
nxc=1
nyc=1
nzc=1

if(nx.ne.1) nxc=int(nx/ratio)
if(ny.ne.1) nyc=int(ny/ratio)
if(nz.ne.1) nzc=int(nz/ratio)

MAXCOARSE = nxc*nyc*nzc

MAXDAT = MAXDAT + MAXCOARSE
```

Changes to the routine `sgsim`

As mentioned above, the loop over realizations was taken out of the original `sgsim` routine. The overlapping nodes were handled as follows:

- When assigning data to nodes, the nodes outside the grid were not considered:

```
call getindx(nx,xmn,xsiz,x(id),ix,testind)
if(.not.testind) goto 77
call getindx(ny,ymn,ysiz,y(id),iy,testind)
if(.not.testind) goto 77
call getindx(nz,zmn,zsiz,z(id),iz,testind)
if(.not.testind) goto 77
```

- Overlapping nodes from the previous piece were entered into the new array of simulated nodes:

```
    if(ipie.gt.1) then
        do iover=1,nover
            sim(iover)=tover(iover)
        end do
    end if
```

- The simulated values are entirely written to the output file only if the piece number is one. Otherwise, only the nodes that do not overlap are written out. The parameter *ntmp* is set so nodes of the coarse grid are stored into the array *coarse*(). If *ipie* is different than 0, which corresponds to the coarse grid, the simulated nodes are read in two steps:

  – The first *nover* nodes are read first and those are written out only if it is the first piece.

  – The remaining $nxyz - nover$ nodes are written out in all cases.

  This is done with the following code:

```
% Set the value for ntmp
    if(ipie.eq.0) then
        ntmp=nxyz
        else
        ntmp=nover
    end if

% Write out the first piece or save the coarse grid
% If it is not the first piece those nodes are overlapped,
% so do not write them out.
    do ind=1,ntmp
        simval = sim(ind)
        if(simval.gt.-9.0.and.simval.lt.9.0) then
            ne = ne + 1
            av = av + simval
            ss = ss + simval*simval
        end if
        if (ipie.eq.0)then
            coarse(ind)=simval
        end if
            if(itrans.eq.1.and.simval.gt.(UNEST+EPSLON)) then
                simval = backtr(simval,ntr,vrtr,vrgtr,zmin,
 +                                zmax,ltail,ltpar,utail,utpar)
                if(simval.lt.zmin) simval = zmin
                if(simval.gt.zmax) simval = zmax
            end if
            if (ipie.eq.1)then
                write(lout,'(f12.4)') simval
        end if
    end do

% Write out the other pieces or remaining coarse grid nodes.
    do ind=ntmp+1,nxyz
        simval = sim(ind)
        if(simval.gt.-9.0.and.simval.lt.9.0) then
            ne = ne + 1
            av = av + simval
            ss = ss + simval*simval
        end if
        if (ipie.eq.0)then
            coarse(ind)=simval
        end if
        if(itrans.eq.1.and.simval.gt.(UNEST+EPSLON)) then
            simval = backtr(simval,ntr,vrtr,vrgtr,zmin,
     +                  zmax,ltail,ltpar,utail,utpar)
            if(simval.lt.zmin) simval = zmin
            if(simval.gt.zmax) simval = zmax
        end if
        if (ipie.ne.0)then
            write(lout,'(f12.4)') simval
        end if
        end do
        av = av / max(real(ne),1.0)
        ss =(ss / max(real(ne),1.0)) - av * av
        write(*,   111) isim,ne,ipie,av,ss
        write(ldbg,111) isim,ne,ipie,av,ss
111     format(/,' Realization ',i3,': number   = ',i8,/,
    +              ' Piece       ',i3,/,
    +              '                 mean     = ',f12.4,
    +              ' (close to 0.0?)',/,
    +              '                 variance = ',f12.4,
    +              ' (close to gammabar(V,V)? approx. 1.0)',/)
```

Examples

Example 1: Two Dimensional Grid with Short Variogram Range

A two dimensional unconditional realization of a 500 by 1000 nodes grid was done using both `sgsim_pw` and `sgsim` to check for possible artifacts due to the separation of the model into smaller pieces. The histogram, map and variogram reproduction were checked (**Figure 2**). The result is very satisfactory. The parameters used in `sgsim_pw` were:

- Ratio for coarse grid: 10. That is, 5000 nodes are simulated in the coarse grid.
- Number of pieces: 10. The first piece has 50000 nodes.
- Levels to overlap: 5. The pieces 2 to 10 have 50000 + 2500 = 52500 nodes.
- Variogram range: 10.

In this case, the overlapping zone was chosen so it covers half the range of correlation given by the variogram. The spacing for the coarse grid was 10, that means that it is not really helping to reproduce the variogram. It may not be necessary in some cases to use it, particularly, when the range of the variogram is short and can be covered mostly using the overlapping zone. With large ranges of correlation, the coarse grid should play a more important role.

Example 2: Two Dimensional Grid with Large Variogram Range

A second two dimensional example with large range gave the results presented in **Figure 3**. In this case, the parameters used in `sgsim_pw` were:

- Ratio for coarse grid: 10. That is, 5000 nodes are simulated in the coarse grid.
- Number of pieces: 10. The first piece has 50000 nodes.
- Levels to overlap: 5. The pieces 2 to 10 have 50000 + 2500 = 52500 nodes.

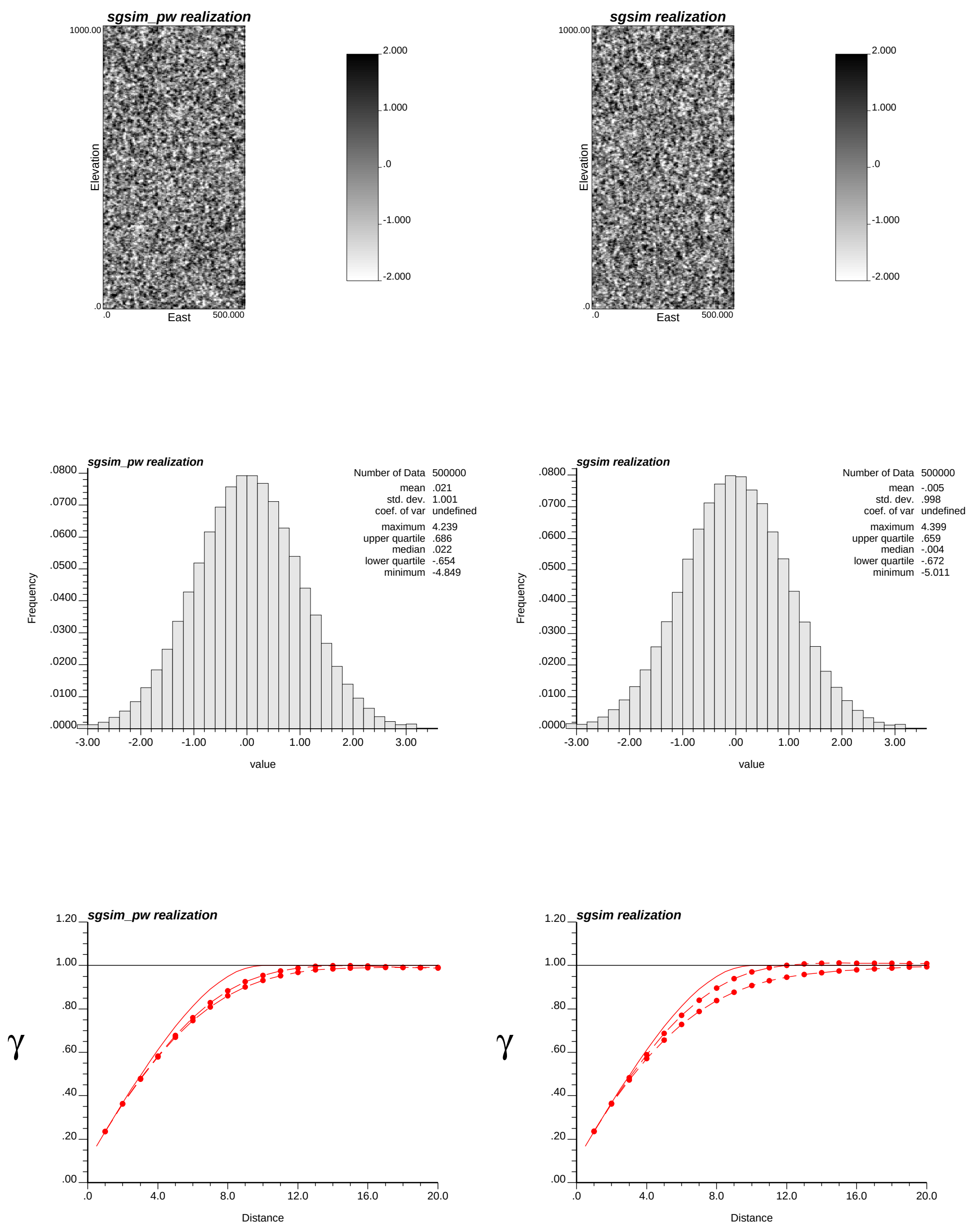

Figure 2: Histogram, map and variogram for a single short range realization done with `sgsim_pw` and `sgsim`.

- Variogram range: 100.

In this example, most of the variogram reproduction is due to the coarse grid. The overlapping zone only helps to get a smooth transition between pieces, without any artifact. Again the result is satisfactory. The variogram reproduction is not as good for long range. This may be due to ergodic fluctuations only.

Example 3: Two Dimensional Grid - Multiple Realizations

To check the variogram reproduction, 10 realizations of the same grid -500 by 1000 nodes- were generated using both algorithms, for a shorter variogram range. **Figure 4** shows the better performance of `sgsim_pw` compared with `sgsim` in reproducing the variogram. It can be concluded that the new algorithm does not have problems reproducing the variogram. The parameters used in the piece-wise algorithm are:

- Ratio for coarse grid: 10. That is, 5000 nodes are simulated in the coarse grid.
- Number of pieces: 10. The first piece has 50000 nodes.
- Levels to overlap: 5. The pieces 2 to 10 have $50000 + 2500 = 52500$ nodes.
- Variogram range: 50.

Example 4: Two Dimensional Grid with Anisotropic Variogram

Anisotropy in the vertical direction was used to test `sgsim_pw`. A variogram with range of 50 units in the direction East with dip 30^o, and 25 units in the perpendicular direction was used to generate 10 realizations. The same parameters than in the previous example were used for the piece-wise construction of the model. Results are shown in **Figure 5**.

Example 5: Two Dimensional Grid with Conditioning Data

The program was also tested using conditioning data. The database `red.dat` was used to test in two-dimensions data reproduction. A 600 by 1200 nodes grid was used to simulate in a very fine grid (0.5 by 0.5 units). 10 realizations using `sgsim_pw` and `sgsim` were generated. Histogram and variogram reproduction were checked. The data were honored by all realizations.

Example 6: Three Dimensional Grid

Finally, a three dimensional realization was generated using conditioning data. The grid size was 400 by 600 by 200 nodes, i.e. 48 million nodes. The output honors the conditioning data. A slice was taken to show the map and histogram (**Figure 7**).

Testing Performance

The performance of both `sgsim_pw` and `sgsim` was tested, by running several models and checking the time to accomplish the task.

`sgsim` runs about 10% faster than `sgsim_pw`, which is due to the simulation of the coarse grid at the beginning of the simulation.

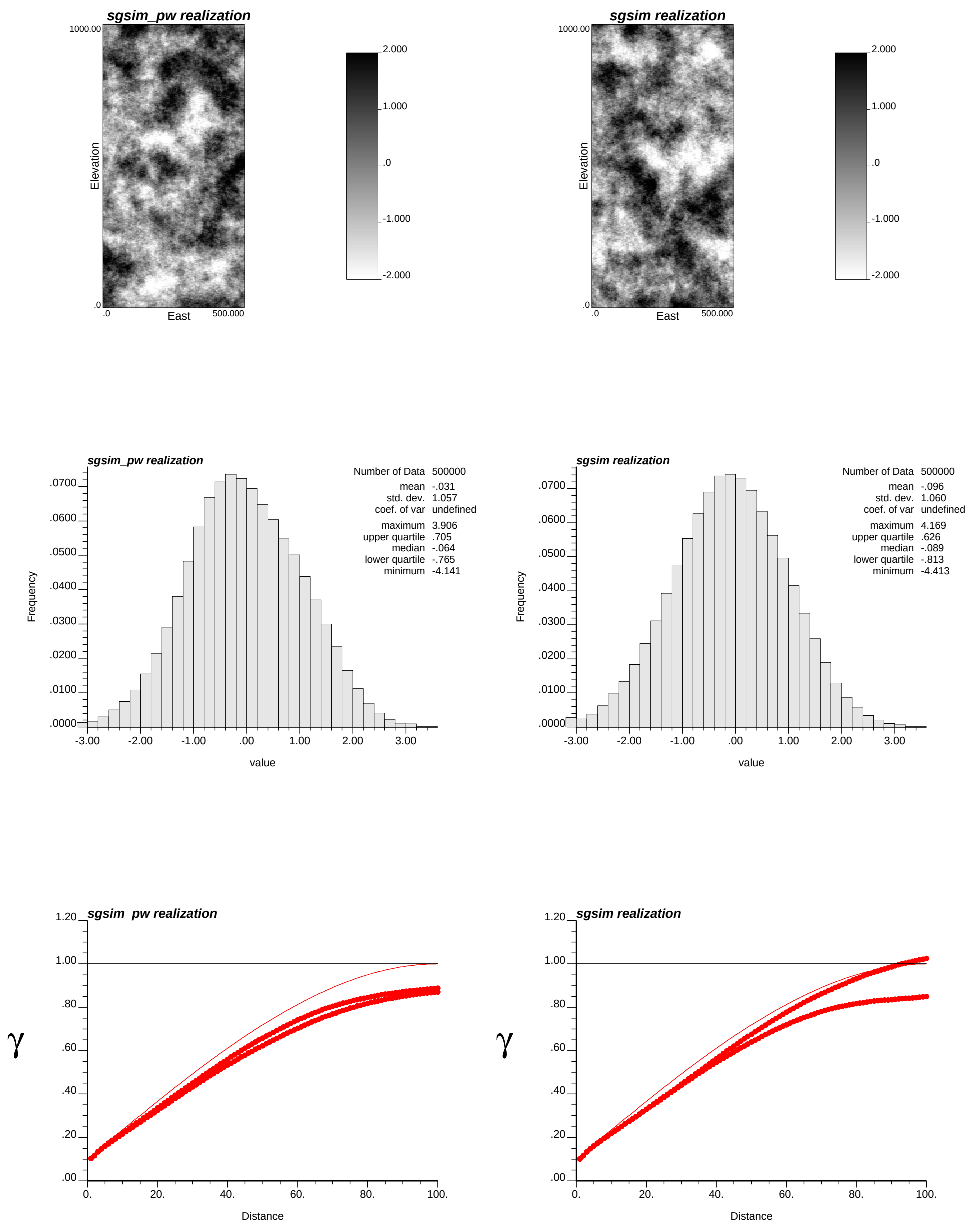

Figure 3: Histogram, map and variogram for single realizations with large range done with `sgsim_pw` and `sgsim`.

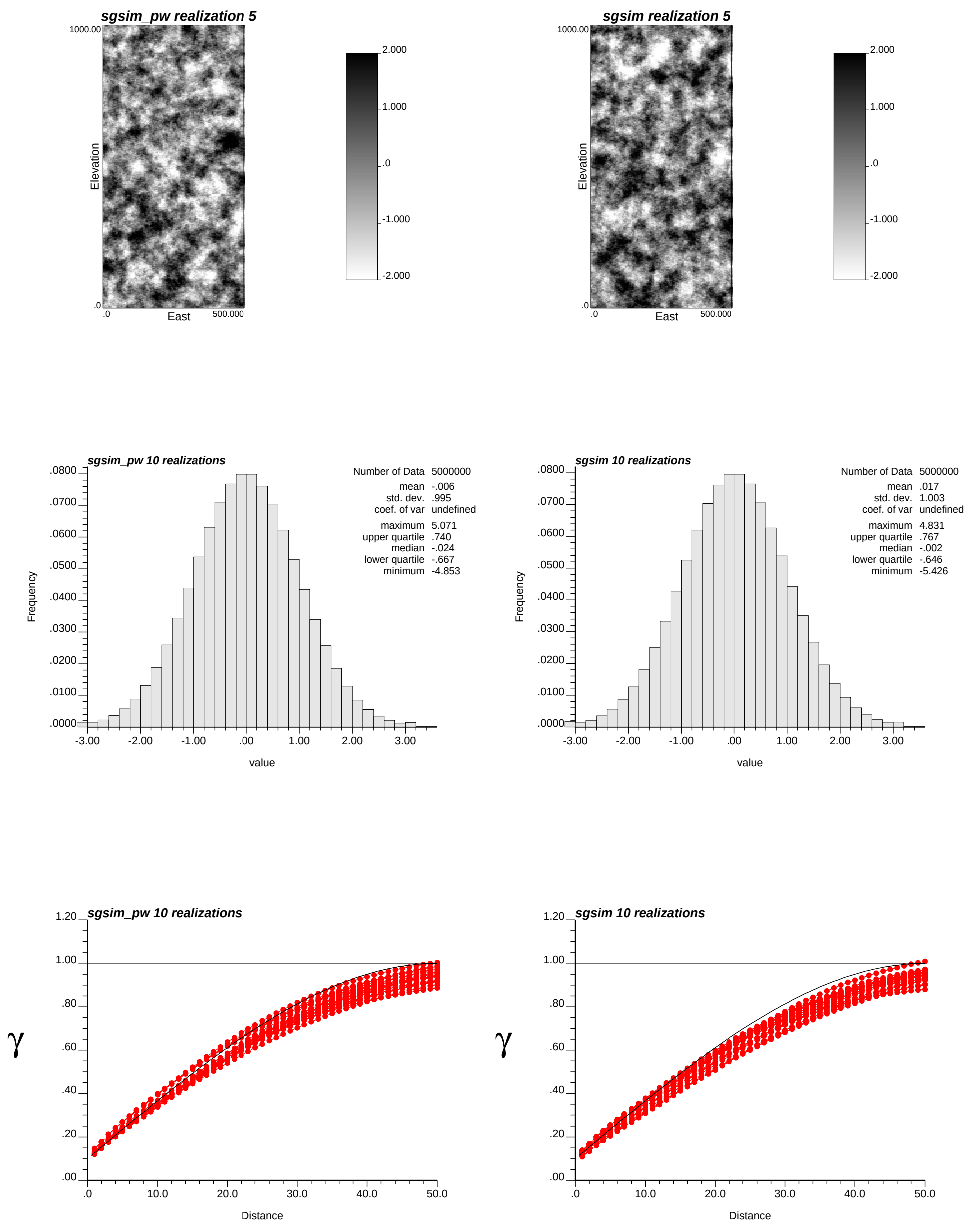

Figure 4: Joint histogram of 10 realizations, map of the fifth realization and variograms showing reproduction using `sgsim_pw` and `sgsim`.

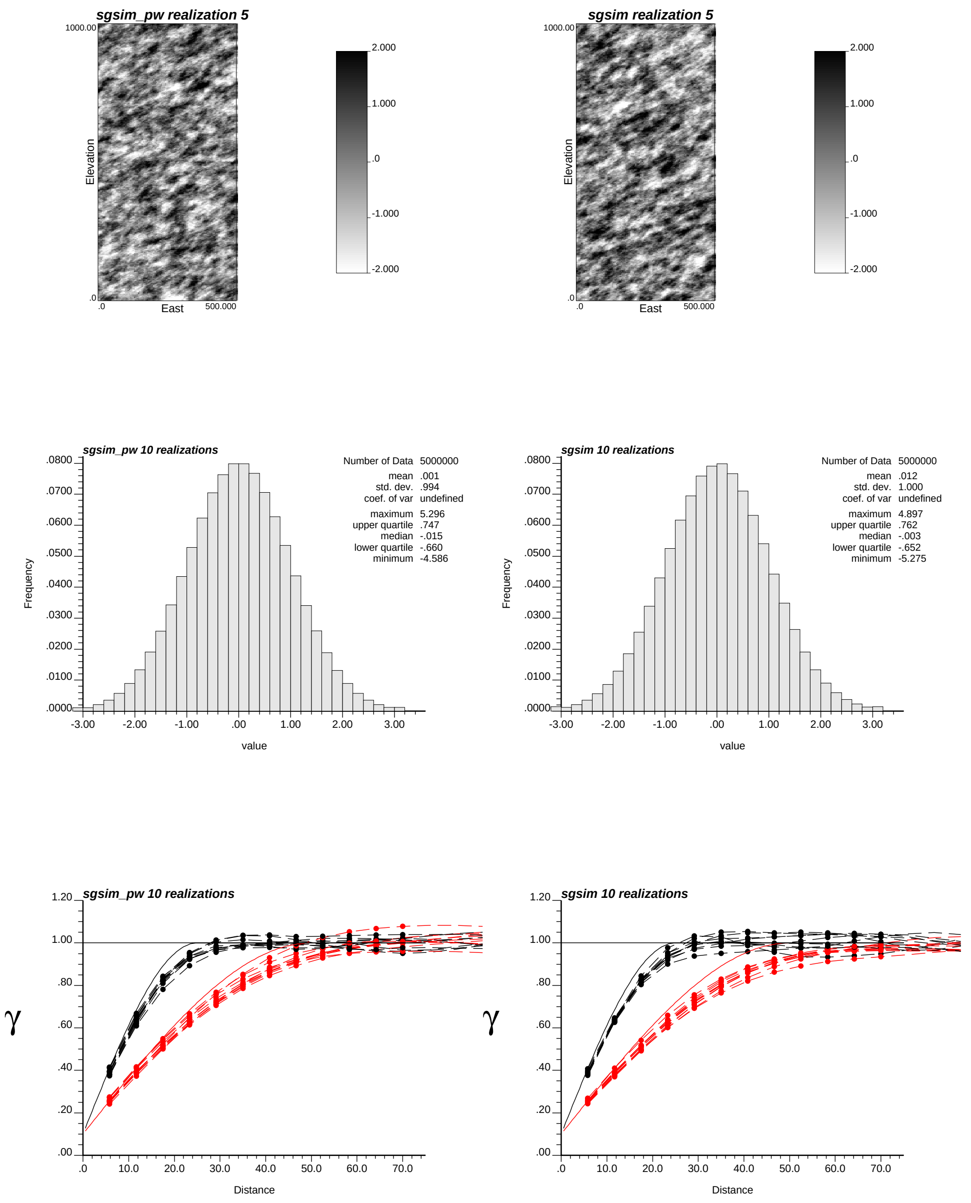

Figure 5: Joint histogram of 10 realizations, map of the fifth realization and variograms showing reproduction for an anisotropic model using `sgsim_pw` and `sgsim`.

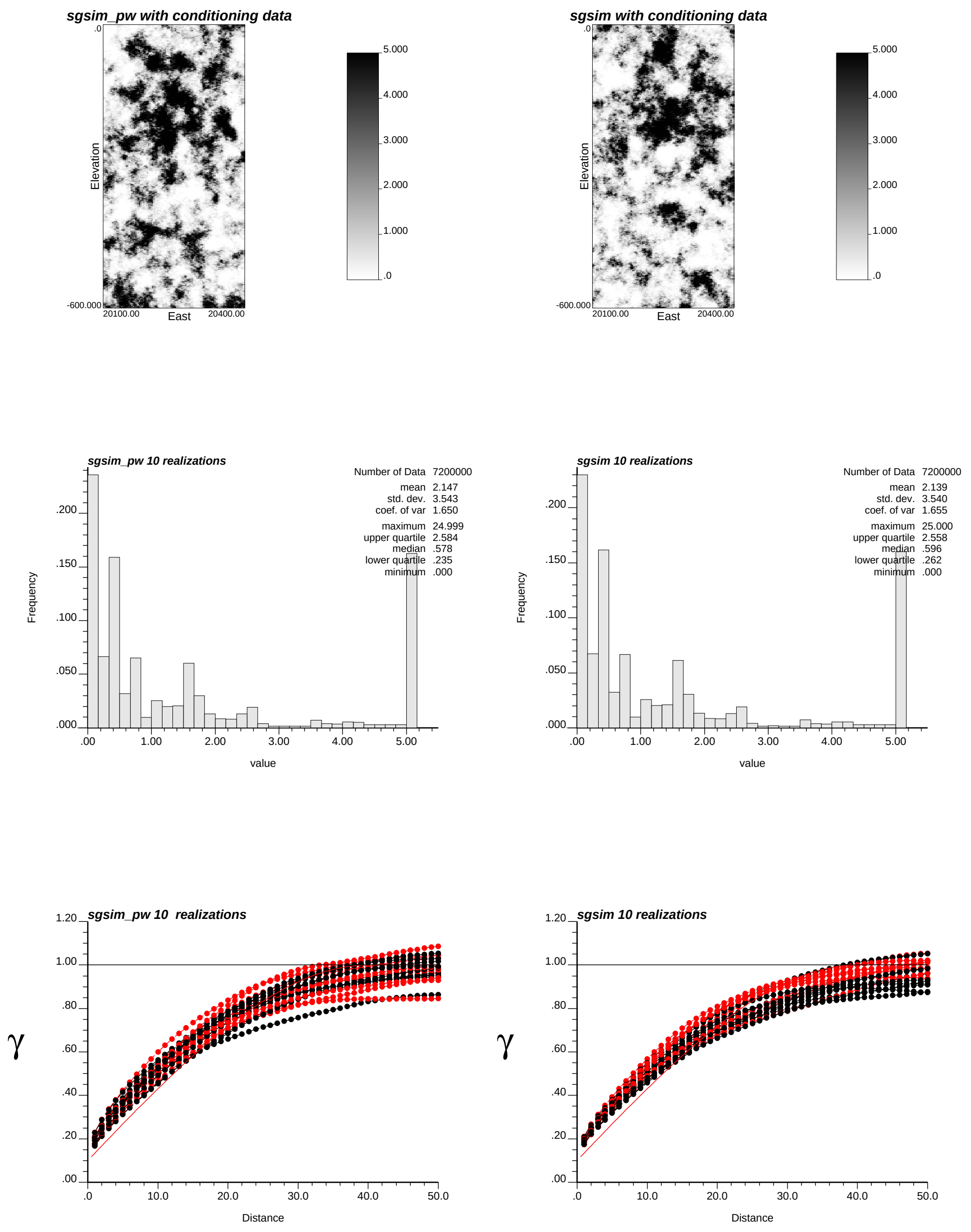

Figure 6: Joint histogram of 10 realizations, map of the fifth realization and variograms showing reproduction using `sgsim_pw` and `sgsim`, and the conditioning database `red.dat`.

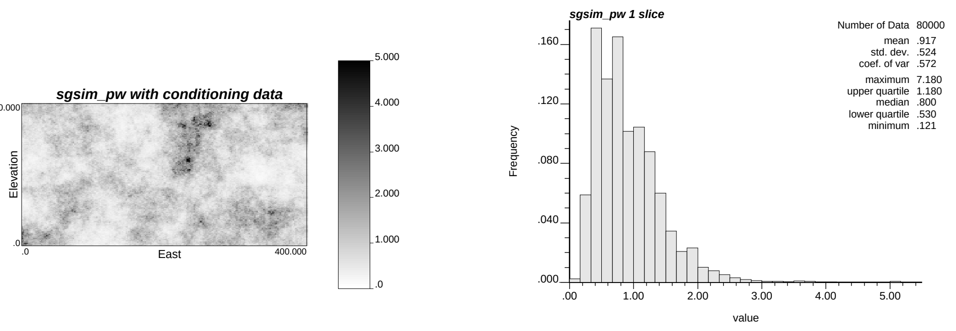

Figure 7: Map and histogram of a slice of a large realization done using `sgsim_pw`. This realization ins conditioned by the dataset `data1.dat`.

model size	`sgsim_pw`				`sgsim`
(nodes)	npie	ratio	nover	time(minutes)	time(minutes)
1M = 100 x 100 x 100	10	10	5	2	2
8M = 200 x 200 x 200	10	10	5	19	18
27M = 300 x 300 x 300	20	20	5	59	—
64M = 400 x 400 x 400	40	20	5	136	—
125M = 500 x 500 x 500	50	20	5	228	—

Table 1: Performance of `sgsim_pw` and `sgsim` for different grid sizes.

Handling Large Files

Since large numerical models occupy large amounts of memory when stored, a binary outputting has been implemented. The program `sgsim_pwb` writes the output file in binary format. It is therefore not possible to read the file directly, so a utility program has been prepared to extract from the large output file the realizations. This program allows the user to chose between extracting all realizations or just picking one. The output is a standard ASCII file.

The parameter file of the program `getsim` is very straightforward (**Figure 8**).

Additionally, since most programs in GSLIB will store the entire realization before performing the operation required (e.g. `pixelplt`), a program called
t getslice is provided, that allows the user to output a given slice into an ASCII file, reducing the size of the array and speeding up subsequent programs. The parameter file is also simple (**Figure 9**).

```
                  Parameters for GETSIM
                  *********************

START OF PARAMETERS:
sgsim_pw.out                    -binary file with simulated realizations
0                               -extract all realizations(1=yes,0=no)
1                               -  realization to extract (if 0)
getsim.out                      -ascii file to output realization
```

Figure 8: Parameter file for `getsim`.

```
              Parameters for getslice
              ***********************

START OF PARAMETERS:
sgsim_pw.out                 - Input file with realizations
1                            - column for variable
10                           - number of realizations in file
100 100 100                  - nx, ny, nz
1                            - realization number
1                            - slice orientation: 1=XY, 2=XZ, 3=YZ
1                            - slice number
getslice.out                 - Output file with slice
```

Figure 9: Parameter file for `getslice`

Discussion

Large numerical models can now be constructed with a few, conceptually simple, modifications to the code of `sgsim`. The new program, `sgsim_pw` can handle larger models. Models up to 125 million nodes have been tested. There is also an improvement in storing these models, by outputting in binary mode using `sgsim_pwb`.

The performance was tested showing that the new algorithm is slightly slower than the original version, because of the simulation of a coarse grid to improve variogram reproduction.

Some problems for larger models remain: *heap space exceeded* appeared a few times. This issue should be thoroughly investigated.