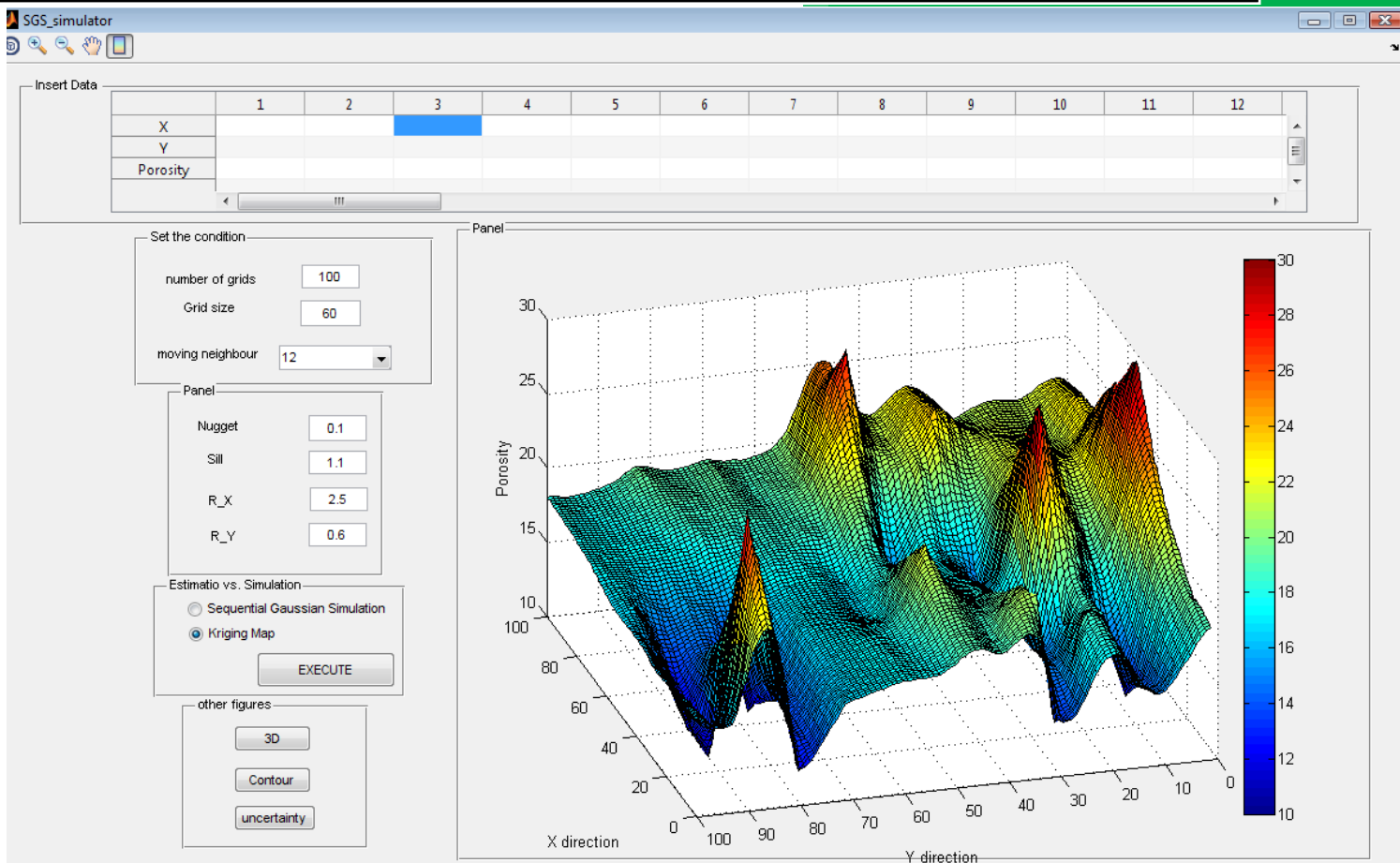


Geostatistical Project report winter 2010



Esmail Ansari

88200782

Supervised By: Dr. Masihi, Dr. Biniiaz
Term Project
Chemical and Petroleum department
Sharif University of Technology
jan.2010

CONTENTS

1. SGS algorithm	1
2. Normal score transform	5
3. SIS algorithm	7
4. Anisotropy models	10
5. SGS results	11
6. Spatial variogram	14
7. Zonal anisotropy flowchart	17
8. SIS results	18
9. conclusions	28
10. Appendix A	29
11. Appendix B	31

Abstract

The stochastic based models will be investigated. The Sequential Gaussian Simulation and Sequential Indicator Simulation will be studied and explained further. The 2D variogram for SIS program will be explored. The realizations of porosity and facies for geometrical and zonal anisotropy will be generated in two dimensions. Various results will be compared and discussed. Features of the programs will take into consideration.

Preface

All parts of the project have been performed by written codes in MATLAB software. The provided codes can handle other conditions as well.

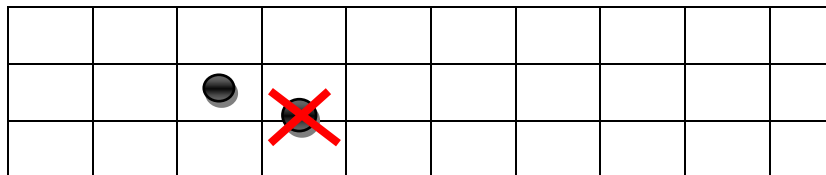
SGS Algorithm

We first quote the method from the course note and then we will explain it further.

- 1- Assign each cell a value equal to that at the nearest conditional cell.
- 2- Set a random path to visit each cell only once.
- 3- Set local neighborhood containing conditional data and previously simulated cells.
- 4- Perform OK at the cell to get the mean of Gaussian distribution Z^* and the variance σ_{OK}^2 (and hence CDF).
- 5- Using uniform random number $R \in (0,1)$ and this CDF to sample the simulated value at that cell.
- 6- Add this to the known data and set $I = I + 1$ and go to the next cell and repeat this visit for all cells. Note that considering the previously simulated values as available data results in reproduction of the covariance between all simulated values.
- 7- This produces a single realization. Then choose another random path and repeat. Note that the restriction on local neighborhood cannot preserve large scale trends.

These steps of course notes must be explained further in order to be clear. Here it is:

1. The first step means that known data must be assigned to the cells (grids) rather than any other location in our 2D plane.



2. The second step of procedure means that the turn of each point for estimation (being kriged) must be assigned to it randomly. No point has any kind of advantage over the other points for being kriged. So each point has a kriging number that simply shows its order for being kriged, and this turn will be assigned to it randomly, but we must be aware that two points don't have the same turn. This objective can be obtained very easily in Matlab using randperm function.

5	9	6
4	12	1
2	7	13
8	3	15
14	10	11

This figure shows random path. The black points are known points. All points will be labeled by a random turn number, the known points will be jumped for kriging.

- The third step means that all your known 2D points (original data + previously simulated points) cannot participate in the kriging matrix because it is very time consuming or maybe infeasible. Gslib says ***“We typically limit ourselves to the nearest 12-48 grid nodes.”***

Q. But why not more points?

A. Gslib says the reasons for having more points: it is theoretically better, more accurate estimate of the conditional mean and variance and better reproduce the variogram.

Q. So what is wrong with having more points?

A. Gslib says: CPU time is proportional to N^3 , memory requirements is proportional to N^2 , negative weights are commonly encountered when data are screened, using fewer data places less emphasis on the assumption of stationary. Let explain these lines further:

These lines say that if the number of points multiplies N then the time that is required for the simulation multiplies N^3 and the memory RAM required multiplies N^2 . Also, having more than 12-48 nearest known neighbor point is not that much required because of screening effect of near point on the distant point which causes that their weights to be negative rather than zero. In addition, as we have assumed that stationary exist in the original points; and hence in case we haven't subtracted the drift, using this number of points reduce the assumption of stationary that we assumed to exist for our data.

Q. So what number between 12 and 48 is a better choice?

A. this answer depends on the number of dimensions (2D or 3D) that you are working with, range of variogram relative to grid node spacing and CPU time restrictions.

4. Step four means that after we have selected the nearest points for kriging matrix, Ordinary kriging will be done on the target point; which results in an estimator Z^* and σ_{OK}^2 .

Then having these two criteria we can select a likelihood function that best represents our distribution.

5. In step five we select a random point from a normal distribution function. (U (0, 1))

The assumption of Gaussian for our likelihood function, in this step, obliges the Sequential Simulation to be Gaussian.

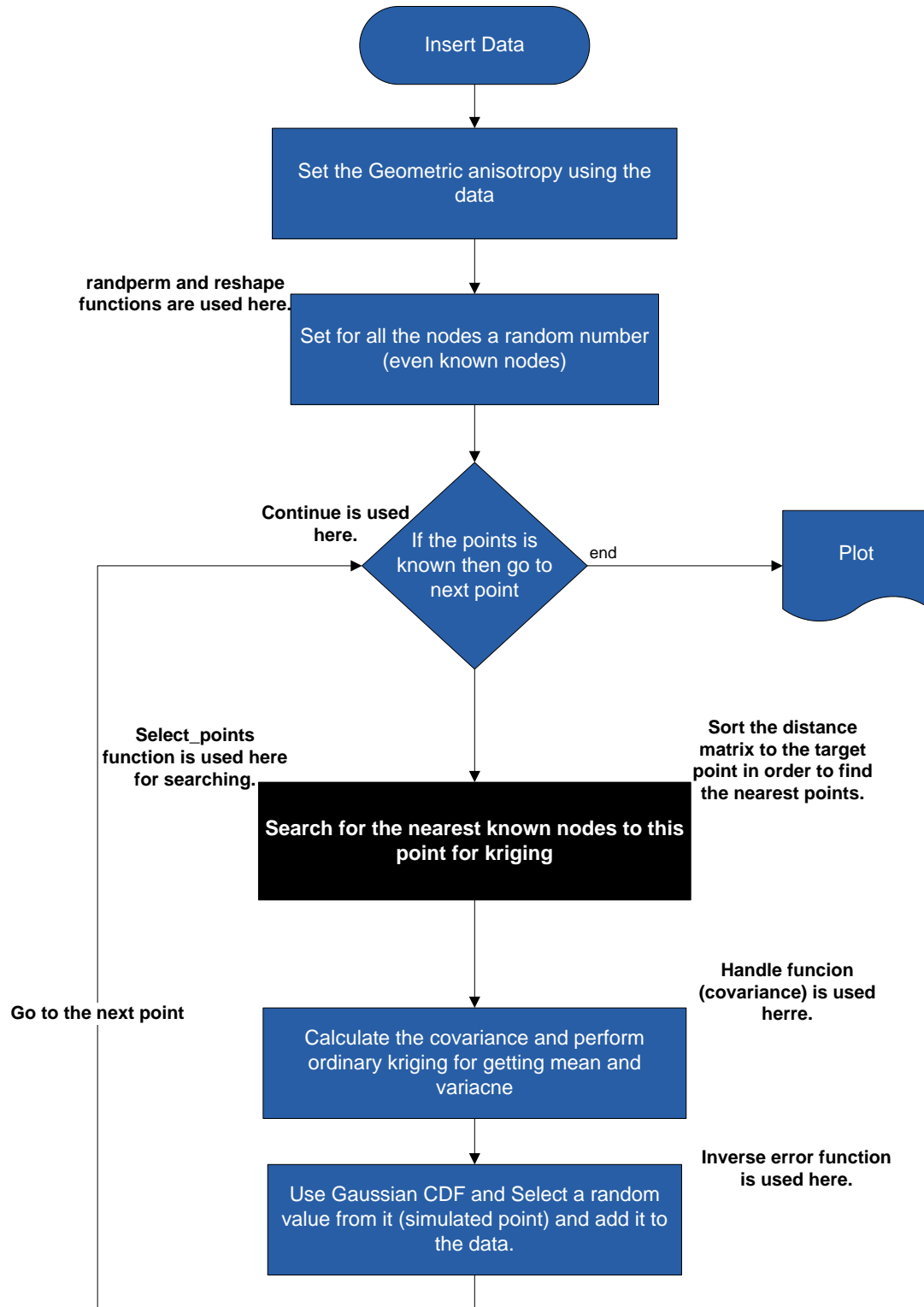
which simply is rand(1) in matlab and then using this random number and the Gaussian likelihood CDF obtained in the previous step, we sample a data. The formula for Gaussian CDF is given below.

$$z = -\sqrt{2} \operatorname{erf}^{-1}[1 - 2 F(z)]$$

$$Z = \frac{z - \mu}{\sigma_{OK}}$$

$$\mu = z^*$$

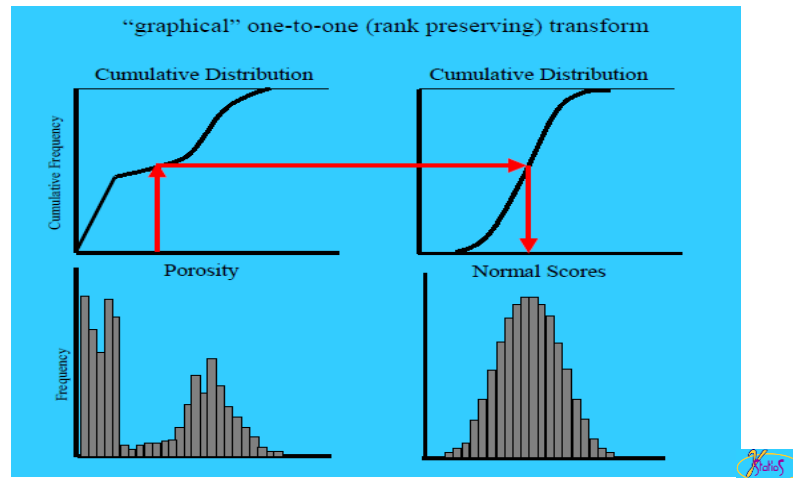
6. Step 6 And 7 is clear.



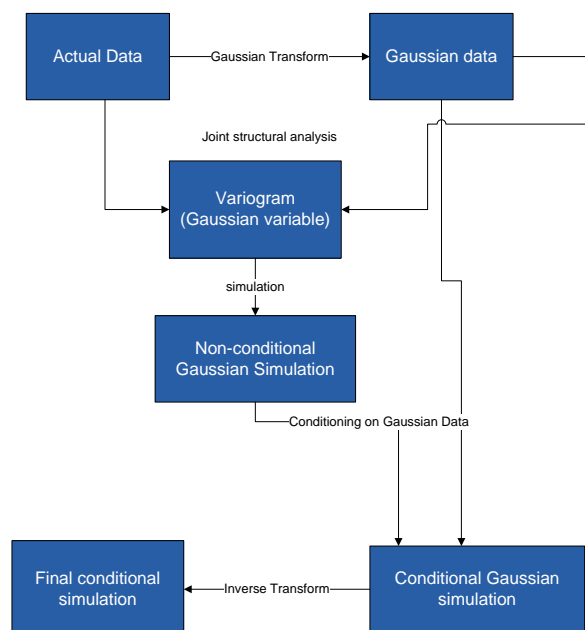
The flowchart of SGS program and the functions used in it.

Normal score transform

The idea is to transform the CDF of the variable to be Gaussian in order to use SGS for the simulation of that variable.

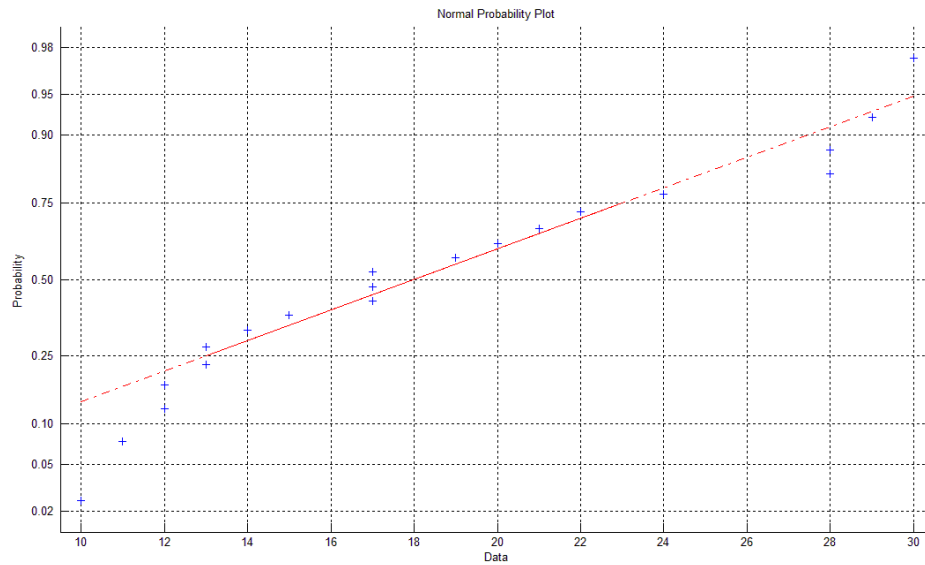


This figure clearly indicates how we can transform our data from a non-normal distribution to Gaussian distribution. (This procedure is called normal score transformation).



This chart shows the transform and inverse transform of data for SGS.

In the **SGS project**, this step was not considered because the normal plot of porosity showed a normal distribution.



This figure is distribution of porosity in 2D. Over the surface the porosity shows a normal behavior.

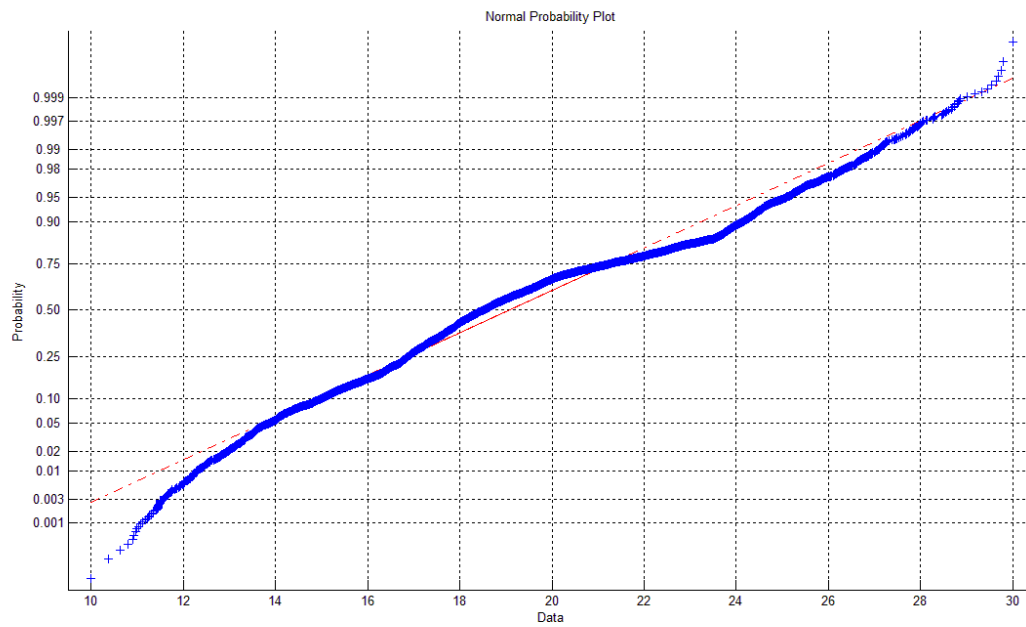


Figure above is drawn after simulation of all the 2D points. The behavior, as we expected to be, is normal.

SIS algorithm

We begin by assuming that the spatial distribution of facies categories is mutually exclusive, that is, two facies cannot exist at the same cell.

These lines are from course notes that will be explained further.

1. Transform given variable to an indicator.

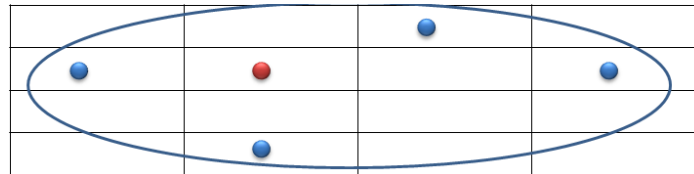
$$\text{Indicator} = \begin{cases} 0 & K \leq 100 \\ 1 & 100 \leq K < 150 \\ 2 & K \geq 150 \end{cases}$$

2. Set a random path to visit each cell only once.

Like SGS I used “randperm”.

3. Set a pre-specified number of conditioning data and previously simulated cells for each cell.

I used number “4” for the nearest points that participate in the kriging. This number has been suggested in the GSLIB site for the SIS.

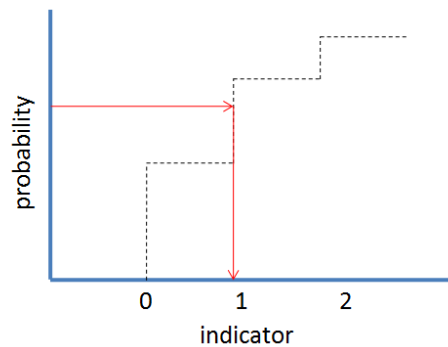


For this I normalize the data so their sum doesn't exceed unit, the negative values are replaced with zero and probability of indicators that are greater than one are set to be one.

4. At an unsimulated point form the kriging indicator estimate how much is the conditional probability of that indicator to get local CDF from the probabilities.

This can be done by using weights that is obtained from the kriging, so that in each case, we sum the weights of points that have the same permeability value. This gives the probability of that facie. In fact for calculating the probability of each lithotype, we are setting the value of that lithotype to be 1 and the others to be zero then multiplying this to the λ vecotr.

5. Use a uniform random $R \in (0,1)$ with the previously obtained the CDF to sample the simulated indicator value at that cell. Figure below shows the CDF of indictor.



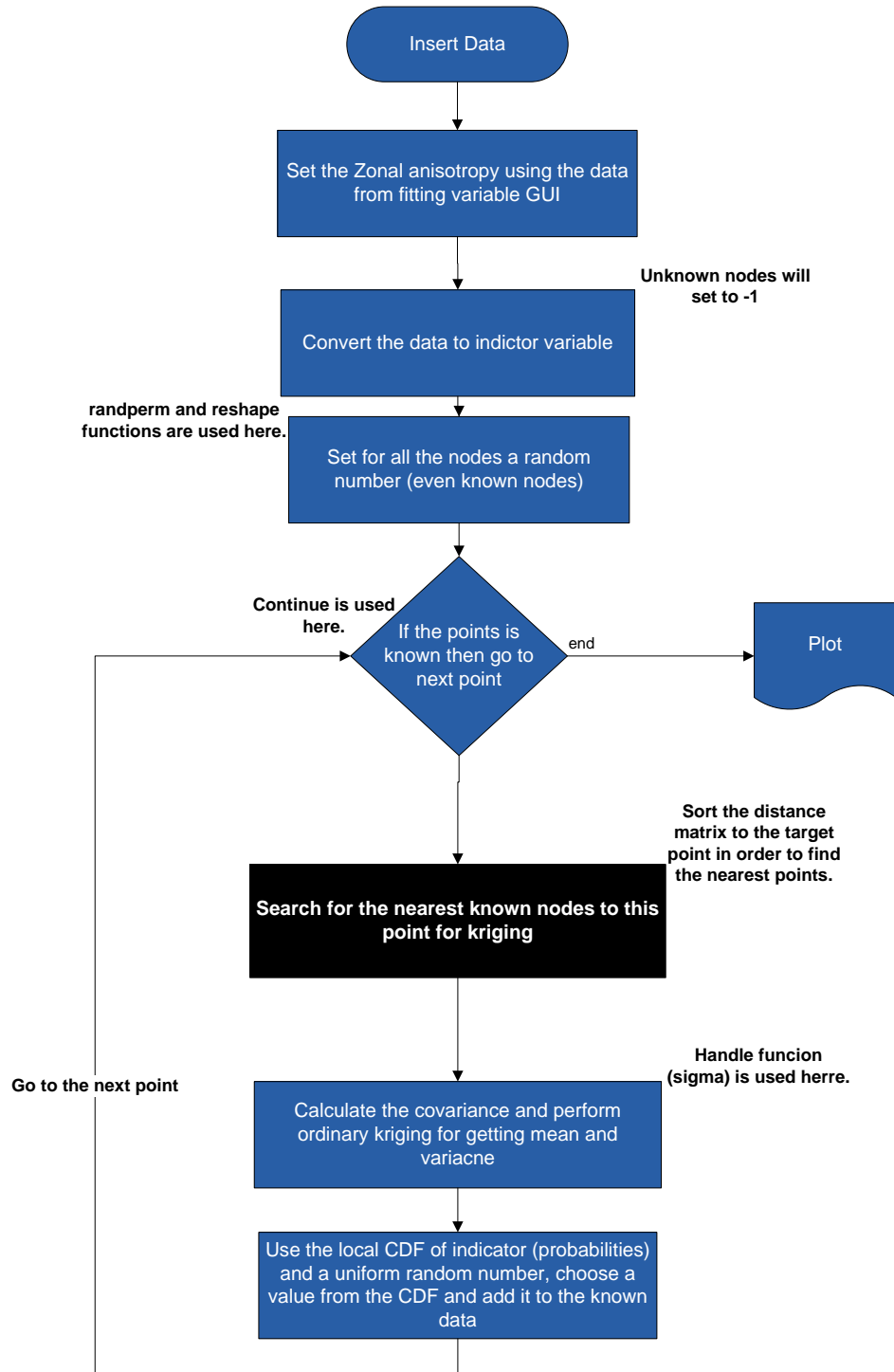
6. Add that value to the data and repeat this for each cell in the random path to generate a realization.

Notes on SIS

1. Probability of indicator may be slightly negative or greater than 1 so we must put all negative probabilities equal to zero and rescale probabilities so they sum to one.
2. Sum of the probabilities may not add to one so we must normalize the probabilities vector.
3. When you're assigning an indicator to the cell. You must assign the nearest value to the cell. (In the project, this condition exists).

Program main variables

The program starts by randomly assigning each node a turn (even the known points). ("*point_turn variable*"). The unknown points are zero in the SGS program and -1 in the SIS program ("*Plane variable*") then the first random point if it is unknown will be selected for kriging. For kriging of this point the nearest points to it will be considered by calculating all the known points distance to that point and then sorting the distances for finding the index of nearest points. ("*distance_matrix variable*"). For the variables we have chosen long indentifying names so I don't describe them here. Note that the index variables referrers to the index of points.



The flowchart of SIS program and the functions used in it.

Anisotropy models of the programs

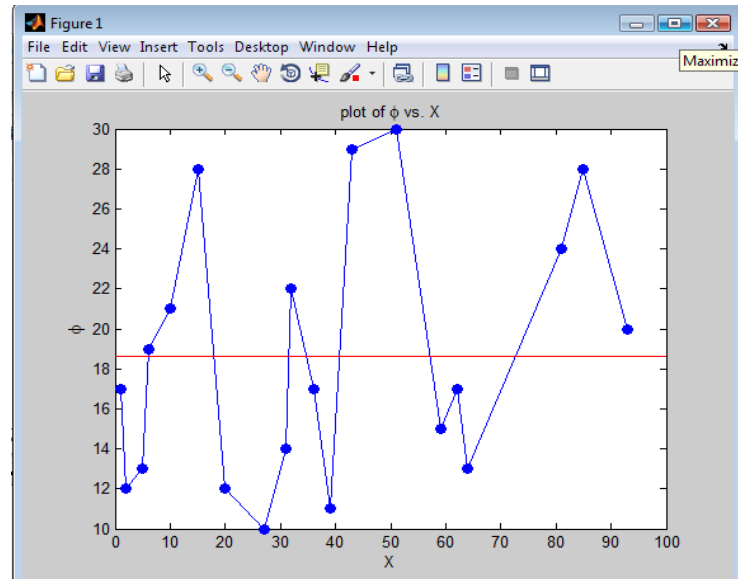
The starts assuming you already know the Spherical model parameters of the anisotropy (whether it is geometric or zonal). In case of zonal anisotropy the variogram will be set the sum of geometric anisotropy and vertical anisotropy.

geometric anisotropy (SGS)	$h = (h_x^2 + k^2 * h_y^2)^{.5}$	$k = \frac{R_x}{R_y} \geq 1$
$cov(h) = sill_x - ((sill_x - nugget_x) * \left(\frac{3}{2} \frac{h}{R_x} - \frac{1}{2} \left(\frac{h}{R_x} \right)^3 \right) + nugget_x) \quad h < R_x$ $cov(h) = 0 \quad otherwise$		

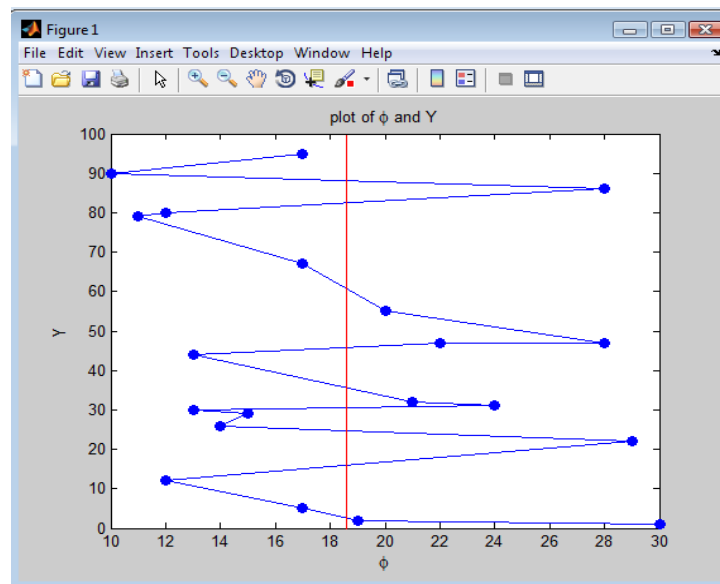
zonal anisotropic (SIS)
$variogram_{vertical} = (sill_y - nugget_y) * \left(\frac{3}{2} \frac{h}{R_y} - \frac{1}{2} \left(\frac{h}{R_y} \right)^3 \right) + nugget_y \quad h > R_y$
$variogram(h, h_y) = variogram_{Geometric}(h) * + variogram_{vertical}(h_y)$

SGS results

I proved in the first part that the distribution of porosity for data on page 157 is Gaussian and does not require Normal Score Transform. The other thing that we must investigate is stationary.

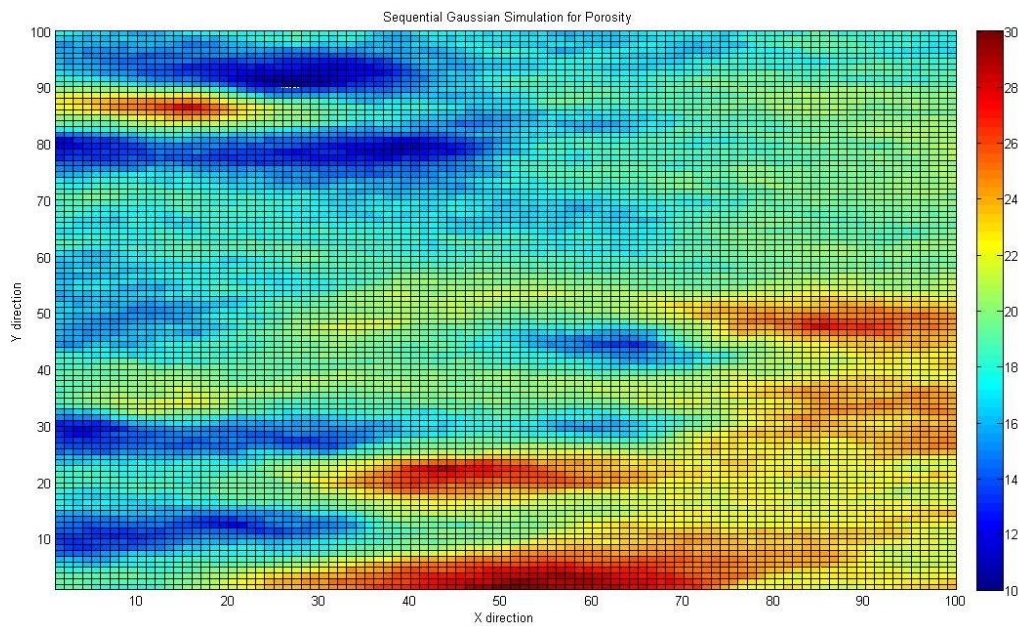
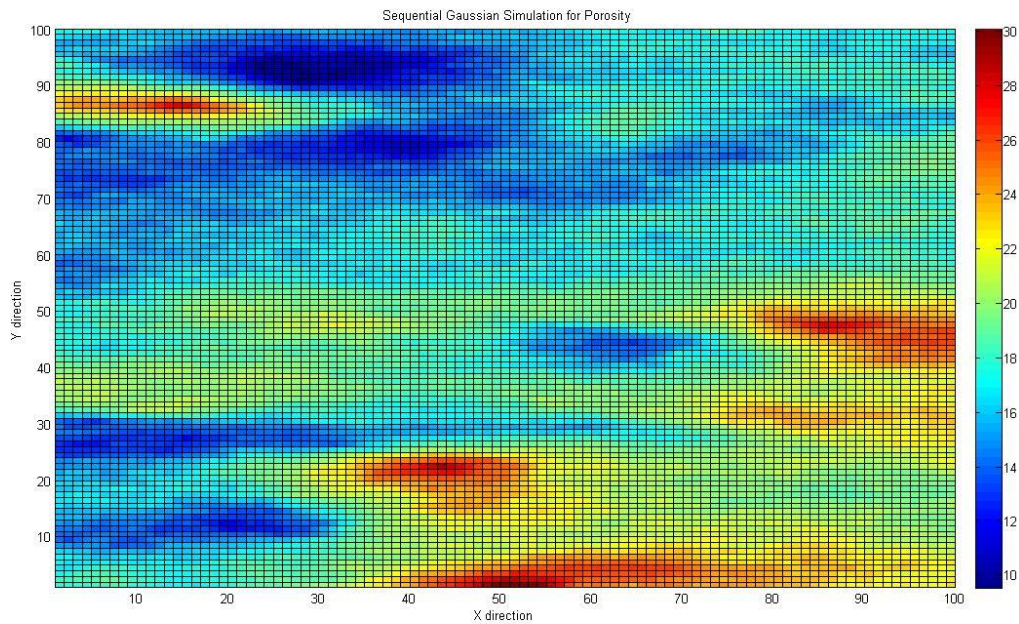


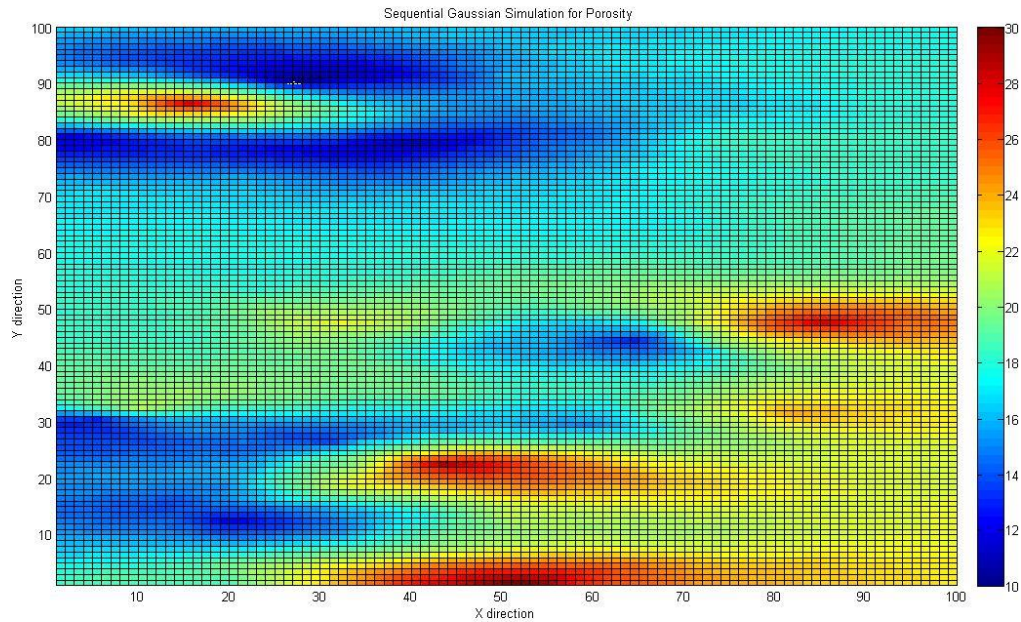
stationary along X direction



Stationary along Y direction

We set the near neighbor points to be 20 and then we run the program. The time of running for 20 points is about 18 seconds. Two realizations and one kriging map for the sample of page 157 is shown below. Comparing these images we can find out that estimation is a smoothing interpolator.





First figures show one realization and the third one shows the kriging map. The kriging map shows a smooth behavior.

And there is a GUI that is prepared for SGS and is very convenient to work with, note that for because the report will be very extensive. I refused to bring figures of uncertainty, 3D, contour map, increasing meshgrids while shifting the data, increasing the meshgrid and don't shifting the data, but

NOTE: There is a Flash file which shows all of these things, please do run it .

The second run in the flash file is for the following example.

Assume we have 32*32 meshgrid for the following distribution and stationary exists.

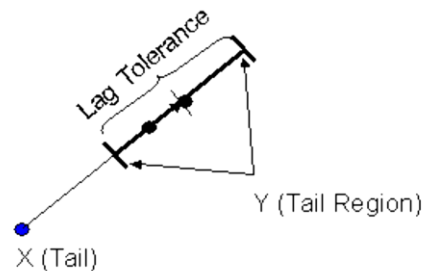
X	5	12	30	14	2	28
Y	3	20	4	14	9	28
Property	23	10	17	28	25	19

Spatial variogram

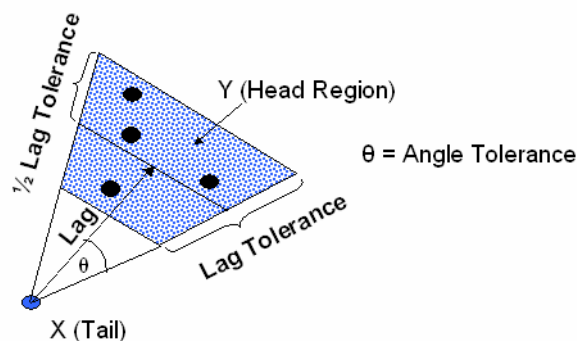
If data are spatially correlated, then on average, sample points that are close to each other are more alike than sample points further away. (More complex spatial correlations exist but this type is the most common).

Rarely in practice, will you ever have any sample points separated by exactly a lag distance h . Therefore, a lag tolerance centered about the lag distance will permit a capture of more data points in the calculation of $\gamma(h)$. In the figure below, all data points on the dark thick line area will be used.

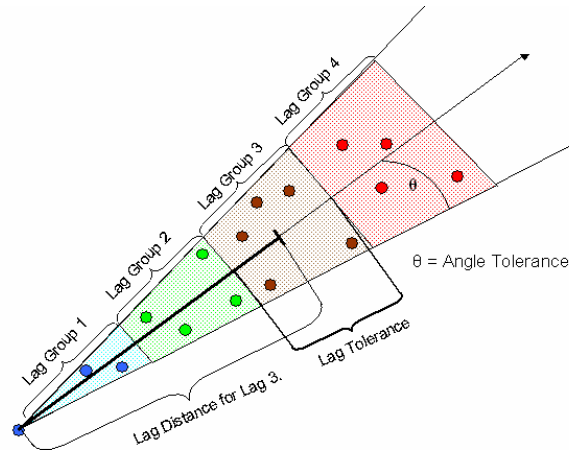
So if we are interested in the variance of all data points separated by 10 feet and we permit a lag tolerance of 2 feet. We will actually be calculating the variance of all pairs of data between 9 and 11 feet apart.



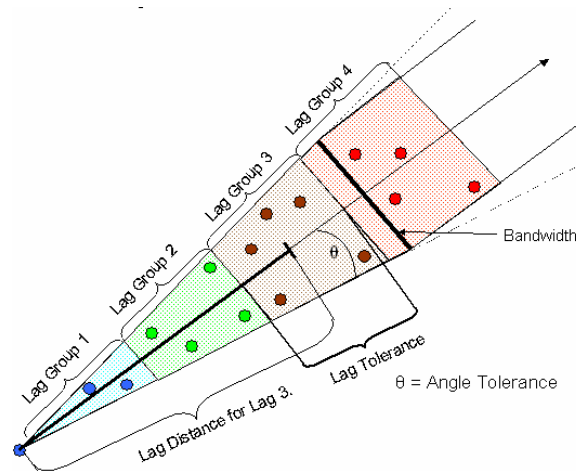
Although assigning a lag tolerance helps, most cases will never have enough samples separated by a lag - tol/2 to lag + tol/2 along a straight line to calculate the semivariogram value. Therefore, an angle tolerance, θ , is also introduced to expand the region and to include more points in the calculation of the semivariogram value for the specified lag distance. In the figure below, all data points within the blue shaded area will be used.



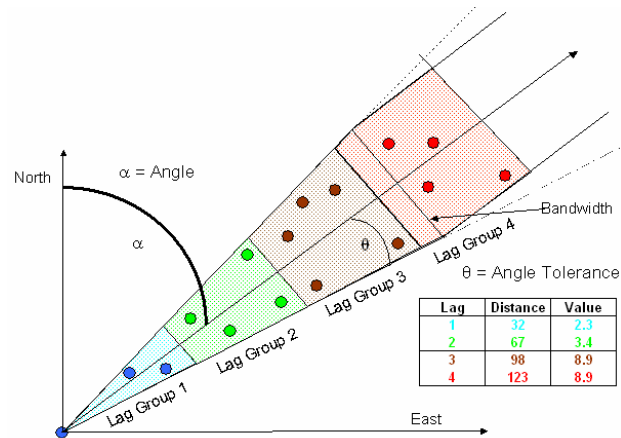
If we repeat this operation for a number of lag distances, we would generate a cone shaped object expanding outward from the point of interest. This cone would be partitioned by lag groups centered about our lag distances.



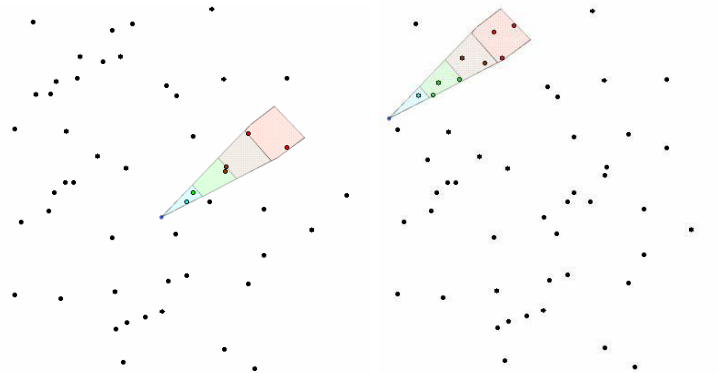
As the cone stretches farther out, it opens up increasingly wide, capturing more and more data points as it moves away. In practice, geostatisticians will often apply a constraint called the bandwidth. This bandwidth limits the expansion of the cone to a certain width. If you do not wish to constrain the cone's expansion, specify a very large bandwidth.



Our final parameter is the angle. The angle specifies in what direction you will be calculating the semi-variogram values. This is sometimes referred to as the angle of anisotropy. Now we are constraining our semi-variogram values to a certain direction.



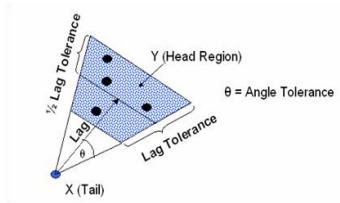
The semi-variogram cone is relocated at every sample location to build up the total set of pairs for each lag bin.



In fact, if anisotropic conditions exist, the direction of highest correlation is considered the major direction of anisotropy. The perpendicular direction is referred to as the minor direction of anisotropy. The major direction of correlation will exhibit semi-variogram values that increase at a slower rate than any other direction.

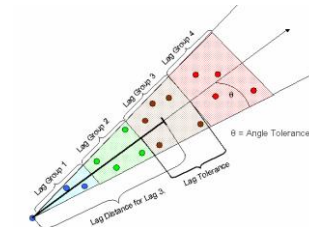
Flowchart of Zonal anisotropy variogram

Criteria =
 Angel between points < specified angle + angle
 tolerance
 Angle > angle-angle tolerance



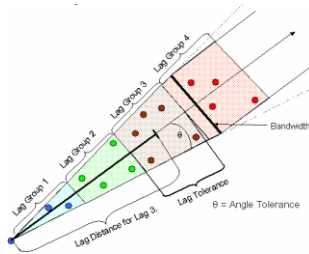
Select all the points that are in the cone
 (without bandwidth)

The first point will be
 stored in the points1
 and it's pair will be
 stored in the points2



For calculating each lag variogram, check all
 the distances between point1 and it's pair (in
 point2).

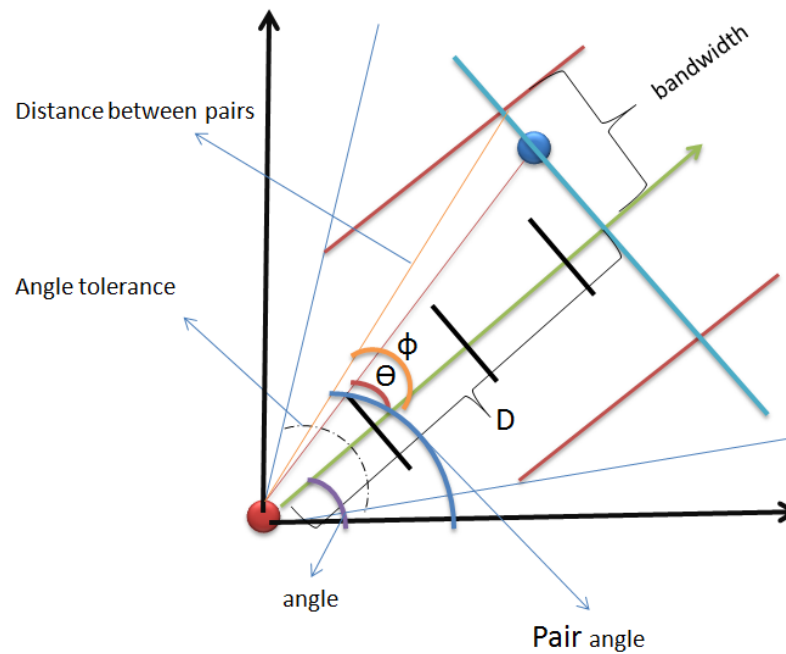
Criteria =
 Distance > lag-lag tolerance
 Distance < lag + lag tolerance



Exclude the points that are beyond bandwidth
 for each lag after chosen lag for bandwidth.

The starting lag for
 bandwidth and the
 bandwidth length
 should be specified.

Criteria for the bandwidth are explained below.



After a specific lag, we check that our pair of points that fall into that lag also have another criterion in order to participate in variogram calculation. From above diagram We always have $\phi = \tan^{-1} \left(\frac{\text{bandwidth}}{D} \right)$ and $D = \text{distance between pairs} * \cos(\text{pair angle} - \text{angle})$ and as $\theta = \text{pair angle} - \text{angle}$ so we can check for the bandwidth every time we including pairs in the variogram.

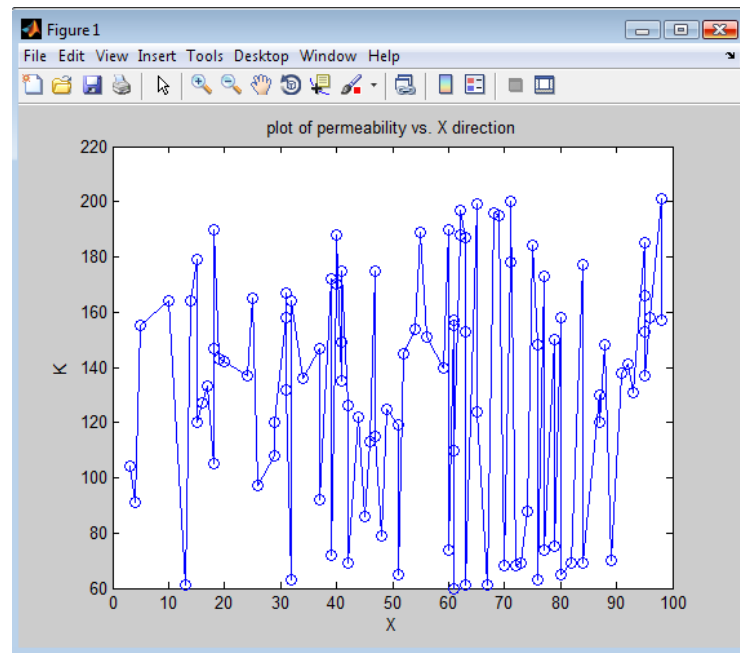
SIS results

After that below assumptions for finding the appropriate main directions were made.

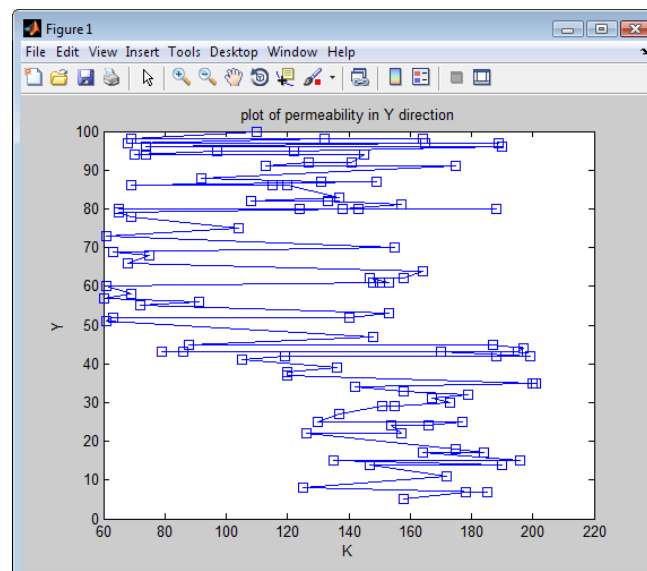
<i>cut off 1 = 100</i>	<i>bandwidth = 2</i>	<i>lag = 4</i>
<i>cut off 2 = 150</i>	<i>starting bandwidth = 5 lag</i>	<i>lag tolerance = 2</i>

The other assumption that I made for calculating variogram are below but you can change these assumptions and you will see that the variogram models and realizations that you will get at the end are very similar.

1. We assume stationary; it means that we have no trends.



The stationary of permeability exists in x direction.



Stationary exist in Y direction up to some lags. (This causes our distribution to have zonal anisotropy).

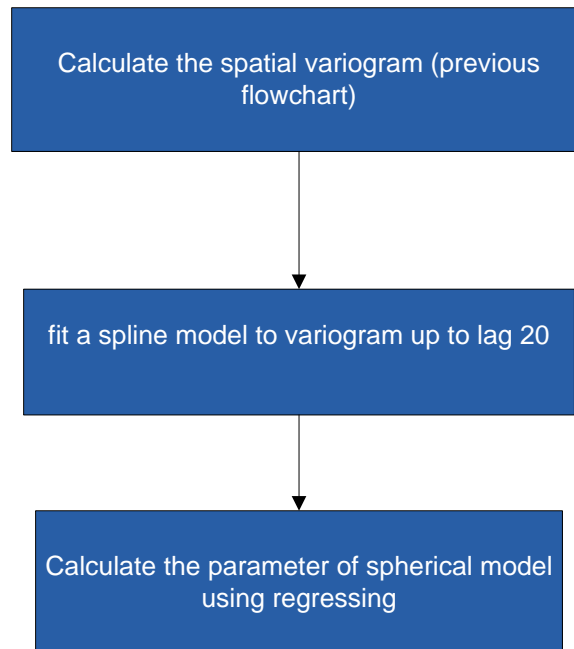
2. We fit the variogram to the near points (lags) assuming far points have no correlation.

(Note that without this assumption the realizations that you would get are very identical because as the R_x changes R_y also changes and other properties like nugget and sill will change as well.)

3. We fit a spline model to the nearest lags to find out how they behave.
4. We fit our spherical model to this spline that because it better predicts the behavior of near lags.

Q) But how can we find the main direction where we have the maximum range?

A) In order to obtain this objective, I wrote a code that fits the Spherical variogram model to the spline model that is fitted to the lags. Here is the flowchart:



The regression model is $Y = A + B X - C X^3$ with the following parameters.

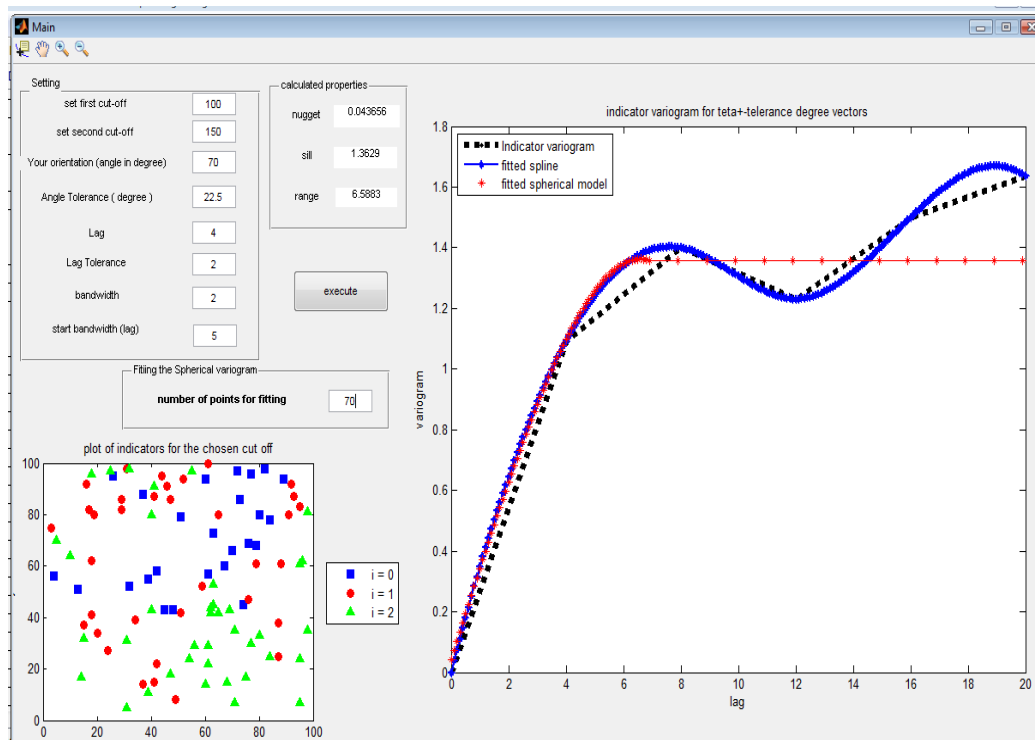
$$A = \text{nugget}$$

$$B = \frac{3}{2} \frac{(\text{sill} - \text{nugget})}{R}$$

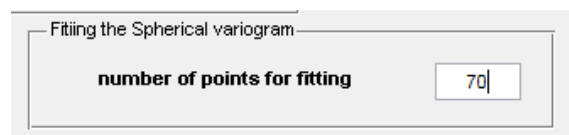
$$C = \frac{1}{2} \left(\frac{\text{sill} - \text{nugget}}{R} \right)^3$$

We can very easily find formula of sill and R explicitly by dividing B^3/C .

Let see the program and explain it visually:

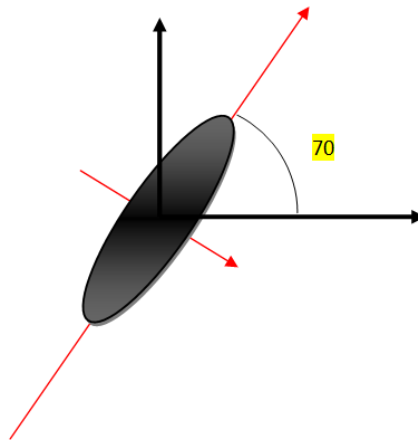


This figure shows the fitted spherical to the assumed conditions at $\theta = 70$, you can get the range, sill and nugget for other direction as we by fitting the spherical variogram.



In any direction that you want to fit the variogram, only change this number until you fit the variogram to that angle.

For the condition that I mentioned, we can find the range and sill in the vertical and horizontal. Note that the major direction was 70 degree (from the origin clockwise) and because of that the minor direction is clearly -20 which you can also find by checking the minimum range of all directions.

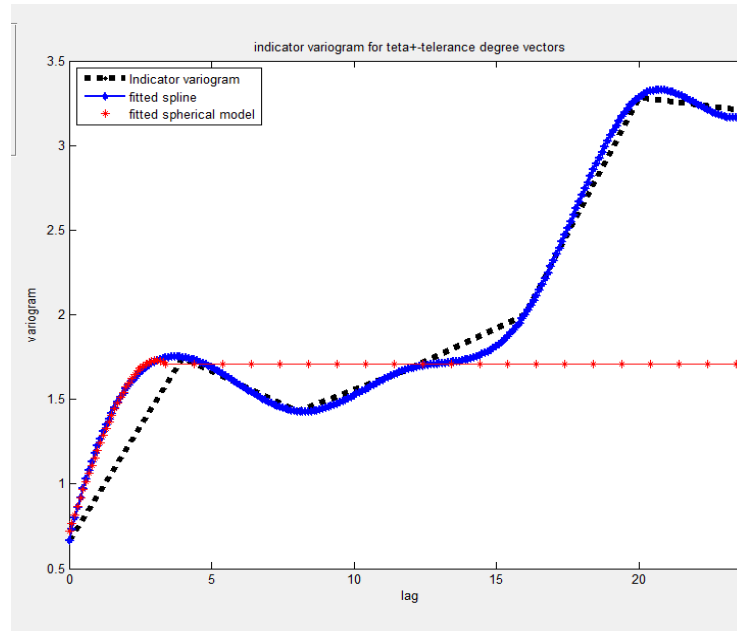


My main directions were found to be about 70 degree and -20 degree. For the sake of convenience using matlab, our coordinates are anticlockwise from the horizontal axis. (Unlike the tendency of getting the vertical axis and clockwise as the basis coordinates.)

Setting	
set first cut-off	100
set second cut-off	150
Your orientation (angle in degree)	70
Angle Tolerance (degree)	22.5
Lag	4
Lag Tolerance	2
bandwidth	2
start bandwidth (lag)	5

You can enter your condition here, but due to lack of point in some lags, it is better to choose your condition wisely that at least starting lags have points to be calculated.

Now by setting the number of points to 35, we fit a spherical model to -20 degree orientation.



Note that here we have assumed that correlation only exist to lag 10 (35 points) for all the figures which is not a realistic assumption to find the main directions. You can fit the spherical variogram to more lags if you wish.

After that we have get some values for variogram models then we can run the SIS program. Note that I fitted the variogram to the first 10 lags and also to more than 50 lag and the results were as follows. You can do it yourself using the GUI very easily. The major direction is set to be 70.

Number of points for fitting for 70 and -20 degree are respectively 70 and 35. This models is fitted to 7 lags and is less reliable.					
$sill_x$	$sill_y$	R_x	R_y	$nugget_x$	$nugget_y$
1.7258	1.3629	3.0632	6.5883	0.71804	0.043656
Numbr of points for 70 and -20 is set to be 550 and 420. This meodel is fitted to more than 30 lags and is more reliable.					
3.8754	3.876	37.1256	59.9668	0.70107	0.38754

The previous figures were for the first model of variogram of table and the following figures are for the second model.

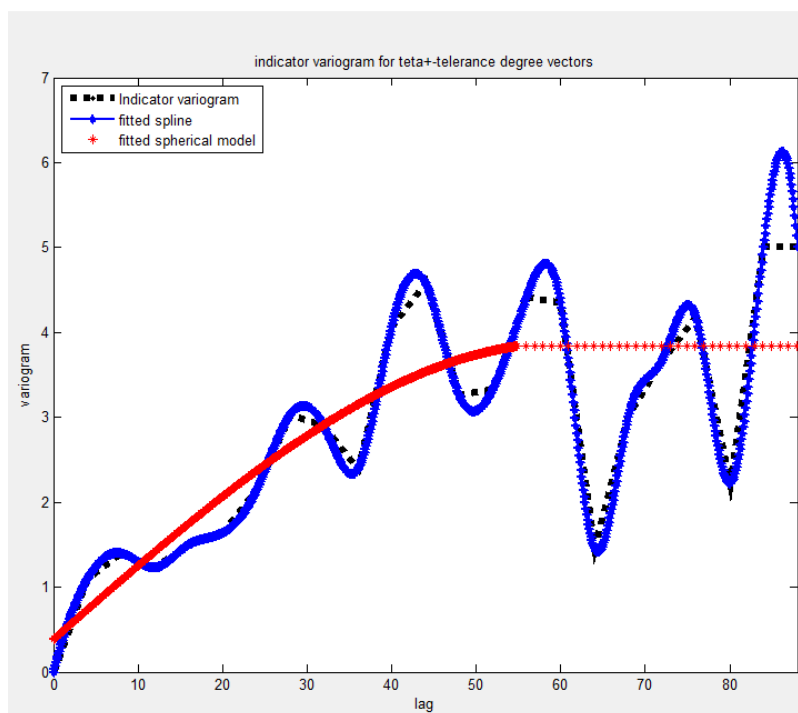


Figure shows the fitted spherical model up to lag 55 for orientation 70 degree. (number of points for regression is set to be 550).

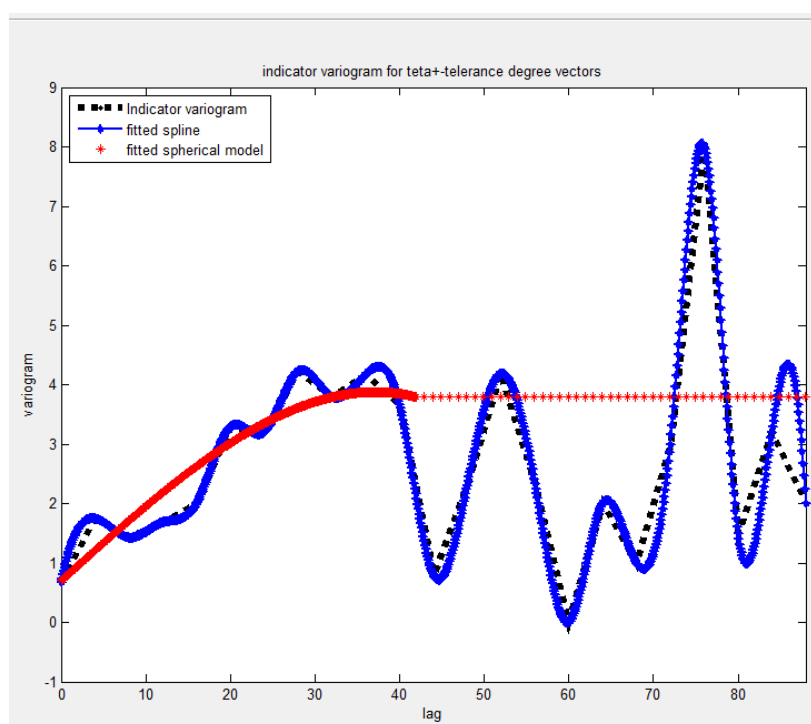
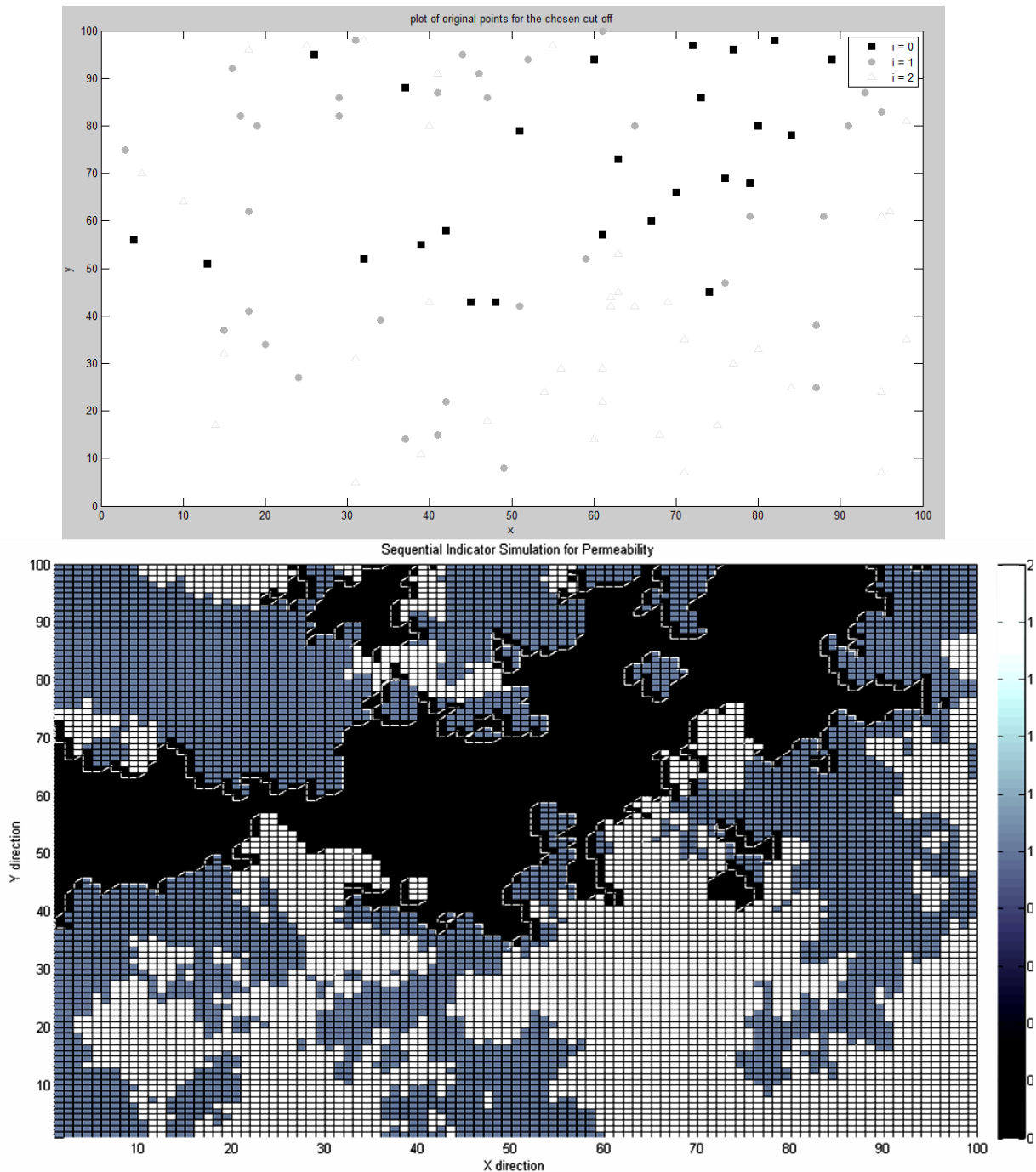
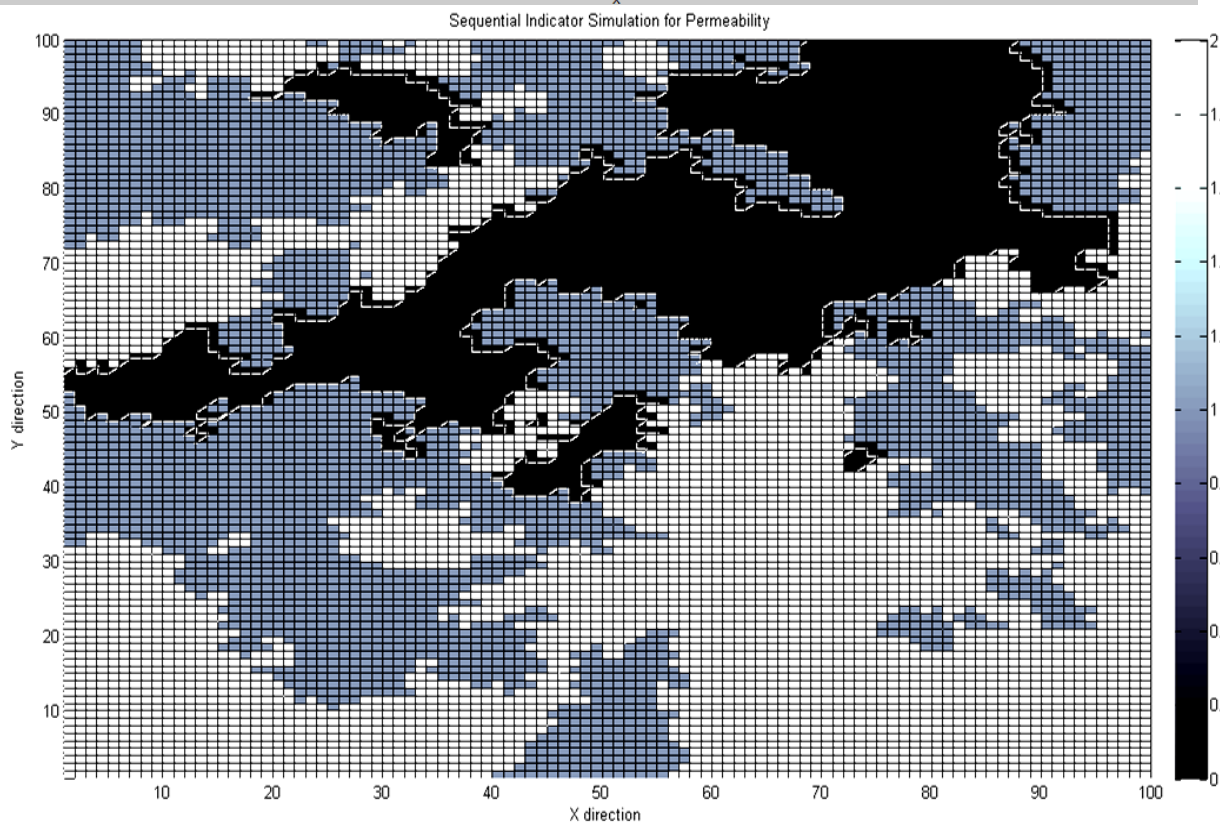
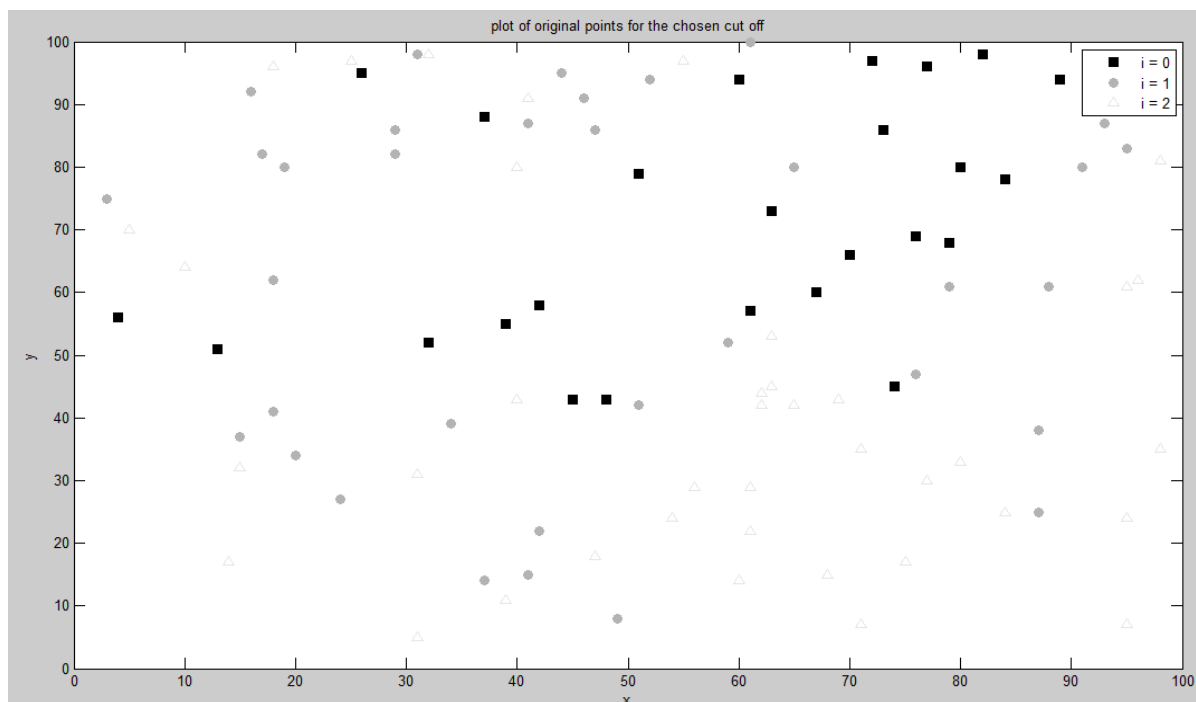


Figure above shows the fitted spherical up to lag 42 for the -20 degree orientation. (number of points used for regression is set to 420)

Now that we have some numbers for the zonal anisotropy, we can run the SIS codes more confidently. The results were as below. **Run time of SIS program is about 15 seconds.**



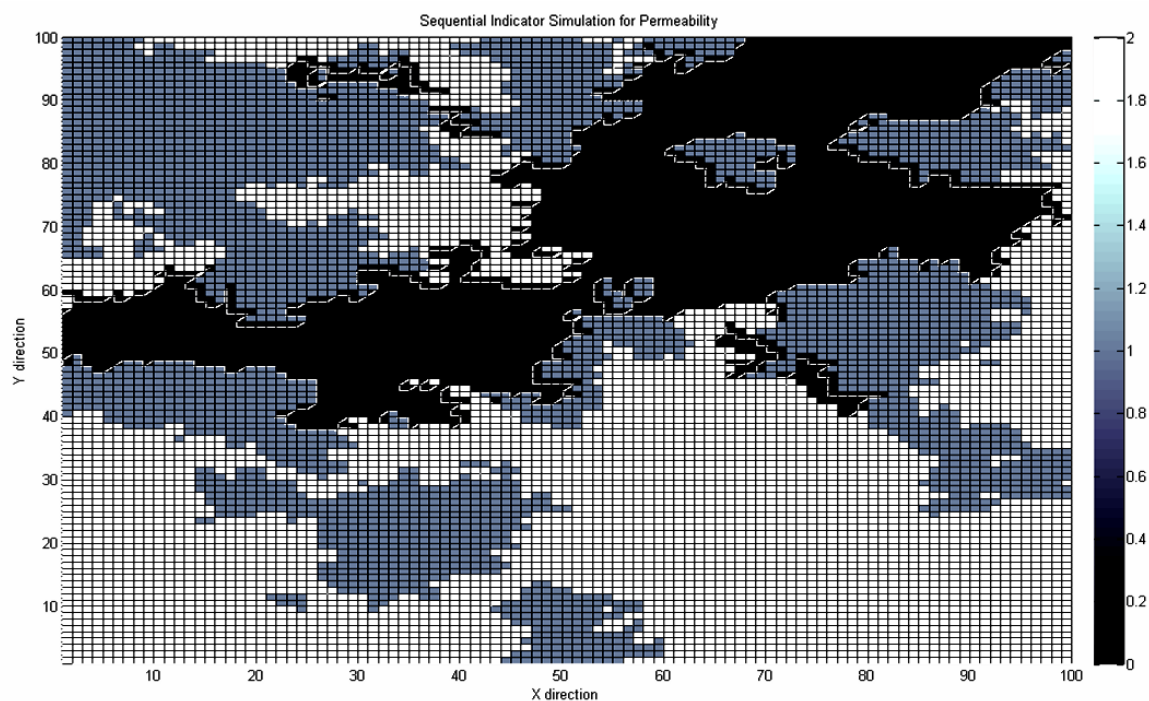
SIS for the first spherical model which is less reliable.
 Before simulation $P(0) = 0.25$ $P(1) = 0.35$ $P(2) = 0.4$
 After simulation $P(0) = 0.2696$ $P(1) = 0.3841$ $P(2) = 0.3463$



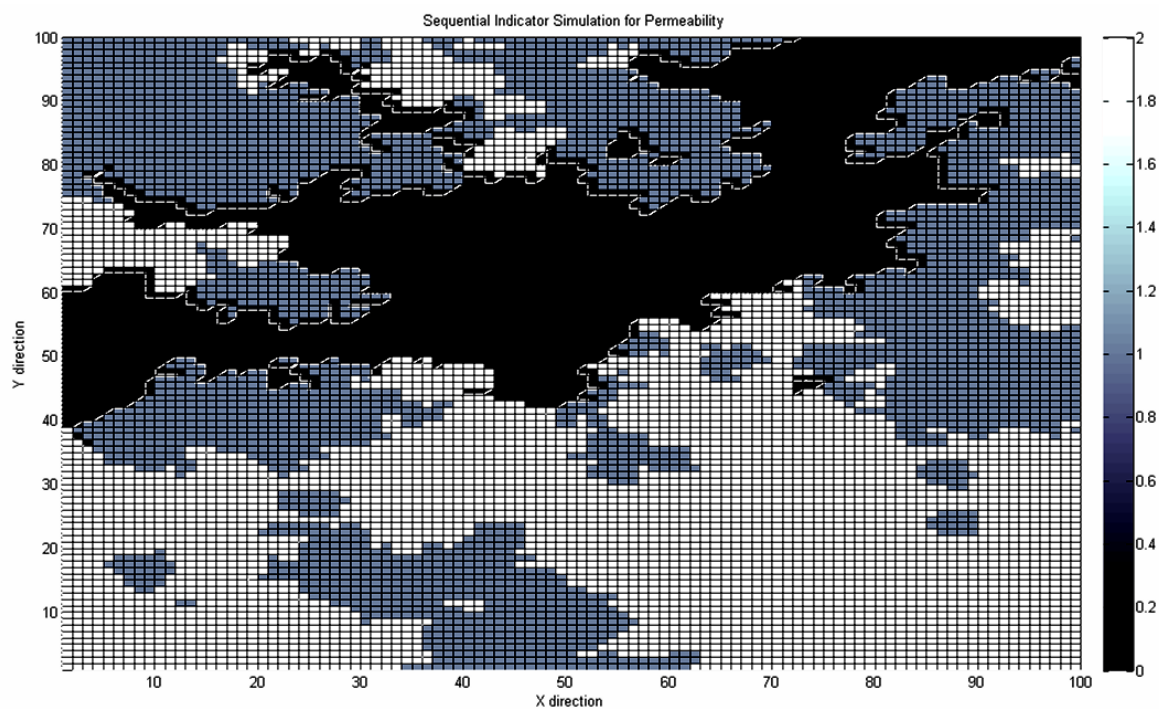
SIS for the second spherical model which is more reliable.

Before simulation $P(0) = 0.25$ $P(1) = 0.35$ $P(2) = 0.4$

After simulation $P(0) = 0.2264$ $P(1) = 0.3595$ $P(2) = 0.4141$



SIS for the second spherical model which is more reliable.
Before simulation $P(0) = 0.25$ $P(1) = 0.35$ $P(2) = 0.4$
After simulation $P(0) = 0.2556$ $P(1) = 0.3129$ $P(2) = 0.4315$



SIS for the second spherical model which is more reliable.

Before simulation $P(0) = 0.25$ $P(1) = 0.35$ $P(2) = 0.4$
After simulation $P(0) = 0.2630$ $P(1) = 0.3388$ $P(2) = 0.3982$

C Conclusions

1. Stochastic modeling can be use for uncertainty assessment, as they generate many realizations that might occur in the nature.
2. Sequential Gaussian Simulation is popular because it is very convenient to implement.
3. The prerequisite to SGS is to transform data to be Gaussian. This can be done by applying normal score transform and after simulation a back transformation is applied.
4. Cells far from the conditioning points can take values with the same probability over the full range of CDF; this means that SGS maximize the uncertainty.
5. If we increase the grids over an area, the image that we get would be smoother.
6. The path for the Simulation must be selected randomly to avoid artifacts.
7. Selecting a value from the CDF must be done uniformly.
8. In order to model variogram, we must take bandwidth into consideration.
9. Variogram models in 2D require some assumptions. (Angle tolerance, bandwidth starting lag, bandwidth, lag tolerance.)
10. For SGS, 12 to 48 near neighbor points to the kriging point is enough. (It is impractical to use all the points for kriging and also this is not necessary).
11. For SIS the amount of neighbor points can be minimized to 4.
12. Indicator kriging can handle any distribution (not limited to Gaussian) and provides local CDF.
13. The probabilities in the SIS may be negative or sum to more than one which should be modified.

APPENDIX A

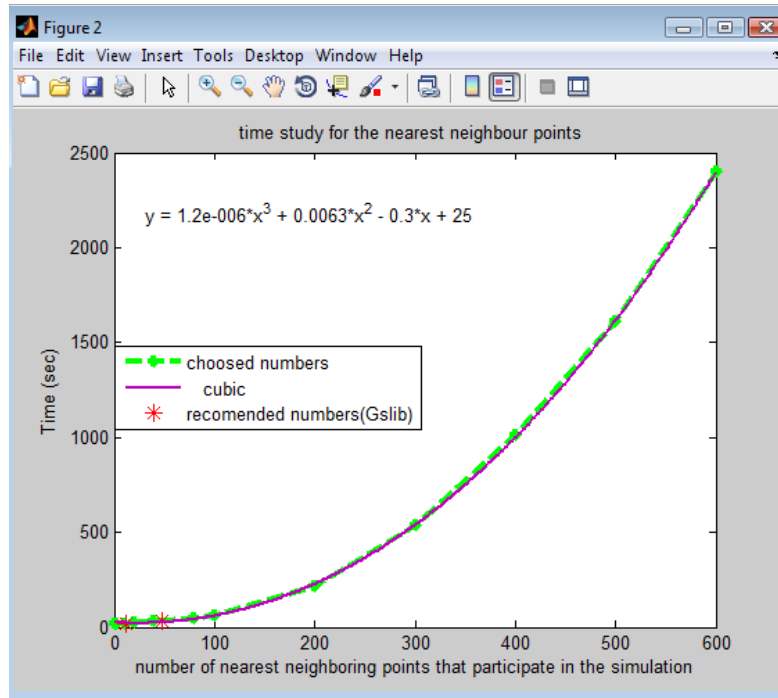
Let's investigate the following claim from GSLIB by plotting the time measured for various neighboring points. Time has measured before optimization.

CPU time is proportional to N^3



The following table shows the time required for each amount of neighboring points.

other conditions are set to the default of page 157			
points_number		time(sec)	
1		19.9208	
2		19.9735	
10		20.6574	
20		23.5254	
40		29.6276	
80		49.6806	
100		64.5455	
200		214.772	
300		537.083	
400		1015.85	
500		1610.07	
600		2401.12	
1000		?	explanation in the report
recommended points from Deutsch (Gslib site)			
12		20.7722	
48		31.5823	



By plotting points_number (which refers to the neighbor points) we can verify the GSLIB claim.

Q) Why is there more than known points in staring the program? (We have 20 known points initially).

A) The algorithm prepared for the SGS and SIS can handle more than initial known points by assigning each unknown points a far distant (10^{10} or NAN) which cause them to have a zero sigma by being too far from the kriging point, as the known points points increases, they are being considered in the kriging equations.

Q) Why we haven't measured the 1000 neighbor point's time?

A) The answer is that we can estimate the time very easily using the equation in the above figure which is ($1e-6 \cdot 1000^3 + 0.0063 \cdot 1000^2 - 0.3 \cdot 1000 + 25$ seconds) or **117.0833 minutes**. I should mention that no software uses 1000 near point for kriging nor it is necessary. (I've explained the reason before).

APPENDIX B

Improving the speed

Here we speak about how we can optimize our work in Matlab in order to have a good speed. For gaining this objective fortunately Matlab offers a lot of features:

1. The first thing we can do is very well known in Matlab and just about anybody knows it and It is using smart arrays properties which will be implemented very easily by using “.” before “+ - * /”.
2. The second thing we can do is that we should eliminate “**For**” loops and use array properties as much as we can.

The commonly used and popular Functions for this aim are “**reshape**”, “**meshgrid**” “**sort**”, “**find**”. Note that these functions are very speedy because they are build-in functions in matlab.

```
33      %% labeling the points with their turns (for setting the random path)
34 -    point_turn = randperm(total_nodes);
35 -    point_turn = reshape(point_turn,X_blocks,Y_blocks);
36
```

This reshapes the matrix of turn point in order to assign a turn to each node,
The meshgrid function relates each two points of two vectors. This gives a very good alternative for eliminating two “**for**” which is more time consuming.

3. To use logic properties instead of “**if**”.

For example in the project I implemented this as below to increase the speed.







```
30  %% covaiance(h) = variance - variogram
31 - sigma = @ (h) ((sill - ((sill-nugget) * (1.5*h/R_x - .5*(h/R_x).^3) + nugget)) .* (h < R_x));
32  |
```

For wiring multiple value functions we can use logical properties which are faster. This line means that







$$\text{cov}(h) = \begin{cases} \text{fitted model} & h < R \\ 0 & h \geq R \end{cases}$$

4. The other thing that we can do in order to have a high speed is that we pre-allocate the arrays so they don't have dynamic sizes. (Their size should be specified from beginning).
5. And maybe the most effective way is to use “**Profiling**”. **Note** that when you are profiling the code, the time will increase tremendously so that can asses which lines take more time, for example the time required time for the plain work (folder SGS) without profiling is 16 seconds and when you are profiling you must wait for 116 seconds.







```
Command Window
fx >> profile on % turns on the profile
profile clear % clears any previous information
main; % runs m.file
profreport % brings up the profile report
```

Line Number	Code	Calls	Total Time	% Time	Time Plot
13	<code>if Plane(i,j)~=0</code>	99800000	25.999 s	22.6%	
18	<code>end</code>	99800000	23.808 s	20.7%	
14	<code>distance_matrix(i,j) = ((i-kri...</code>	49994810	13.682 s	11.9%	
23	<code>[dummy_sorted_index] = sort(di...</code>	9980	13.278 s	11.6%	
16	<code>distance_matrix(i,j) = nan; % ...</code>	49805190	12.249 s	10.7%	
All other lines			25.787 s	22.5%	
Totals			114.805 s	100%	

The result of profiling for 100 * 100 block (about 16 seconds without profiling, the red ellipse shows just the percentages can be useful). This figure demonstrates that replacing meshgrid function instead of two for will be helpful.

Line Number	Code	Calls	Total Time	% Time	Time Plot
61	<code>[dummy_sorted_index] = sort(di...</code>	9980	14.547 s	41.0%	
57	<code>distance_matrix = ((IY_BLOCKS-...</code>	9980	7.292 s	20.6%	
120	<code>contour_handle=clabel(c,h);</code>	1	5.101 s	14.4%	
56	<code>[IX_BLOCKS IY_BLOCKS]=meshgrid...</code>	9980	1.008 s	2.8%	
80	<code>kriging_matrix = sigma(h);</code>	9980	0.925 s	2.6%	
All other lines			6.584 s	18.6%	
Totals			35.456 s	100%	

The result of profiling for 100 * 100 block (about 20 seconds without profiling). This figure shows that the sort function that we have used in our program takes more time than other functions. This figure reveals the bottlenecks of our program. Line 120 is for contour map.

Line Number	Code	Calls	Total Time	% Time	Time Plot
61	[dummy sorted_index] = sort(di...	39980	187.778 s	56.3%	
57	distance_matrix = ((IY_BLOCKS-...	39980	97.155 s	29.1%	
56	[IX_BLOCKS IY_BLOCKS]=meshgrid...	39980	18.599 s	5.6%	
120	contour_handle=clabel(c,h);	1	8.190 s	2.5%	
51	distance_matrix = zeros(Y_bloc...	39980	3.329 s	1.0%	
All other lines			18.326 s	5.5%	
Totals			333.377 s	100%	

Result of profiling for 200 * 200 mesh grid. Comparing this figure with the previous on; we can easily notice that if our grids multiply by 2 the time will increase severely. The primary reason is the sort function.

```

0.01  9980  11      for i=1:X_blocks
0.45  998000 12          for j=1:Y_blocks
26.00 99800000 13              if Plane(i,j)~=0
13.68 49994810 14                  distance_matrix(i,j) = ((i-kriging_point_index(1))^2+(j-kriging_point_index(2))^2)^.5;
11.97 49805190 15                  else
12.25 49805190 16                      distance_matrix(i,j) = nan; % nan will be assigned to the distance of zero nodes as we want them not to
11.79 49805190 17                      end % participate in the kriging.
23.81 99800000 18              end
0.26  998000 19          end
20
21          %% here we sort all the distances to select the nearest points to our
22          % target point, ( number of nearest points = points_number).
13.28  9980  23      [dummy sorted_index] = sort(distance_matrix(:));
-----

```

Figure above shows that which lines consume the time more, so we can replace them with better lines. The lines 12-18 are slowing the speed. the time for was 20 seconds.

```

0.47  9980  11      [IX_BLOCKS IY_BLOCKS]=meshgrid((1:X_blocks),(1:Y_blocks));
12          %for i=1:X_blocks
13              %for j=1:Y_blocks
14                  %if Plane(i,j)~=0
15                      % distance matrix(i,j) = ((i-kriging_point_index(1))^2+(j-kriging_point_index(2))^2)^.5 * (Plane(i,j)~=0) + A * (Plane(i,j)~=0);
4.96  9980  16      distance_matrix = ((IY_BLOCKS-kriging_point_index(1)).^2+(IX_BLOCKS-kriging_point_index(2)).^2).^5 .* (Plane ~= 0) + A .*
17                      %else
18                      %distance_matrix(i,j) = nan; % nan will be assigned to the distance of zero nodes as we want them not to
19                      %end % participate in the kriging.
20                  %end
21              %end
22
23          %% here we sort all the distances to select the nearest points to our
24          % target point, ( number of nearest points = points_number).
9.26  9980  25      [dummy sorted_index] = sort(distance_matrix(:));
0.08  9980  26      sorted_index = sorted_index(1:points_number);

```

Comparing to the previous figure, the lines of 12-18 are replaced with meshgrid and logical operators. The time has improved 4 seconds and was 16 seconds.