

**mGstat**

Copyright © 2004-2012 Thomas Mejer Hansen

---

**COLLABORATORS**

	<i>TITLE :</i> mGstat		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Thomas Mejer Hansen	July 10, 2012	

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>Installation</b>	<b>1</b>
1.1	mGstat . . . . .	1
1.1.1	System Requirements . . . . .	1
1.1.2	mGstat from SVN . . . . .	1
1.2	Extras . . . . .	2
1.2.1	GSTAT . . . . .	2
1.2.1.1	Making GSTAT available for mGstat . . . . .	2
1.2.2	VISIM . . . . .	2
1.2.3	SNESIM . . . . .	3
1.2.4	S-GeMS . . . . .	3
1.2.4.1	Setting the system environment variable for S-GeMS. . . . .	3
1.2.4.2	Compiling S-GeMS for Linux. . . . .	3
1.2.4.3	Running S-GeMS using WINE. . . . .	4
1.3	Global settings . . . . .	4
1.3.1	MGSTAT_VERBOSE_LEVEL . . . . .	4
1.3.2	SEMIVAR_DEF . . . . .	4
1.3.3	GSTLAPPLIHOME . . . . .	4
1.3.4	SGEMS_DEV . . . . .	5
<b>2</b>	<b>Native Matlab kriging algorithms</b>	<b>6</b>
2.1	Specifications of data locations . . . . .	6
2.2	Modeling spatial correlation . . . . .	7
2.2.1	Semivariogram specification . . . . .	7
2.2.2	Synthetic semivariogram . . . . .	8
2.2.3	Experimental semivariogram . . . . .	8
2.2.3.1	Native Matlab . . . . .	8
2.2.3.2	GSTAT . . . . .	8
2.3	Estimation . . . . .	9
2.3.1	Kriging Options . . . . .	9
2.3.1.1	kriging methods . . . . .	9

---

2.3.1.1.1	Simple kriging . . . . .	9
2.3.1.1.2	Ordinary kriging . . . . .	9
2.3.1.1.3	Kriging with a trend / Universal kriging . . . . .	9
2.3.1.2	kriging neighborhood . . . . .	11
2.3.2	Kriging examples . . . . .	11
2.3.2.1	1D Kriging, no data uncertainty . . . . .	11
2.3.2.2	Kriging a series of points - noise free data . . . . .	11
2.3.2.3	Kriging a series of point - SK, OK, Ktrend . . . . .	11
2.3.2.4	Kriging a series of point - SK, OK, Ktrend . . . . .	12
2.3.2.5	Kriging a series of point - SK, OK, Ktrend - Neighborhood . . . . .	12
2.3.2.6	Kriging a series of point - SK, OK, Ktrend - Neighborhood, noisy data . . . . .	12
2.4	Simulation . . . . .	13
2.4.1	Sequential Gaussian Simulation . . . . .	13
2.4.2	LU Gaussian Simulation . . . . .	13
2.4.3	FFT Moving Average . . . . .	13
<b>3</b>	<b>GSTAT from Matlab</b>	<b>14</b>
3.1	Working with GSTAT and mGstat . . . . .	14
3.1.1	The GSTAT parameter file in mGstat . . . . .	14
3.1.1.1	'data' field of GSTAT parameter file . . . . .	15
3.1.2	An interactive GSTAT session in Matlab . . . . .	15
3.2	GSTAT-related m-files . . . . .	15
3.2.1	gstat . . . . .	15
3.2.1.1	output precision . . . . .	15
3.2.2	gstat_read_par . . . . .	15
3.2.3	gstat_write_par . . . . .	16
3.2.4	gstat_krig . . . . .	16
3.2.5	gstat_convert . . . . .	16
3.2.6	gstat_binary . . . . .	16
3.2.7	semivar_exp_gstat . . . . .	16
3.3	GSTAT examples . . . . .	17
3.3.1	GSTAT ex03 . . . . .	17
3.3.2	GSTAT ex04 . . . . .	18
3.3.3	GSTAT ex05 . . . . .	18
3.3.4	GSTAT ex06 . . . . .	19
3.3.5	GSTAT ex07 . . . . .	19
3.3.6	GSTAT ex09 . . . . .	20
3.3.7	GSTAT ex10 . . . . .	20

---

---

<b>4</b>	<b>VISIM - Direct and Gaussian sequential simulation</b>	<b>22</b>
4.1	VISIM	22
4.1.1	Running VISIM from Matlab	22
4.1.1.1	Check path to VISIM	22
4.1.1.2	Running VISIM from Matlab	23
4.1.2	Plotting VISIM realizations	24
4.1.2.1	Plotting VISIM E-type	24
4.1.2.2	Plotting VISIM histogram	25
4.1.2.3	Plotting VISIM semivariogram	26
4.1.2.4	Plotting the sensitivity kernel	26
4.1.2.5	Plotting VISIM conditional lookup table	26
4.1.2.6	Plotting the distribution of the prediction error of the used volume average data	27
4.2	VISIM examples	27
4.2.1	visim_init	27
4.2.2	Direct Sequential Simulation	27
4.2.3	Correlated data errors	29
4.2.4	Simulation of linear inverse problems	29
4.2.5	Cross borehole tomography example	30
4.2.5.1	Conditional simulation through error simulation	30
4.2.5.2	Calculating averaging kernels for cross borehole tomography	30
4.2.5.3	Linear inversion : Cross borehole tomography example	31
<b>5</b>	<b>SNESIM - Single Normal Equation SIMulation</b>	<b>32</b>
5.1	Working with SNESIM and mGstat	32
5.1.1	Working with SNESIM and mGstat	32
<b>6</b>	<b>S-GeMS - The Stanford Geostatistical Modeling Software</b>	<b>34</b>
6.1	Install S-GeMS	34
6.2	S-GeMS data format	34
6.2.1	GEOEAS to S-GeMS	34
6.2.1.1	GEOEAS Point Set to S-GeMS-binary	34
6.2.1.2	GEOEAS GRID to S-GeMS-binary	35
6.2.2	S-GeMS to EAS	35
6.3	Using S-GeMS	35
6.3.1	Sequential simulation using S-GeMS	36
6.3.1.1	Specification of variogram model	38
6.3.2	Unconditional sequential Gaussian simulation using S-GeMS	39
6.3.3	Conditional sequential Gaussian simulation	39
6.3.3.1	conditional data as a variable	39

---

---

6.3.3.2	conditional data from file . . . . .	39
6.3.4	Unconditional SNESIM and FILTERSIM Gaussian simulation using S-GeMS . . . . .	41
6.3.5	Convert image to training image; . . . . .	42
6.3.6	Simulation demonstration . . . . .	43
6.3.7	Probability perturbation (PPM) . . . . .	46
<b>7</b>	<b>Matlab Reference</b>	<b>48</b>
7.1	Core functions . . . . .	48
7.1.1	Contents . . . . .	48
7.1.2	CreateMisfitFunction . . . . .	49
7.1.3	MakeXmlRef . . . . .	49
7.1.4	arcinfo2eas . . . . .	49
7.1.5	block_log . . . . .	49
7.1.6	cokrig_sk . . . . .	50
7.1.7	colorbar_shift . . . . .	50
7.1.8	colormap_nan . . . . .	50
7.1.9	colormap_squeeze . . . . .	51
7.1.10	comb_cprob . . . . .	51
7.1.11	comb_cprob_ind . . . . .	51
7.1.12	comb_cprob_nd . . . . .	52
7.1.13	covar_exp . . . . .	52
7.1.14	cpdf . . . . .	53
7.1.15	create_nscore_lookup . . . . .	53
7.1.16	create_nscore_lookup_old . . . . .	53
7.1.17	csemivar_exp . . . . .	53
7.1.18	deformat_variogram . . . . .	53
7.1.19	dual . . . . .	54
7.1.20	edist . . . . .	54
7.1.21	estim_taran . . . . .	54
7.1.22	etype . . . . .	54
7.1.23	f77strip . . . . .	54
7.1.24	fft_ma_3d . . . . .	55
7.1.25	format_variogram . . . . .	55
7.1.26	fresnel_punch . . . . .	55
7.1.27	gaussian_simulation_cholesky . . . . .	56
7.1.28	gstat . . . . .	56
7.1.29	gstat_binary . . . . .	57
7.1.30	gstat_convert . . . . .	57
7.1.31	gstat_krig . . . . .	57

---

---

7.1.32	gstat_krig_blinderror . . . . .	58
7.1.33	hpd_2d . . . . .	59
7.1.34	hpd_2d_point . . . . .	59
7.1.35	icpdf . . . . .	59
7.1.36	indicator_transform_con . . . . .	59
7.1.37	indicator_transform_dis . . . . .	60
7.1.38	inscore . . . . .	60
7.1.39	isorange . . . . .	60
7.1.40	krig . . . . .	60
7.1.41	krig_blinderror . . . . .	61
7.1.42	krig_covar_lik . . . . .	61
7.1.43	krig_crossval_1d_exh . . . . .	61
7.1.44	krig_npoint . . . . .	62
7.1.45	krig_optim_1d_exh . . . . .	62
7.1.46	krig_optim_mcmc . . . . .	62
7.1.47	krig_optim_ml . . . . .	62
7.1.48	krig_optim_range . . . . .	62
7.1.49	krig_volume . . . . .	63
7.1.50	least_squares_inversion . . . . .	63
7.1.51	least_squares_oneone . . . . .	63
7.1.52	least_squares_partition_data . . . . .	63
7.1.53	least_squares_slice . . . . .	63
7.1.54	mgstat_clean . . . . .	63
7.1.55	mgstat_demo . . . . .	64
7.1.56	mgstat_dir . . . . .	64
7.1.57	mgstat_set_path . . . . .	64
7.1.58	mgstat_verbose . . . . .	64
7.1.59	nanmean . . . . .	64
7.1.60	nanstd . . . . .	65
7.1.61	nanvar . . . . .	65
7.1.62	nhood . . . . .	65
7.1.63	normcdf . . . . .	65
7.1.64	norminv . . . . .	66
7.1.65	normpdf . . . . .	66
7.1.66	nscore . . . . .	66
7.1.67	plot_scale . . . . .	67
7.1.68	pos2index . . . . .	67
7.1.69	ppp . . . . .	67
7.1.70	precal_cov . . . . .	67

---



---

7.1.71	print_mul . . . . .	68
7.1.72	progress_txt . . . . .	68
7.1.73	rank_transform . . . . .	68
7.1.74	read_arcinfo_ascii . . . . .	69
7.1.75	read_bin . . . . .	69
7.1.76	read_eas . . . . .	69
7.1.77	read_emm . . . . .	69
7.1.78	read_gstat_ascii . . . . .	70
7.1.79	read_gstat_par . . . . .	70
7.1.80	read_gstat_semivar . . . . .	70
7.1.81	read_petrel . . . . .	70
7.1.82	read_punch_par . . . . .	70
7.1.83	read_surfer_grid . . . . .	70
7.1.84	rot2 . . . . .	71
7.1.85	scatter_dot . . . . .	71
7.1.86	scatter_hpd . . . . .	71
7.1.87	semivar . . . . .	71
7.1.88	semivar_exp . . . . .	71
7.1.89	semivar_exp_gstat . . . . .	72
7.1.90	semivar_map . . . . .	73
7.1.91	semivar_optim . . . . .	73
7.1.92	semivar_synth . . . . .	73
7.1.93	set_mgstat_path . . . . .	74
7.1.94	sgsim . . . . .	74
7.1.95	space2char . . . . .	74
7.1.96	spherical_spreading . . . . .	74
7.1.97	strip_space . . . . .	75
7.1.98	suptitle . . . . .	75
7.1.99	title_alt . . . . .	75
7.1.100	vonk2d . . . . .	76
7.1.101	watermark . . . . .	76
7.1.102	write_arcinfo_ascii . . . . .	76
7.1.103	write_bin . . . . .	77
7.1.104	write_eas . . . . .	77
7.1.105	write_gstat_ascii . . . . .	77
7.1.106	write_gstat_par . . . . .	77
7.1.107	write_punch_par . . . . .	78
7.1.108	write_surfer_grid . . . . .	78
7.2	VISIM functions . . . . .	78

---

---

7.2.1	G_to_visim . . . . .	78
7.2.2	calc_gstat_semivar . . . . .	78
7.2.3	covar_prob . . . . .	79
7.2.4	read_visim . . . . .	79
7.2.5	semivar_mat . . . . .	79
7.2.6	visim . . . . .	79
7.2.7	visim_calc_semivar_prob . . . . .	79
7.2.8	visim_cholesky . . . . .	79
7.2.9	visim_clean . . . . .	79
7.2.10	visim_error_sim . . . . .	79
7.2.11	visim_format_variogram . . . . .	79
7.2.12	visim_init . . . . .	79
7.2.13	visim_lsq . . . . .	79
7.2.14	visim_make_movie . . . . .	79
7.2.15	visim_mask . . . . .	80
7.2.16	visim_merge_volume_data . . . . .	80
7.2.17	visim_movie . . . . .	80
7.2.18	visim_plot . . . . .	80
7.2.19	visim_plot_condtab . . . . .	80
7.2.20	visim_plot_etype . . . . .	80
7.2.21	visim_plot_hist . . . . .	80
7.2.22	visim_plot_kernel . . . . .	80
7.2.23	visim_plot_kernel2 . . . . .	80
7.2.24	visim_plot_rpath . . . . .	80
7.2.25	visim_plot_semivar . . . . .	80
7.2.26	visim_plot_semivar2 . . . . .	80
7.2.27	visim_plot_semivar_compare . . . . .	81
7.2.28	visim_plot_semivar_mul . . . . .	81
7.2.29	visim_plot_semivar_real . . . . .	81
7.2.30	visim_plot_sim . . . . .	81
7.2.31	visim_plot_stat . . . . .	81
7.2.32	visim_plot_volfit . . . . .	81
7.2.33	visim_plot_volume . . . . .	81
7.2.34	visim_precal_linearization . . . . .	81
7.2.35	visim_prior_prob . . . . .	81
7.2.36	visim_prior_prob_gibbs . . . . .	81
7.2.37	visim_prior_prob_mcmc . . . . .	81
7.2.38	visim_prior_prob_mcmc_tomolin . . . . .	81
7.2.39	visim_prior_prob_sill . . . . .	82

---

---

7.2.40	visim_semivar . . . . .	82
7.2.41	visim_semivar_pdf . . . . .	82
7.2.42	visim_semivar_uncon . . . . .	82
7.2.43	visim_set_broadband_covariance . . . . .	82
7.2.44	visim_set_conditional_point . . . . .	82
7.2.45	visim_set_multiscale_covariance . . . . .	82
7.2.46	visim_set_resim_data . . . . .	82
7.2.47	visim_setup_punch . . . . .	82
7.2.48	visim_setup_tomo_kernel . . . . .	82
7.2.49	visim_slice_volume . . . . .	82
7.2.50	visim_to_G . . . . .	82
7.2.51	visim_tomo_setup . . . . .	83
7.2.52	visim_tomography . . . . .	83
7.2.53	visim_tomography_forward . . . . .	83
7.2.54	visim_tomography_linearize . . . . .	83
7.2.55	vvisim . . . . .	83
7.2.56	write_visim . . . . .	83
7.3	SNESIM functions . . . . .	83
7.3.1	read_snesim . . . . .	83
7.3.2	snesim . . . . .	83
7.3.3	snesim_init . . . . .	83
7.3.4	snesim_prior_sampler . . . . .	83
7.3.5	snesim_set_resim_data . . . . .	83
7.3.6	write_snesim . . . . .	83
7.4	S-GeMS functions . . . . .	84
7.4.1	eas2sgems . . . . .	84
7.4.2	sgems . . . . .	84
7.4.3	sgems2eas . . . . .	84
7.4.4	sgems_clean . . . . .	84
7.4.5	sgems_demo . . . . .	84
7.4.6	sgems_get_par . . . . .	84
7.4.7	sgems_grid . . . . .	84
7.4.8	sgems_grid_py . . . . .	84
7.4.9	sgems_image2ti . . . . .	84
7.4.10	sgems_plot_project . . . . .	84
7.4.11	sgems_plot_structure . . . . .	84
7.4.12	sgems_ppm . . . . .	84
7.4.13	sgems_read . . . . .	85
7.4.14	sgems_read_project . . . . .	85

---

---

7.4.15	sgems_read_xml . . . . .	85
7.4.16	sgems_set_resim_data . . . . .	85
7.4.17	sgems_variogram_xml . . . . .	85
7.4.18	sgems_write . . . . .	85
7.4.19	sgems_write_grid . . . . .	85
7.4.20	sgems_write_pointset . . . . .	85
7.4.21	sgems_write_xml . . . . .	85
7.5	Misc functions . . . . .	85
7.5.1	LowPass1 . . . . .	85
7.5.2	LowPass1nan . . . . .	85
7.5.3	LowPass2 . . . . .	85
7.5.4	LowPass2nan . . . . .	86
7.5.5	accept_rate . . . . .	86
7.5.6	caxis_squeeze . . . . .	86
7.5.7	cmap_quantile . . . . .	86
7.5.8	cmap_rwb . . . . .	86
7.5.9	conv2_strip . . . . .	86
7.5.10	conv_strip . . . . .	86
7.5.11	despike . . . . .	86
7.5.12	figure_focus . . . . .	86
7.5.13	find_row_array . . . . .	86
7.5.14	gamma_coefficients . . . . .	86
7.5.15	gaussian_likelihood . . . . .	86
7.5.16	generalized_gaussian . . . . .	87
7.5.17	generalized_gaussian_2d . . . . .	87
7.5.18	isoctave . . . . .	87
7.5.19	jura . . . . .	87
7.5.20	kernel_buursink_2d . . . . .	87
7.5.21	kernel_finite_2d . . . . .	87
7.5.22	kernel_fresnel_2d . . . . .	87
7.5.23	kernel_fresnel_monochrome_2d . . . . .	87
7.5.24	kernel_multiple . . . . .	87
7.5.25	kernel_slowness_to_velocity . . . . .	87
7.5.26	kml_line . . . . .	87
7.5.27	kml_points . . . . .	87
7.5.28	mspectrum . . . . .	88
7.5.29	munk_fresnel_2d . . . . .	88
7.5.30	munk_fresnel_3d . . . . .	88
7.5.31	pick_first_arrival . . . . .	88

---

7.5.32	plot_mh_error_hist . . . . .	88
7.5.33	plot_mh_loglikelihood . . . . .	88
7.5.34	punch . . . . .	88
7.5.35	rayinv_grid_v . . . . .	88
7.5.36	rayinv_load_tx . . . . .	88
7.5.37	rayinv_load_v . . . . .	88
7.5.38	rayinv_plot_tx . . . . .	88
7.5.39	rickerwavelet . . . . .	88
7.5.40	sample_generalized_gaussian . . . . .	89
7.5.41	set_paper . . . . .	89
7.5.42	set_resim_data . . . . .	89
7.5.43	statusbar . . . . .	89
7.5.44	su_write_model . . . . .	89
7.5.45	test_fd_su_gpr . . . . .	89

---

# List of Tables

2.1 kriging options . . . . . 10

## **Abstract**

mGstat is a geostatistical Matlab toolbox. Both native functions and interfaces to GSTAT, VISIM(GSLIB), SNESIM, and S-GeMS are provided

# Chapter 1

## Installation

### 1.1 mGstat

The latest version of mGstat is available from [Sourceforge.net](https://sourceforge.net).

Unpacking the downloaded file as

```
cd /home/tmh/matlab
tar xvfz mgstat-1.00.tgz
```

will create a directory structure called 'mGstat'. To use the toolbox from Matlab, simply add the path where mGstat is installed to the Matlab path as

```
>> addpath /home/tmh/matlab/mGstat
>> addpath /home/tmh/matlab/mGstat/visim
>> addpath /home/tmh/matlab/mGstat/snesim
>> addpath /home/tmh/matlab/mGstat/sgems
>> addpath /home/tmh/matlab/mGstat/misc
```

The above paths can also be set running "[mgstat\\_set\\_path](#)".

See the Matlab documentation for instructions on how to permanently add the paths to the mGstat toolbox. In Windows this can be done using the `pathtool`

#### 1.1.1 System Requirements

mGstat is implemented using Matlab R2009b.

An effort has been made to ensure that the toolbox works with Octave version 3.2.4 ([download](#)). Octave is a free alternative to Matlab. Note however that Matlab is the main tool for development.

mGstat should work on any operating system running Matlab or Octave. However, interfaces GSTAT ([?title]), S-GeMS ([?title]), VISIM ([?title]), and SNESIM ([?title]).

#### 1.1.2 mGstat from SVN

The latest development version of mGstat can be obtain from an SVN (Subversion) repository, which can be checked out using the following command:

```
svn co https://mgstat.svn.sourceforge.net/svnroot/mgstat/trunk mgstat
```

The repository can also be [browsed at SourceForge](#).



## 1.2 Extras

In addition to the native matlab geostatistical commands, mGstat contains interfaces to GSTAT ([?title]), S-GeMS ([?title]), VISIM ([?title]), and SNESIM ([?title]).

As of version 0.9, mGstat includes compiled windows binaries for, GSTAT, VISIM and SNESIM. Source and compiled windows version of S-GeMS is available online (see below).

For unix systems, GSTAT, VISIM, SNESIM, and nfd must be compiled from scratch. The windows exe-file for %sgems; can optionally be run on Linux/OSX using WINE. mGstat includes source code for VISIM and SNESIM. Source code for GSTAT, S-GeMS, and nfd can be downloaded online (see below). Note that all programs but nfd is released under open source licences.

### 1.2.1 GSTAT

As of version 0.8, mGstat comes with a gstat.exe pre-compiled for Windows, and thus no further installation should be necessary.

As of version 1.0, mGstat comes with a pre-compiled binary for Intel Mac gstat\_mac\_g3, and thus no further installation should be necessary.

From the GSTAT website (<http://www.gstat.org/>) both source and binary packages can be downloaded.

The [GSTAT PDF manual](#) is a very useful source of information to have access to while using GSTAT.

#### 1.2.1.1 Making GSTAT available for mGstat

mGstat use the m-file gstat\_binary to locate the GSTAT binary file. gstat\_binary searches the system path to find a binary executable called gstat, and chooses the first it encounters. In the mGstat/bin directory is also scanned.

Simply running from within Matlab should return the full path to the GSTAT binary as

```
>> gstat_binary

ans =

/usr/local/bin/gstat
```

If the gstat binary file is not in the system path, or if you have difficulties using the default automatic localization, gstat\_binary.m can be manually edited to point to a specific location. Simple replace

```
gstat = '';
```

with

```
gstat='/usr/local/bin/gstat';
```

### 1.2.2 VISIM

As of version 0.8, mGstat comes with a visim.exe pre-compiled for Windows. Therefore no further installation should be necessary on Windows.

As of version 0.9, the source code for VISIM is part of the mGstat code. The source code is located in mGstat/visim/visim/src

To compile visim ;

```
cd mGstat/visim/visim/src/gslib
# EDIT Makefile to point to your fortran 77 or fortran 90 compiler
make
cd mGstat/visim/visim/src
# EDIT Makefile to point to your fortran 77 or fortran 90 compiler
# EDIT visim.inc to change/set the maximum array sized
make
```

this should create a binary 'visim' or 'visim.exe' (windows). Copy this file to

```
cp mGstat/visim/visim/src/visim mGstat/bin/.
```

A detailed manual for VISIM can be obtained from [Inverse modeling and geostatistics project](#).

### 1.2.3 SNESIM

As of version 0.9, mGstat comes with a snesim.exe pre-compiled for Windows. Therefore no further installation should be necessary on Windows.

As of version 0.9, the source code for VISIM is part of the mGstat code. The source code is located in mGstat/visim/visim/src

### 1.2.4 S-GeMS

S-GeMS source, and compiled windows exe files, can be downloaded from <http://sgems.sourceforge.net/>. S-GeMS version 2.1 is supported by mGstat ([http://downloads.sourceforge.net/sgems/sgems-2.1\\_installer.exe](http://downloads.sourceforge.net/sgems/sgems-2.1_installer.exe)).

On windows systems the S-GeMS binary is assumed to be located in c:\Program Files\SGeMS\sgems.exe. If it is located on another location, it should be manually set in "sgems".

On Linux, mGstat can call both a natively compiled S-GeMS binary and the Windows compiled S-GeMS binary using WINE

#### 1.2.4.1 Setting the system environment variable for S-GeMS.

In order to call S-GeMS from within Matlab the environment variable GSTLAPPLI must be set and point to the installation directory for S-GeMS.

Using linux and the bash shell use:

```
export GSTLAPPLI=/path/to/sgems_dir
```

Using windows Vista, right click on 'My Computer'; select properties; select Advanced System Setting; Choose 'environment variables';

Using windows XP, right click on 'My Computer'; select 'Properties'; select the 'Advanced' tab; click 'Environment Variables'; Click 'New' to add a system variable called GSTLAPPLI and set to the value /path/to/sgems\_dir

#### 1.2.4.2 Compiling S-GeMS for Linux.

A guide for compiling S-GeMS from source code on Ubuntu Linux is available here: [http://imgp.imm.dtu.dk/compile\\_sgems.php](http://imgp.imm.dtu.dk/compile_sgems.php).

### 1.2.4.3 Running S-GeMS using WINE.

If you fail to compile S-GeMS natively, you can run the windows binary EXE through using operating systems supported by **WINE**. WINE allow running windows applications on Linux, BSD, Solaris and Mac OS X. SGeMS seems to work fine using WINE

Start by installing wine, which is available as a package for most Linux systems (for Debian/Ubuntu use):

```
sudo apt-get install wine
```

Then install the S-GeMS windows executable (download from [http://downloads.sourceforge.net/sgems/sgems-2.1\\_installer.exe](http://downloads.sourceforge.net/sgems/sgems-2.1_installer.exe)) using WINE:

```
wine sgems-2.1_installer.exe
```

This should create desktop icon for S-GeMS (running through wine).

To use S-GeMS through WINE edit `sgems.m` to make make sure the `use_wine_on_unix` set to '1': or set a system environmental variable as:

```
setenv('USE_WINE_ON_UNIX','1')
```

## 1.3 Global settings

A number of environment variables can be set to control the behaviour of mGstat. The environment variable can be set in the operating system, or directly through Matlab using:

```
setenv('ENV_VAR_NAME','ENV_VAR_VALUE');
```

### 1.3.1 MGSTAT\_VERBOSE\_LEVEL

The amount of information printed to the screen when using mGstat can be controlled by the setting the `MGSTAT_VERBOSE_LEVEL`. By default it is set to

```
setenv('MGSTAT_VERBOSE_LEVEL','1')
```

A higher number increase, while a smaller number reduce the information printed to screen.

### 1.3.2 SEMIVAR\_DEF

There are different definitions of semivariogram models. To make use of the definions used by GSTAT and S-GeMS (and GSLIB) set the following:

```
setenv('SEMIVAR_DEF','GSTAT')
setenv('SEMIVAR_DEF','SGeMS')
```

The default is to use SGeMS definitions. See also “[Modeling spatial correlation](#)”.

### 1.3.3 GSTLAPPLIHOME

The installation directory of S-GeMS can be given by the variable: `GSTLAPPLIHOME` This is used to find the executable file for launching S-GeMS.

### 1.3.4 SGEMS\_DEV

If you are using the development version of S-GeMS you should set this variable to 1:

```
setenv('SGEMS_DEV','1')
```

The default option is to you the stable version of S-GeMS, and then this variable does not have to be set.

Note that mGstat may not yet support the latest development version of S-GeMS, so it is not recommended to use the development version for normal users.

---

## Chapter 2

# Native Matlab kriging algorithms

This chapter discuss how to run a number of kriging algorithms implemented natively in Matlab.

These algorithms are note fast, but may be useful for smaller problems, and for teaching purposes. For larger scale problems consider using the Matlab interfaces to GSTAT ([?title]) and S-GeMS ([?title]).

### 2.1 Specifications of data locations

A point in mGstat can be multidimensional (higher than 3 dimensions). Only the native Matlab implementations of geostatistical algorithm can handle this. GSTAT and GSLIB are restricted to 3D.

A point is given by a 1-row vector, where the number of columns definexs the dimension. For example, the location of the three 1D-points  $(x_1, x_2, x_3) = (1, 5, 10)$  is given by

```
>> pos=[1;5;10];
>> [ndata,ndim]=size(pos)

ndata =
      3

ndim =
      1
```

More than one location is specified by a matrix where each row defines one point, and the number of rows is the number of locations. For example, the three 4-dimensional points,  $(2, 2, 4, 8)$ ,  $(1, 2, 3, 4)$ ,  $(6, 6, 2, 2)$  is given by

```
>> pos=[2 2 4 8; 1 2 3 4; 6 6 2 2];
>> [ndata,ndim]=size(pos)

ndata =
      3

ndim =
      4
```

To transform locations from array structures to the matrix shape used by mGstat is straightforward in Matlab. The following two arrays denoting x and y locations

```
x =[ 1      2      3
     1      2      3];

y =[ 4      4      4
     5      5      5];
```

can be converted to the format required by mGstat by

```
>> pos=[x(:) y(:)]
pos =
     1     4
     1     5
     2     4
     2     5
     3     4
     3     5

>> [ndata,ndim]=size(pos)
ndata =
     6

ndim =
     2
```

## 2.2 Modeling spatial correlation

### 2.2.1 Semivariogram specification

Semivariogram models are specified using the GSTAT notation. For example a Spherical semivariogram model with a range of 1 and a sill of 0.1 is specified as

```
V = '0.1 Sph(1)'
```

Nested semivariogram models are specified as for example:

```
V= '0.1 Nug(0) + 0.1 Sph(1)'
```

2D anisotropy is specified as the angle of the primary direction clockwise from North, `rotation`, and the fraction of range of the range in the secondary direction (perpendicular to the primary direction) to the range of the primary direction, `anisotropy_factor`:

```
V = 'sill Sph(range,rotation,anisotropy_factor)'
```

To specify an angle of 30 degrees from north and a fraction of 0.3 :

```
V = '0.1 Sph(1,30,.3)'
```

See more details in the [GSTAT manual](#).

Internally in Matlab the string describing the semivariogram model is translated into a Matlab structure. There is significant computational improvements by doing this conversion, and it allows an easier way to set semivariogram parameters from Matlab. Converting a semivariogram from string format a Matlab structure can be done using “`deformat_variogram`”:

```
V = deformat_variogram('0.1 Sph(1,30,.3)')
V =
    par1: 0.1000
    par2: [1 30 0.3000]
    type: 'Sph'
    itype: 1
```

The semivariogram in form of the Matlab structure is also much easier to manipulate.

To convert a semivariogram in form of a Matlab structure back into a readable string using “`format_variogram`”:

```
format_variogram(V,1)
ans =
0.1 Sph(1,30,0.3)
```

## 2.2.2 Synthetic semivariogram

A synthetic semivariogram can be calculated using `semivar_synth`, “`semivar_synth`”:

```
V='0.1 Nug(0) + 1 Gau(1.5)';
[sv,d]=semivar_synth([0:.1:6]);
plot(d,sv)
```

Note that there are different conventions for the definition of some semivariogram models. GSLIB and S-GeMS use one convention, while for example GSTAT use another. Default mGstat make use of the definitions used in mGstat. To change this see “`SEMIVAR_DEF`”.

## 2.2.3 Experimental semivariogram

There are two ways to calculate the experimental semivariogram. A native matlab function and a wrapper to GSTAT. The native matlab function allows use of multidimensional data, while the GSTAT wrapper only allows up to 3D data observations. The native Matlab function allows computation of several angle ranges at once while the GSLIB wrapper must be called separately for each angle array. However, the GSLIB wrapper is much faster.

### 2.2.3.1 Native Matlab

“`semivar_exp`” is a native Matlab function to compute directional dependent experimental semivariograms from multi dimensional data observations. An example assuming an isotropic semivariogram model:

```
[gamma,h]=semivar_exp(pos,val);
plot(h,gamma);
```

The semivariogram in a number of angle ranges can be simultaneously calculated using:

```
% Example directional [0,45,90,135,180]:
[gamma,h,angle_center]=semivar_exp(pos,val,20,4);
plot(h,gamma);
legend(num2str(angle_center))
```

This computes the semivariogram in for angle arrays, from 0-45, 45-90, 90-135, 135-180. 'angle\_center' is the center of each angle gather : 22.5, 67.5, 112.5, 157.5 degrees.

The angle range can manually be specified using:

```
% Example directional [0,45,90,135,180]:
ang=[0 45 90 135 180];
[gamma,h,angle_center]=semivar_exp(pos,val,20,4);
plot(h,gamma);
legend(num2str(angle_center))
```

The semivariogram for angles between for example 13 and 22 degrees, can be computed using :

```
% Example directional 13-22 deg
[gamma,h,angle_center]=semivar_exp(pos,val,20,[13 22]);
plot(h,gamma);
legend(num2str(angle_center))
```

### 2.2.3.2 GSTAT

“`semivar_exp_gstat`” is a wrapper for GSTAT for computing the directional dependent experimental semivariogram for one angle range. It supports up to three dimensional data observations. The angle range is specified as the angle and a tolerance. Thus the semivariogram in within the angle range 20 +/-10 degrees is found using :

```
[gamma,h]=semivar_exp_gstat(pos,val,20,10)
```

## 2.3 Estimation

Multidimensional kriging estimation with noisy data observations can be performed using “**krig**”

```
[d_est,d_var,lambda_sk,K_dd,k_du,inhood]=krig(pos_known,val_known,pos_est,V,options);

ndata : number of data observations
ndims : dimensions of data location (>=1)
nest  : number of data locations to be estimated

pos_known [ndata,ndims] : Locations of data observations
val_known [ndata,1 or 2] : col1 : Data value as measured at 'pos_known'
                        col2 : Data uncertainty as measured at
                        'pos_known' (optional)
pos_est   [1,ndims] : Location of data to be estimated
V : Variogram model, e.g. '1 Sph(100)'
options : kriging options.
```

Please note data uncertainty is listed as the second column of `val_known`. Data uncertainty is defined as the variance of Gaussian noise distribution associated to each data measurement. Specified in this manner, the noise on data observations is considered uncorrelated. If only one column is specified, data observations are treated as noise free.

### 2.3.1 Kriging Options

`options`, the last argument, is a Matlab structure that controls most aspect of the kriging, such as choosing the neighborhood, and kriging method. The names for most of these options are same as the names used to control GSTAT, see [?title]

#### 2.3.1.1 kriging methods

the Kriging methods: simple Kriging, ordinary Kriging and Kriging with a trend are all available using `krig`. The difference between these three methods is the way the trend is modelled. Simple Kriging assumed a constant and known mean. Ordinary kriging makes use of an varying unknown mean (that is estimated within the neighborhood). Kriging with a trend model the trend as a smoothly varying polynomial trend. Thus ordinary kriging is but a form of kriging with a trend,; a 0th order polynomial trend.

##### 2.3.1.1.1 Simple kriging

By default ordinary Kriging in an exhaustive neighborhood (i.e. all data measurements are used all the time) is performed. This is identical to simple Kriging when `options.mean=mean(pos_known(:,1))`

##### 2.3.1.1.2 Ordinary kriging

If the 'mean' is given as `options.mean`, Simple Kriging is performed.

##### 2.3.1.1.3 Kriging with a trend / Universal kriging

For each dimension (direction) the order of the polynomial fit of the trend can be specified by `options.polyfit`

if the value of `options.polyfit` is an integer scalar, then the same polynomial fit is used for all dimensions. Thus the default linear trend in all directions is identical to specifying `options.polyfit=1`.

if the value of `options.polyfit` is an array of integers, each integer in the array must specify the order of the trend for each dimension. The length of the array must be equal to the size of the dimension of the observation. Thus, for 2D data observations one can specify a 2nd order polynomial trend in the first direction and a 0th order fit in the second direction as `options.polyfit=[2 0]`.



<b>option.[?]</b>	<b>GSTAT equiv</b>	<b>description</b>	<b>link</b>
sk_mean,mean	sk_mean	Specify the global mean, as used by simple kriging	
trend	trend	Krig only the trend.	
max	max	The maximum number of data to use in the search neighborhood	
polytrend	d	The polynomial order of the trend for each dimension	
xvalid	xvalid	If specified as xvalid=1 cross validation on known data locations is performed	
isorange		isorange=1 assumes no rotation in anisotropy. If chosen, each entry in the range section of a variogram corresponds to the range for a certain dimension. For example 'Sph(1,10,1,100)' corresponds to a range of 1 in the first dimension, 10 in the second dimension, 1 in the third dimension and 100 in the fourth dimension. If GSTAT is used for kriging in 1, 2 and 3 dimensions, the range selections are translates properly to GSTAT format using the isorange function.	

Table 2.1: kriging options

### 2.3.1.2 kriging neighborhood

By default all data observations are always used. The kriging neighborhood denote the data that are actually used by the kriging systems.

A maximum number of data used by the kriging system is selected as `options.max=10`. The data locations closes to the point being estimated is retained in the data neighborhood.

See also “**nhood**” that controls the data neighborhood behavior.

## 2.3.2 Kriging examples

Based on the data observations below, a number of example are shown illustrating the use of kriging in mGstat

### 2.3.2.1 1D Kriging, no data uncertainty

In the following example 3 data measurements has been made :  $x(1)=0$ ;  $x(5)=3$ ;  $x(2)=2$ . Using a Spherical semivariogram model with a range of '0.2' and a sill of '1', the mean and variance of the distribution of the local probability density function at  $x(2)$  is found:

```
pos_known=[1;5;10];
val_known=[0;3;2]; % adding some uncertainty
V='1 Sph(.2)';      % Select variogram model
pos_est=[2]';
[d_est,d_var]=krig(pos_known,val_known,pos_est,V)
```

```
d_est =

    1.6667
```

```
d_var =

    1.3333
```

from: mGstat/examples/mgstat\_examples/krig\_ex1.m

### 2.3.2.2 Kriging a series of points - noise free data

```
pos_known=[1;5;10];
val_known=[0;3;2]; %
V='1 Sph(.2)';      % Select variogram model
pos_est=[0:.1:10]';
[d_est,d_var]=krig(pos_known,val_known,pos_est,V);
plot(pos_est,d_est,'k',pos_known,val_known,'ro')
print -dpng krigex2
```

### 2.3.2.3 Kriging a series of point - SK, OK, Ktrend

```
pos_known=[1;5;10]; %
val_known=[0;3;2]; %
V='1 Sph(.2)';      % Select variogram model
pos_est=[0:.1:10]';
[d_est_ok,d_var_ok]=krig(pos_known,val_known,pos_est,V);
options.mean=2;
```

```
[d_est_sk,d_var_sk]=krig(pos_known,val_known,pos_est,V,options);
options.trend=1;
[d_est_kt,d_var_kt]=krig(pos_known,val_known,pos_est,V,options);
plot(pos_est,[d_est_sk,d_est_ok,d_est_kt],'-',pos_known,val_known,'ro')
legend('SK','OK','KT','Data')
print -dpng krigex3
```

### 2.3.2.4 Kriging a series of point - SK, OK, Ktrend

```
rand('seed',1)
ndata=10;
pos_known=rand(ndata,1)*10;
val_known=randn(ndata,1); %
V='1 Sph(.2)'; % Select variogram model
pos_est=[0:.1:10]';
clear options;
[d_est_ok,d_var_ok]=krig(pos_known,val_known,pos_est,V,options);
options.mean=2;
[d_est_sk,d_var_sk]=krig(pos_known,val_known,pos_est,V,options);
options.trend=1;
[d_est_kt,d_var_kt]=krig(pos_known,val_known,pos_est,V,options);
plot(pos_est,[d_est_sk,d_est_ok,d_est_kt],'-',pos_known,val_known,'k*')
legend('SK','OK','KT','Data')
print -dpng krigex4
```

### 2.3.2.5 Kriging a series of point - SK, OK, Ktrend - Neighborhood

```
rand('seed',1)
ndata=10;
pos_known=rand(ndata,1)*10;
val_known=randn(ndata,1); %
V='1 Sph(.2)'; % Select variogram model
pos_est=[0:.1:10]';
clear options;options.max=4;
[d_est_ok,d_var_ok]=krig(pos_known,val_known,pos_est,V,options);
options.mean=2;
[d_est_sk,d_var_sk]=krig(pos_known,val_known,pos_est,V,options);
options.trend=1;
[d_est_kt,d_var_kt]=krig(pos_known,val_known,pos_est,V,options);
plot(pos_est,[d_est_sk,d_est_ok,d_est_kt],'-',pos_known,val_known,'k*')
legend('SK','OK','KT','Data')
print -dpng krigex5
```

### 2.3.2.6 Kriging a series of point - SK, OK, Ktrend - Neighborhood, noisy data

```
rand('seed',1)
ndata=30;
pos_known=rand(ndata,1)*10;
val_known=randn(ndata,1); %
val_var=zeros(ndata,1)+.1; %
V='1 Sph(.1)'; % Select variogram model
pos_est=[-2:.1:12]';
clear options;options.max=4;
[d_est_ok,d_var_ok]=krig(pos_known,[val_known val_var],pos_est,V,options);
options.mean=mean(val_known);
[d_est_sk,d_var_sk]=krig(pos_known,[val_known val_var],pos_est,V,options);
```

```
options.trend=1;
[d_est_kt,d_var_kt]=krig(pos_known,[val_known val_var],pos_est,V,options);
plot(pos_est,[d_est_sk,d_est_ok,d_est_kt],'-',pos_known,val_known,'k*')
legend('SK','OK','KT','Data')
print -dpng krigex6
```

## 2.4 Simulation

### 2.4.1 Sequential Gaussian Simulation

sgsim can be used to perform sequential Gaussian simulation. It make use of “**krig**”, and can be called using the same command arguments. In addition, one can set 'options.nsim' to define how many realizations is generated.

```
options.nsim = 10;
```

Note that this implementation og SGSIM is vert slow, and only included as an example. Consider using VISIM og SGEMS instead [\[link\]](#).

### 2.4.2 LU Gaussian Simulation

Gaussian Simulation based on LU decomposition of the covariance model is a precise (no use of neighborhood) method for generating realizations of Gaussian random models. But, it is only practical when the number of model parameters is realatively small, beacuse a full  $(n_x \times n_y \times n_z) \times (n_x \times n_y \times n_z)$  covariance model need to be setup.

```
% Load Jura Data
[d_prediction,d_transect,d_validation,h_prediction,h_transect,h_validation,x,y,pos]=jura ←
    (.1);
Va = '1 Sph(8)';
pos_known = d_prediction(:,1:2);
val_known = d_prediction(:,5);

options.nsim=9;
[sim_data,m_est,Cm_est]=lusim(pos_known,val_known,pos,Va,options);

for i=1:options.nsim;
subplot(3,3,i);
    imagesc(x,y,reshape(sim_data(:,i),length(y),length(x)));%axis image;caxis([-2 2])
    axis image;
end
```

### 2.4.3 FFT Moving Average

The FFT Moving Average method is a very efficient method for generating unconditional realizations of Gaussian random models. As opposed to the “**LU Gaussian Simulation**” method, one only need to define the covariance model in a grid of size  $(n_x \times n_y \times n_z)$ . [\[REF TO PAPER\]](#)

```
x=[1:1:40];y=1:1:35;z=1:30;
Va='1 Sph(10,30,.5)';
[out,z]=fft_ma(x,Va); % 1D
[out,z]=fft_ma(x,y,Va); % 2D
[out,z]=fft_ma(x,y,z,Va); %3D
```

Some information about padding and resimulation ...

## Chapter 3

# GSTAT from Matlab

This chapter discuss how to run GSTAT from within Matlab.

### 3.1 Working with GSTAT and mGstat

#### 3.1.1 The GSTAT parameter file in mGstat

GSTAT can be run non-interactively using by parsing a parameter file. It is by reading and writing this printer file that mGstat interfaces with GSTAT. Consider the following GSTAT parameter file

```
#
# Local simple point kriging on a mask map
#
data(ln_zinc): 'zinc.eas', x=1, y=2, v=3, log,
  min=20, max=40, radius=1000, sk_mean=5.9;
variogram(ln_zinc): 0.0554 Nug(0) + 0.581 Sph(900);
mask: 'mask_map';
predictions(ln_zinc): 'lzn_skpr';
variances(ln_zinc): 'lzn_skvr';
```

using `read_gstat_par` to read the parameter into the data structure `G`, gives

```
>> G=read_gstat_par(par)
G =
    mgstat: [1x1 struct]
    data: {[1x1 struct]}
    variogram: {[1x1 struct]}
    mask: {[1x1 struct]}
    predictions: {[1x1 struct]}
    variances: {[1x1 struct]}

>> G.mgstat
ans =
    parfile: '/home/tmh/RESEARCH/PROGRAMMING/gstat-2.4.4/cmd/ex05.cmd'
    comment: {'#' '#Local simple point kriging on a mask map' '#'}

>> G.data{1}
ans =
    data: 'ln_zinc'
    file: 'zinc.eas'
    x: 1
    y: 2
```

```

        v: 3
        log: ''
        min: 20
        max: 40
        radius: 1000
        sk_mean: 5.9000

>> G.predictions{1}
ans =
    data: 'ln_zinc'
    file: 'lzn_skpr'

```

The `mgstat` field to the `G` structure, is specific to `mGstat` and stores the comments of the parameter file, and the location on disk of the parameter file.

The rest of the fields of `G` refers to lines in the `GSTAT` parameter file.

### 3.1.1.1 'data' field of GSTAT parameter file

Take for example the `data` field as listed above. The field is called `data`, because this is the identifier (first string) in the parameter file. Filename and data identifiers are always specified using 'data' and 'file' fields. Several options can be specified for the 'data'. This are simply listed as fields of `G.data{1}`. If an option does not supply a value (as the 'log' option) it simply refers to an empty string.

In case several data type are listed in the parameter file, they are read into separate structures, as `G.data{1}`, `G.data{2}`, ...

## 3.1.2 An interactive GSTAT session in Matlab

to come

## 3.2 GSTAT-related m-files

### 3.2.1 gstat

`gstat` is used to call `gstat` from within Matlab. It can be called using either a `GSTAT` command file as :

```
[pred_mean,pred_var]=gstat('ex06.cmd');
```

or using a Matlab `mGstat` structure as

```

G=read_gstat_par('ex06.cmd'); % read parameter file
G.G.variogram{1}.V(1).type='Sph'; % change variogram type to Spherical
[pred_mean,pred_var]=gstat(G);

```

More info at : Section [7.1.28](#)

#### 3.2.1.1 output precision

By default the output precision of `GSTAT` is set to format '%16.8f'. This can be specified manually as :

```
G.set.format='%4.2f'
```

### 3.2.2 gstat\_read\_par

More info at : Section [7.1.79](#)

### 3.2.3 gstat\_write\_par

More info at : [Section 7.1.106](#)

### 3.2.4 gstat\_krig

`gstat_krig` is the equivalent of the native Matlab command `krig`. It is called in similar fashion to `krig`, but calls GSTAT for kriging as opposed to the Matlab kriging routines.

More info at : [Section 7.1.31](#)

### 3.2.5 gstat\_convert

Converts binary formatted data formats to ascii.

More info at : [Section 7.1.30](#) and [Section 1.2.1.1](#)

### 3.2.6 gstat\_binary

More info at : [Section 7.1.29](#)

### 3.2.7 semivar\_exp\_gstat

NB: In the forthcoming releases this m-file will be rename `gstat_semivar_exp`

This m-file computes the experimental semivariogram using the same conventions as [Section 7.1.88](#), but using GSTAT as backend instead of the native Matlab implementation.

[Section 7.1.89](#) is much more CPU efficient than [Section 7.1.88](#).

An example (from `MGSTAT_INSTALL/example/test_gstat_semivar_exp.m`)

```
% GENERATE A REFERENCE DATA SET USING UNCONDITIONAL GAUSSIAN SIMULATION
x=[0:.05:10];
y=[0:.05:10];
V=visim_init(x,y);
V.rseed=1;
V.Va.a_hmax=4; % maximum correlation length
V.Va.a_hmin=.5; % minumum correlation length
V.Va angl=90-22.5; % Rotation angle of dip(clockwise from north)
V.Va.it=1; % Gaussian semivariogram
V=visim(V); % run visim;

[x_obs,y_obs]=meshgrid(x,y);
d_obs=V.D(:, :, 1);
n_obs=prod(size(d_obs));

% CHOOSE SOME DATA FOR SEMIVARIOGRAM ANALYSIS
n_use=1000;
i_use=round(rand(1,n_use)*(n_obs-1))+1;
i_use=unique(i_use);

x_use=x_obs(i_use);
y_use=y_obs(i_use);
d_use=d_obs(i_use);

% PLOT DATA
figure(1);
```

```

imagesc(V.x,V.y,V.D(:,:,1));
title(visim_format_variogram(V))
axis image;
hold on
plot(x_use,y_use,'w.','MarkerSize',22)
scatter(x_use,y_use,20,d_use,'filled')
hold off
drawnow;

% SEMIVARIOGRAM ANALYSIS ISOTROPIC
[gamma_iso,hc,np,av_dist]=semivar_exp_gstat([x_use(:) y_use(:)],[d_use(:)]);
figure(2);
plot(hc,gamma_iso);
title('isotropic');xlabel('Distance');ylabel('\gamma')

% SEMIVARIOGRAM ANALYSIS ANISOTROPIC
ang_array=[0,22.5,45,67.5,90];
ang_tolerance=10;
for i_ang=1:length(ang_array);
    [gamma_an(:,i_ang),hc,np,av_dist]=semivar_exp_gstat([x_use(:) y_use(:)],[d_use(:)], ←
        ang_array(i_ang),ang_tolerance);
end
figure(3);
plot(hc,gamma_an);xlabel('Distance');ylabel('\gamma')
title('ANisotropic');
legend(num2str(ang_array'))

% SYNTHETICAL SEMIVARIOGRAM
gamma_synth=semivar_synth('0.0001 Nug(0) + 1 Sph(1)',hc);
figure(4)
plot(hc,gamma_an,'b-')
hold on
plot(hc,gamma_iso,'r-','linewidth',2)
plot(hc,gamma_synth,'k-','linewidth',2)
hold off
;xlabel('Distance');ylabel('\gamma')
legend(num2str(ang_array'))

```

More info at : Section [7.1.89](#)

## 3.3 GSTAT examples

### 3.3.1 GSTAT ex03

```

1 % ex03 : gstat example ex03.cmd
2 cmd_file='ex03';
3 [pred,pred_var,pred_covar,mask,G]=gstat(sprintf('%s.cmd',cmd_file));
4
5 [obs,obs_header,obs_title]=read_eas(G.data{1}.file);
6
7 figure(3);clf;
8 imagesc(mask.x,mask.y,pred(:,:,1));
9
10 hold on
11 plot(obs(:,1),obs(:,2),'k.','MarkerSize',10);
12 scatter(obs(:,1),obs(:,2),10,obs(:,3));
13 hold off
14 axis image

```



```

15 cb=colorbar;
16 set(get(cb,'Ylabel'),'string',obs_header{3})
17 title(sprintf('GSTAT %s.cmd - %s',cmd_file,obs_title))
18 xlabel(obs_header{1})
19 ylabel(obs_header{2})
20
21 print('-dpng',sprintf('%s.png',cmd_file))

```

### 3.3.2 GSTAT ex04

```

1 % ex04 : gstat example ex04.cmd
2 cmd_file='ex04';
3 [pred,pred_var,pred_covar,mask,G]=gstat(sprintf('%s.cmd',cmd_file));
4
5 [out,out_header,out_title]=read_eas(G.set.output);
6 [obs,obs_header,obs_title]=read_eas(G.data{1}.file);
7
8 figure(4);clf;
9 scatter(obs(:,1),obs(:,2),10,obs(:,3));
10 hold on
11 plot(out(:,1),out(:,2),'k.','MarkerSize',12);
12 scatter(out(:,1),out(:,2),10,out(:,3));
13 hold off
14 axis image
15 cb=colorbar;
16 set(get(cb,'Ylabel'),'string',obs_header{3})
17 title(sprintf('GSTAT %s.cmd - %s',cmd_file,obs_title))
18 xlabel(obs_header{1})
19 ylabel(obs_header{2})
20
21 print('-dpng',sprintf('%s.png',cmd_file))

```

### 3.3.3 GSTAT ex05

```

1 % ex05 : gstat example ex05.cmd
2 cmd_file='ex05';
3 [pred,pred_var,pred_covar,mask,G]=gstat(sprintf('%s.cmd',cmd_file));
4
5 [obs,obs_header,obs_title]=read_eas(G.data{1}.file);
6
7 if (isfield(G.data{1},'log')); pred=exp(pred);end
8
9 figure(5);clf;
10 subplot(1,2,1);
11 imagesc(mask.x,mask.y,pred(:,:,1));
12 hold on
13 plot(obs(:,1),obs(:,2),'k.','MarkerSize',12);
14 scatter(obs(:,1),obs(:,2),10,obs(:,3));
15 hold off
16 axis image
17 cb=colorbar;
18 set(get(cb,'Ylabel'),'string',obs_header{3})
19 xlabel(obs_header{1})
20 ylabel(obs_header{2})
21 title('Mean')
22
23 subplot(1,2,2);
24 imagesc(mask.x,mask.y,pred_var(:,:,1));

```

```

25 colorbar
26 hold on
27 plot(obs(:,1),obs(:,2),'k.','MarkerSize',12);
28 hold off
29 axis image
30 %cb=colorbar;
31 %set(get(cb,'Ylabel'),'string',obs_header{3})
32 xlabel(obs_header{1})
33 ylabel(obs_header{2})
34 title('Variance')
35
36 watermark(sprintf('GSTAT %s.cmd - %s',cmd_file,G.mgstat.comment{2}));
37
38 print('-dpng',sprintf('%s.png',cmd_file))

```

### 3.3.4 GSTAT ex06

```

% ex06 : gstat example ex06.cmd
cmd_file='ex06';
[pred,pred_var,pred_covar,mask,G]=gstat(sprintf('%s.cmd',cmd_file));
figure(6),clf;
imagesc(mask.x,mask.y,pred(:, :, 1));
cb=colorbar;
axis image
title('Unconditional Gaussian simulation')
watermark(sprintf('GSTAT %s.cmd - %s',cmd_file,G.mgstat.comment{2}));

print('-dpng',sprintf('%s.png',cmd_file))

```

### 3.3.5 GSTAT ex07

```

1 % ex07 : gstat example ex07.cmd
2 cmd_file='ex07';
3 [pred,pred_var,pred_covar,mask,G]=gstat(sprintf('%s.cmd',cmd_file));
4
5
6 [obs,obs_header,obs_title]=read_eas(G.data{1}.file);
7
8 if (isfield(G.data{1},'log')); pred=exp(pred);end
9
10
11
12 figure(7);clf;
13 imagesc(mask.x,mask.y,pred(:, :, 1));
14
15 hold on
16 plot(obs(:,1),obs(:,2),'k.','MarkerSize',12);
17 scatter(obs(:,1),obs(:,2),10,obs(:,3));
18 hold off
19 axis image
20 cb=colorbar;
21 set(get(cb,'Ylabel'),'string',obs_header{3})
22 xlabel(obs_header{1})
23 ylabel(obs_header{2})
24
25
26 cb=colorbar;
27 axis image

```

```

28 title('Unconditional Gaussian simulation')
29 watermark(sprintf('GSTAT %s.cmd - %s',cmd_file,G.mgstat.comment{2}));
30
31 print('-dpng',sprintf('%s.png',cmd_file))

```

### 3.3.6 GSTAT ex09

```

1 % ex09 : gstat example ex09.cmd
2 cmd_file='ex09';
3 disp(sprintf('%s : There (seems to be) a bug in GSTAT setting NCOLUMNS = 6, when there is ←
   only 5 columes in the output EAS file',mfilename));
4 [pred,pred_var,pred_covar,mask,G]=gstat(sprintf('%s.cmd',cmd_file));
5

```

### 3.3.7 GSTAT ex10

```

1 cmd_file='ex10';
2 [pred,pred_var,pred_covar,mask,G]=gstat(sprintf('%s.cmd',cmd_file));
3
4
5 [obs1,obs_header1,obs_title1]=read_eas(G.data{1}.file);
6 [obs2,obs_header2,obs_title2]=read_eas(G.data{2}.file);
7
8 for i=1:2;
9     if (isfield(G.data{i},'log')); pred(:, :, i)=exp(pred(:, :, i));end
10 end
11
12 clf;
13 subplot(2,2,1);
14 imagesc(mask.x,mask.y,pred(:, :, 1));axis image;title([G.predictions{1}.data,' mean estimate' ←
   ])
15 hold on
16 plot(obs1(:,1),obs1(:,2),'k.','MarkerSize',12);
17 scatter(obs1(:,1),obs1(:,2),10,obs1(:,G.data{1}.v),'filled');
18 hold off
19 axis image
20 cb=colorbar;
21 set(get(cb,'Ylabel'),'string',obs_header1{3})
22 xlabel(obs_header1{1})
23 ylabel(obs_header1{2})
24
25 subplot(2,2,2);
26 imagesc(mask.x,mask.y,pred(:, :, 2));axis image;title([G.predictions{2}.data,' mean estimate' ←
   ])
27 hold on
28 plot(obs2(:,1),obs2(:,2),'k.','MarkerSize',12);
29 scatter(obs2(:,1),obs2(:,2),10,obs1(:,G.data{2}.v),'filled');
30 hold off
31 axis image
32 cb=colorbar;
33 set(get(cb,'Ylabel'),'string',obs_header2{3})
34 xlabel(obs_header2{1})
35 ylabel(obs_header2{2})
36
37 subplot(2,2,3);
38 imagesc(mask.x,mask.y,pred_var(:, :, 1));axis image;
39 colorbar
40 title([G.predictions{1}.data,' variance estimate'])

```

```
41
42 subplot(2,2,4);
43 imagesc(mask.x,mask.y,pred_var(:,:,2));axis image;
44 colorbar
45 title([G.predictions{1}.data,' variance estimate'])
46
47
48 watermark(sprintf('GSTAT %s.cmd - %s',cmd_file,G.mgstat.comment{2}));
49
50 print('-dpng',sprintf('%s',cmd_file))
```

---

## Chapter 4

# VISIM - Direct and Gaussian sequential simulation

This chapter discuss how to run VISIM from within Matlab.

### 4.1 VISIM

VISIM is a sequential simulation code based on GSLIB ('Geostatistical Software LIBrary', [Stanford Center for Reservoir Forecasting, Stanford University](#)) for sequential Gaussian and direct sequential simulation with histogram reproduction. In addition to classical simulation based on noise-free hard data of point support, VISIM also handles noisy data of mixed support, enabling linear stochastic inversion using the sequential simulation approach.

VISIM can be used to generate samples of the a posteriori distribution of a linear inverse problem.

For detailed information the usage of VISIM visit the [VISIM homepage at IMGP](#). [[user manual pdf](#)].

References to the use of VISIM should be made to :

Hansen, T. M., and Mosegaard, K., 2008. VISIM : Sequential simulation for linear inverse problems. Computers and Geosciences, 34(1), pp 53-76. [[doi:10.1016/j.cageo.2007.02.003](#)].

#### 4.1.1 Running VISIM from Matlab

The three main files to control VISIM from Matlab are Section [7.2.4](#) Section [7.2.56](#) Section [7.2.6](#)

##### 4.1.1.1 Check path to VISIM

When running Section [7.2.6](#) an attempt is made to look for a VISIM binary. First the 'bin' sub folder where mGstat is installed is searched, then the system path is searched for 'visim' binary file. By running Section [7.2.6](#) with no arguments the path to the use binary (if located) is listed:

```
visim;  
Using VISIM binary : ~/bin/visim
```

If the binary for VISIM is not found you have to edit the 'visim.m' m-file and manually give set the path in the top of the file.

#### 4.1.1.2 Running VISIM from Matlab

Section 7.2.6 is used to run VISIM from within Matlab. The input is either a VISIM parameter file (as a string) or a VISIM Matlab structure, as obtained using Section 7.2.4.

Therefore the following two approaches will run VISIM on the `visim_sgsim.par`:

```
V = visim('visim_sgsim.par');
```

```
V = visim('visim_sgsim.par');
V = visim(V);
```

which will result in the following out to the screen :

```
VISIM Version: 1.000

filename is sgsim_uncond.par
sgsim_uncond.par
Initializing data2vol covar lookup table
Initializing vol2vol covar lookup table
Working on realization number 1
Working on realization number 2
Working on realization number 3
...
Working on realization number 100

V =

    parfile: 'sgsim_uncond.par'
    cond_sim: 0
    fconddata: [1x1 struct]
         cols: [6x1 double]
    fvolgeom: [1x1 struct]
    fvolsum: [1x1 struct]
    trimlimits: [2x1 double]
    debuglevel: -11
         out: [1x1 struct]
        nsim: 100
        ccdf: 0
    refhist: [1x1 struct]
         nx: 21
        xmn: 0.1250
       xsiz: 0.2500
         ny: 49
        ymn: 0.1250
       ysiz: 0.2500
         nz: 1
        zmn: 0.1250
       zsiz: 0.2500
         x: [1x21 double]
         y: [1x49 double]
         z: 0.1250
       rseed: 69068
    minorig: 0
    maxorig: 1
    nsimdata: 28
    volnh: [1x1 struct]
    densitypr: 0
    assign_to_nodes: 1
    max_data_per_octant: 0
    search_radius: [1x1 struct]
```

```

search_angle: [1x1 struct]
gmean: 0.1304
gvar: 2.0000e-04
Va: [1x1 struct]
tail: [1x1 struct]
D: [21x49x7 double]
etype: [1x1 struct]

```

The output of running visim is the Matlab structure listed above, that contains all parameters of the VISIM used parameter file, as well as the simulated/estimated data `V.D`, and the E-type mean and variance as `V.etype.mean` and `V.etype.var`

An example of running 9 runs of VISIM while changing the primary anisotropy direction from 0 to 90 is the following

```

Vorig = read_visim('visim_sgsim.par');
for i=1:4
    V{i}=Vorig;
    V{i}.Va.ang1=i*10
    V{i}=visim(V{i});
end

```

Each of the Matlab structures `V{1}, V{2}, ..., V{9}` now contain the simulation result.

## 4.1.2 Plotting VISIM realizations

To plot a number of generated realizations use Section 7.2.6.

The simplest use of Section 7.2.6 is simple to specify a number of realizations to plot.

```

V=read_visim('sgsim_cond_1.par');
visim_plot_sim(V,2)

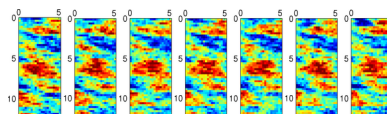
```

An example of using all the options available for Section 7.2.6

```

nsim=7;      % number of realizations to plot
cax=[.1 .16]; % scaling for colorbar axis
FS = 10;     % FontSize
nxsub=7;     % Number of subplots in the X-direction
nzsub=7;     % Number of subplots in the z-direction
visim_plot_sim(V,nsim,cax,FS,nxsub,nzsub);

```



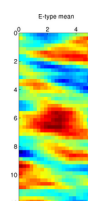
### 4.1.2.1 Plotting VISIM E-type

In case VISIM is run in simulation mode, `visim_plot_etype` plots the E-type (the point wise mean and/or variance) of all generated simulations.

```

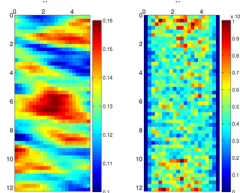
cax=[.1 .16]; % scaling for colorbar axis
visim_plot_etype('sgsim_cond_1.par');

```



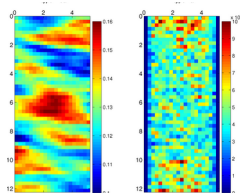
The following will plot both the E-type mean and variance.

```
V=read_visim('sgsim_cond_1.par');
plotAll=1;
visim_plot_etype(V,plotAll);
```



The following will plot both the E-type mean and variance with the specified colorscales.

```
V=read_visim('sgsim_cond_1.par');
cax=[.1 .16]; % scaling for colorbar axis E-type Mean
cax_var=[0 1e-4]; % scaling for colorbar axis E-type variance
plotAll=1;
visim_plot_etype(V,plotAll,cax,cax_var);
```

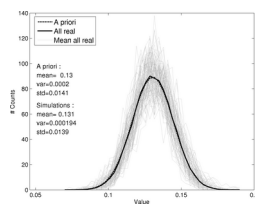


See more info at Section [7.2.20](#)

#### 4.1.2.2 Plotting VISIM histogram

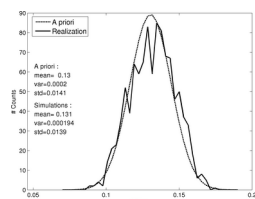
To plot the histogram for all realizations, compared to the a priori chosen one use `visim_plot_hist`:

```
V=read_visim('sgsim_uncond.par');
visim_plot_hist(V);
```



and to plot just the histogram of one simulation use

```
V=read_visim('sgsim_uncond.par');
visim_plot_hist(V,10);
```



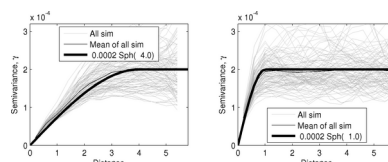
See more info at Section [7.2.21](#)



### 4.1.2.3 Plotting VISIM semivariogram

To plot the experimental semivariograms compared to the a priori chosen semivariogram, use `visim_plot_semivar_real` :

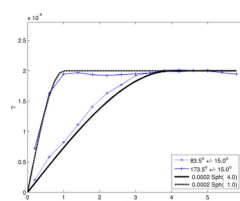
```
V=read_visim('sgsim_uncond.par');
visim_plot_semivar_real(V);
```



See more info at Section [7.2.29](#)

To compare the mean experimental computed for a number of realizations to the a priori chosen semivariogram, use

```
V=read_visim('sgsim_uncond.par');
visim_plot_semivar(V,1:V.nsim);
```



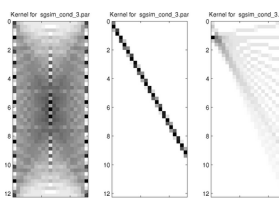
See more info at Section [7.2.25](#)

### 4.1.2.4 Plotting the sensitivity kernel

The sensitivity kernel for all or a specific selection of volume average data can be plotted using :

```
subplot(1,3,1)
visim_plot_kernel('sgsim_cond_3.par');
subplot(1,3,2)
visim_plot_kernel('sgsim_cond_3.par',11);
subplot(1,3,3)
visim_plot_kernel('sgsim_cond_3.par',11:25);
colormap(1-gray)
```

This will result in the following :



See more info at Section [7.2.22](#)

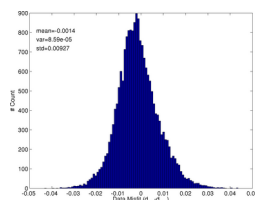
### 4.1.2.5 Plotting VISIM conditional lookup table

See more info at Section [7.2.19](#)

#### 4.1.2.6 Plotting the distribution of the prediction error of the used volume average data

To plot the distribution of the volume average estimates as calculated from the realization and the observed volume average values use:

```
V=read_visim('sgsim_cond_1.par');
visim_plot_volfit(V);
```



See more info at Section [7.2.32](#)

## 4.2 VISIM examples

Here follows a few examples for problems that can be addressed using VISIM

### 4.2.1 visim\_init

`visim_init` generate a reference data structure (equivalent of a VISIM parameter file) for unconditional simulation:

```
V=visim_init;
V=visim(V);
visim_plot_sim(V);
```

`visim_init` can optionally be called with a suggested geometry:

```
V=visim_init(1:1:50,1:2:200);
V=visim(V);
visim_plot_sim(V);
```

Finally, `visim_init` an existing VISIM structure can be passed, that will form the base of an update VISIM structure:

```
V=visim_init(1:1:50,1:2:200,V);
V=visim(V);
visim_plot_sim(V);
```

When `visim_init` is called, the global variance `V.gvar` is checked for consistency with the semivariogram model, and fixed appropriately such that the global variance is set to the total sill value of the specified semivariogram. The tail values will also be checked for consistency if `V.ccdf=1`.

### 4.2.2 Direct Sequential Simulation

If `V.ccdf=1`, direct sequential simulation is performed. This enable the use of a non-Gaussian distribution of the subsurface parameters (the target distribution). The shape of the target distribution is given by the data found in the file `V.refhist.fname`. The target distribution can be treated as either a continuous or a discrete function using `V.refhist.do_discrete`.

The following example will perform unconditional simulation with a trimodal target distribution assumed to be continuous:

```

1 V=visim_init(1:1:61,1:1:61);
2 V.parfile='visim_example_dssim_cont_1.par';
3 d1=randn(1,2000)*sqrt(.2)+8;
4 d2=randn(1,1200)*sqrt(.1)+10;
5 d3=randn(1,2800)*sqrt(.1)+13;
6 d_target=[d1,d2,d3]';
7 V.refhist.fname='dssim_target.eas';
8 write_eas(V.refhist.fname,d_target); % write target distribution
9 V.ccdf=1; % use DSSIM
10 V.refhist.do_discrete=0; % Assume continious target histogram
11 V.nsim=10;
12 V=visim_init(V);
13 V.Va.it=3; % Choose Gaussian semivariogram
14 V.Va.a_hmax=15; % correlation length (direction of max continuity)
15 V.Va.a_hmin=15; % correlation length (direction of min continuity)
16 V=visim(V);
17 figure(1);visim_plot_sim(V);
18 print_mul('visim_example_dssim_cont_sim');
19 figure(2);visim_plot_hist(V);
20 print_mul('visim_example_dssim_cont_hist');
21
22

```

The following example will perform unconditional simulation with a target distribution chosen as [1 10 10 40], assmued to be continious:

```

1 V=visim_init(1:1:61,1:1:61);
2 V.parfile='visim_example_dssim_cont_2.par';
3 d_target=[1 10 10 40]';
4 V.refhist.fname='dssim_target_discrete.eas';
5 write_eas(V.refhist.fname,d_target); % write target distribution
6 V.ccdf=1; % use DSSIM
7 V.refhist.do_discrete=0; % Assume continious target histogram
8 V.nsim=10;
9 V=visim_init(V);
10 V.Va.it=3; % Choose Gaussian semivariogram
11 V.Va.a_hmax=15; % correlation length (direction of max continuity)
12 V.Va.a_hmin=15; % correlation length (direction of min continuity)
13 V=visim(V);
14
15 figure(1);visim_plot_sim(V);
16 print_mul('visim_example_dssim_cont_1_sim');
17 figure(2);visim_plot_hist(V);
18 print_mul('visim_example_dssim_cont_1_hist');
19
20

```

The last DSSIM example is a example of unconditional simulation with a target distribution chosen as [1 10 10 40], assmued to be discrete. This means that no other values than the ones given in the target distirbution can be simulated. This can be used for categorical simulation:

```

1 V=visim_init(1:1:61,1:1:61);
2 V.parfile='visim_example_dssim_discrete_1.par';
3 d_target=[1 10 10 40]';
4 V.refhist.fname='dssim_target_discrete.eas';
5 write_eas(V.refhist.fname,d_target); % write target distribution
6 V.ccdf=1; % use DSSIM
7 V.refhist.do_discrete=1; % Assume continious target histogram
8 V.nsim=10;
9 V=visim_init(V);
10 V.Va.it=3; % Choose Gaussian semivariogram

```

```

11 V.Va.a_hmax=15; % correlation length (direction of max continuity)
12 V.Va.a_hmin=15; % correlation length (direction of min continuity)
13 V=visim(V);
14
15 figure(1);visim_plot_sim(V);
16 print_mul('visim_example_dssim_discrete_1_sim');
17 figure(2);visim_plot_hist(V);
18 print_mul('visim_example_dssim_discrete_1_hist');
19
20

```

### 4.2.3 Correlated data errors

Correlated data errors can be specified as an EAS formatted ascii file with the name `datacov_[V.fout]`. If for example the output file for `visim.par` is set to `visim.out`, then the file with correlated data errors should be name `datacov_visim.out`. If this file exist, correlated data errors are read from the file appropriately. If the file does not exist, UNcorrelated data errors are assumed, as given by the fourth column of the `V.fvolsum.fname` file.

### 4.2.4 Simulation of linear inverse problems

VISIM can be used to solve any linear inverse problem, using both estimation and simulation. The function `G_to_visim` converts a linear inverse problem in Matlab format to a visim format:

```

load lsq_example.mat
V=G_to_visim(x,y,z,d_obs,G,Cd,m0);

% MAKE SURE THE KERNEL LOOKS OK
figure;
subplot(1,2,1);visim_plot_kernel(V); % All kernels
subplot(1,2,2);visim_plot_kernel(V,2); % the 2nd kernel

V.volnh.max=200; % MAX NUMBER OF VOLUMES TO USE
V.Va.a_hmax=0;
V.Va.a_hmin=0;

% Estimation using VISIM
V.nsim=0;
Vest=visim(V);

% Simulation using Matlab
V.nsim=10;
Vsim=visim(V);

% Estimation using Matlab
m0=zeros(V.nx*V.ny,1)+m0;
d0=G*m0;
Cm=V.gvar.*eye(V.nx*V.ny);
m_est = m0 +Cm*G'*inv(G*Cm*G' + Cd)*(d_obs-d0);
Cm_est = Cm - Cm*G'*inv( G*Cm*G' + Cd )*G*Cm;

subplot(2,3,1);
imagesc(V.x,V.y,reshape(m_est,V.ny,V.nx));axis image
cax=caxis;
title_alt('LSQ mean',1)
subplot(2,3,2);
imagesc(V.x,V.y,Vest.etype.mean');axis image;caxis(cax)
title_alt('VISIM LSQ mean',2)
subplot(2,3,3);

```

```

imagesc(V.x,V.y,Vsim.etype.mean');axis image;caxis(cax)
title_alt('VISIM Etype mean',3)

subplot(2,3,4);
imagesc(V.x,V.y,reshape(diag(Cm_est),V.ny,V.nx));axis image
cax=caxis;
title_alt('LSQ var',4)
subplot(2,3,5);
imagesc(V.x,V.y,Vest.etype.var');axis image;caxis(cax)
title_alt('VISIM LSQ var',5)
subplot(2,3,6);
imagesc(V.x,V.y,Vsim.etype.var');axis image;caxis(cax)
title_alt('VISIM LSQ etype',6)

```

## 4.2.5 Cross borehole tomography example

As VISIM conditions to linear average measurements of the model parameter space, VISIM can be used to draw samples from the a posteriori probability distribution linear inverse problems.

### 4.2.5.1 Conditional simulation through error simulation

Conditional simulation through error simulation is a fast alternative to traditional conditional sequential Gaussian simulation, Journel and Huijbregts (1978) page 495. See the following reference for more details: Hansen, T. M. and Mosegaard, K. : VISIM : Sequential simulation for linear inverse problems, Computers and Geosciences 2007, doi:10.016/j.cageo.2007.02.003.

```

V=read_visim('visim_sgsim_cond_3.par');

V.nsim=100;

% Traditional conditional Gaussian simulation
Vseqsim=visim(V);
% Conditional simulation through error simulation
Verrsim=visim_errsim(V);

```

### 4.2.5.2 Calculating averaging kernels for cross borehole tomography

To deal with any kind of linear inverse problem in VISIM the averaging kernel describing the forward problem must be given

The averaging kernel is defined in two text files `visim_volgeom.eas` (defining the geometry of each average kernel) and `visim_volsum.eas`. (giving the observed average and measurement error for each defined volume average.

These two files can be calculated for tomography problems with arbitrary source-receiver geometry, and for a specific velocity model. Both the high frequency approximation to the wave equation, rays, and Fresnel zone based kernels can be generated using the `visim_setup_punch`

```

V=read_visim('sgsim_reference.par');
nx=V.nx;
ny=V.ny;
nz=V.nz;

m_ref=read_eas('visim_sgsim_refmod.eas');
m_ref=reshape(m_ref(:,3),ny,nx)';
% m_ref=zeros(nx,ny)+0.13;

S=linspace(1,11.5,7);
R=linspace(2,10,7);

[ss,rr]=meshgrid(S,R);

```

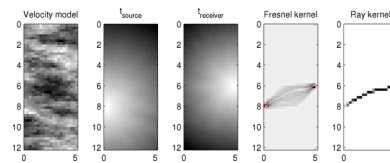
```

S=[ss(:).*0+1.1 ss(:)];
R=[rr(:).*0+4.9 rr(:)];

type=1; % [1]: Ray approximation, [2]: Fresnel zone based kernels
doPlot=1;% [0]: No visual progress, [1]: Plot all kernels as they are computed.

% CALCULATE GEOMETRY IN REF MODEL
[V_ray,G_ray]=visim_setup_punch(V,S,R,m_ref,[],[],'ref_rays',type,doPlot);
type=2; % [1]: Ray approximation, [2]: Fresnel zone based kernels
[V_fre,G_fre]=visim_setup_punch(V,S,R,m_ref,[],[],'ref_frechet',type,doPlot);

```



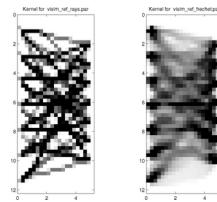
See more info at Section [7.2.22](#)

To visualize the generated averaging kernels use for example :

```

subplot(1,2,1)
visim_plot_kernel(V_ray);caxis([0 .1])
subplot(1,2,2)
visim_plot_kernel(V_fre);caxis([0 .1])

```



Now we compute the observed travel time from the reference model and the current ray geometry

```

t_obs=G_ray*m_ref(:);
t_err=0.*t_obs.*.01;
[V_tomo]=visim_setup_punch(V,S,R,m_ref,t_obs,t_err,'tomo_ray',type,doPlot);

```

This generates the filename `visim_tomo_ray.par`. This parameter file can now be used to perform estimation and/or simulation :

```

V_tomo.cond_sim=3; % condition to volume averages only (no point data)
V_tomo.nsim=100; % 100 realizations
V_tomo.volh.max=100; % max 100 average data in the neighborhood.
V_tomo=visim(V_tomo);

```

#### 4.2.5.3 Linear inversion : Cross borehole tomography example

## Chapter 5

# SNESIM - Single Normal Equation SIMulation

This chapter discuss how to run SNESIM from within Matlab. SNESIM is original Fortran code developed for Single Normal Equation SIMulation method developed by Sebastian Strebelle.

This fortran based implementation has been superceeded by an implementaion is available through S-GeMS. See the chapter [?title].

### 5.1 Working with SNESIM and mGstat

mGstat implements 4 m-files that allow interactions with SNESIM Here we make no effort to explain the meaning of all the options for the snesim parameter file, but refer to the documentation for SNESIM

#### 5.1.1 Working with SNESIM and mGstat

Get a copy of a snesim parameter file, as well as the associated training image and data templates using “**snesim\_init**”:

```
S = snesim_init;
S =

    fconddata: [1x1 struct]
    ncat: 2
    cat_code: [0 1]
    pdf_target: [0.7000 0.3000]
    use_vert_prop: 0
    fvertprob: [1x1 struct]
    pdf_target_repro: 1
    pdf_target_par: 0.5000
    debug_level: -2
    fdebug: [1x1 struct]
    out: [1x1 struct]
    nsim: 1
    nx: 80
    xmn: 0.2500
    xsiz: 0.5000
    ny: 120
    ymn: 0.2500
    ysiz: 0.5000
    nz: 1
    zmn: 0.2500
    zsiz: 0.5000
    rseed: 500
    ftemplate: [1x1 struct]
```

```

        max_cond: 16
    max_data_per_oct: 0
    max_data_events: 20
    n_mulgrids: 2
    n_mulgrids_w_stree: 1
        fti: [1x1 struct]
    nxtr: 250
    nytr: 250
    nztr: 1
    hmax: 10
    hmin: 10
    hvert: 5
    amax: 7
    amin: 3
    avert: 0
    parfile: 'snesim.par'

```

This will read the snesim parameter file into a Matlab structure that can be easily altered.

This SNESIM S structure can be written to a SNESIM parameter files using : “**write\_snesim**”:

```
write_snesim(S,'snesim.par');
```

A SNESIM parameter file can be read into a Matlab structure S, using: “**read\_snesim**”:

```
S=read_snesim('snesim.par');
```

SNESIM can be from Matlab using: “**snesim**”:

```

S = snesim_init;
S.nsim = 10;
S = snesim(S);

```

The output from running snesim is located in the data structure as S.D. In the present case:

```

size(S.D)
80 120 10

```

Thus to plot the realizations one could use

```

for i=1:S.nsim
    subplot(4,3,i);
    imagesc(S.x,S.y,S.D(:, :, i));
    axis image
end

```



## Chapter 6

# S-GeMS - The Stanford Geostatistical Modeling Software

This chapter discuss how to run S-GeMS from within Matlab.

### 6.1 Install S-GeMS

See Section 1.2.4 on details how to install S-GeMS on Windows and Linux.

### 6.2 S-GeMS data format

S-GeMS handles two data formats: The classical ASCII GEOEAS format, that has been widely used in the geostatistical community. In addition, S-GeMS make use of a new in BINARY format. The binary format is much faster to work with, and handles both point set data and grid data, will full description of the grid properties (cell size, origin, grid size).

In order to run S-GeMS interactively from Matlab only the binary format can be used, as there is no way to instruct S-GeMS about grid size properties reading a EAS file.

Reading and writing of the GEOEAS format are done using the Section 7.1.76 and Section 7.1.104 function.

Binary S-GeMS formatted data (both point set and grid data) can be read using the Section 7.4.13 function.

Binary point set data can be written using the Section 7.4.20 function, and binary grid data can be written using the Section 7.4.19 function, and

#### 6.2.1 GEOEAS to S-GeMS

EAS files can be converted to S-GeMS-binary formatted files using Section 7.4.1.

##### 6.2.1.1 GEOEAS Point Set to S-GeMS-binary

An EAS with data formatted as a point-set, the data section starts with 'ndim' columns defining the location in ndim-space, followed by N columns of DATA.

Use the following syntax:

```
O=eas2sgems(file_eas,file_sgems,ndim);
```

Convert a 3D EAS file with two data sets (5 cols, 3 dimensions) using

```
ndim=3
eas2sgems('file.eas','file.sgems',ndim)
```

Convert a 2D EAS file with two data sets (4 cols, 2 dimensions) using

```
ndim=2
eas2sgems('file.eas','file.sgems',ndim)
```

### 6.2.1.2 GEOEAS GRID to S-GeMS-binary

For an EAS with data formatted as GRIDS, the data section consist of N columns, representing N grids. An EAS not does not contain information about the cell size (dx,dy,dx) cell size, or the location of the first cell for each dimension (x0,y0,z0).

It 'may' (not part of strict format) contain information about the size of the grid(s) in the first line 'xxxxx (90x10x1)'.

Use the following syntax:

```
O=eas2sgems(file_eas,file_sgems,nx,ny,nz,dx,dy,dz,x0,y0,z0);
```

Convert an EAS file with 2 grids, assuming the grid size is given in the EAS header ('HEADER (60x70x1)'), and(dx,dy,dz)=(1,1,1), (x0,y0,z0)=(0,0,0):

```
ndim=2
eas2sgems('file.eas','file.sgems')
```

Same as above, but all manual settings:

```
eas2sgems('file.eas','file.sgems',60,70,1,1,1,1,0,0,0);
```

Same as above, but but (x0,y0,z0)=(10,10,6):

```
eas2sgems('file.eas','file.sgems',60,70,1,10,10,6,0,0,0);
```

### 6.2.2 S-GeMS to EAS

S-GeMS-binary formatted files can be converted to EAS ASCII formatted files using Section 7.4.3. Simply call :

```
sgems2eas('file.sgems','file.eas');
```

## 6.3 Using S-GeMS

In order to make full use of the Matlab interface to S-GeMS some knowledge of the use of S-GeMS is essential. The book Applied Geostatistics with SGeMS (Remy, Boucher and Wu, Cambridge University Press, 2009), written by the developers of S-GeMS is highly recommended.

The Matlab interface to S-GeMS relies on a feature of S-GeMS that allow S-GeMS to read and execute a series of Python commands from the command line, without the need to load the graphical user interface, as for example:

```
sgems -s sgems_python_script.py
```

The Matlab interface consists of methods and functions to automatically create such a Python script, execute the script using S-GeMS and load the simulated/estimated results into Matlab

One function (Section 7.4.7) handles these actions allowing simulation on grids in the following manner:

1. Define a parameter file (Section 7.4.6, Section 7.4.15)

2. Write a python script that (Section 7.4.8)
  - (a) sets up a grid or pointset where simulation or estimation is performed
  - (b) performs the simulation/estimation
  - (c) export the simulated/estimated data
3. Load the data into Matlab (Section 7.4.18)

For a complete list of S-GeMS related commands on mGstat see Section 7.4

### 6.3.1 Sequential simulation using S-GeMS

This section contains a rather detailed explanation of using S-GeMS to perform simulation. Much more compact example can be found in the following chapters.

Unconditional and conditional sequential simulation can be performed using Section 7.4.7 :

```
S = sgems_grid(S);
```

Where *S* is a Matlab data structure containing all the information needed to setup and run S-GeMS

A number of different simulation algorithms are available in S-GeMS The behavior of each algorithm is controlled through an XML file. Such an XML file can for example be exported from S-GeMS by choosing to save a parameter file for a specific algorithm.

Such an XML formatted parameter is needed to perform any kind of simulation. A number of 'default' parameter files available using the Section 7.4.6 function. For example to obtain a default parameter file for sequential Gaussian simulation use

```
S = sgems_get_par('sgsim')

S =

    xml_file: 'sgsim.par'
      XML: [1x1 struct]
```

As can be seen the adds the name of the XML file (*S.xml\_file*) as well as a XML data structure in the S-GeMS matlab structure *S.XML*.

All supported simulation/estimation types can be found calling *sgems\_get\_par* without arguments:

```
1 >> sgems_get_par
2 sgems_get_par : available SGeMS type dssim
3 sgems_get_par : available SGeMS type filtersim_cate
4 sgems_get_par : available SGeMS type filtersim_cont
5 sgems_get_par : available SGeMS type lusim
6 sgems_get_par : available SGeMS type sgsim
7 sgems_get_par : available SGeMS type snesim_std
```

Now all parameters for 'sgsim' simulation can be set directly from the Matlab command line. To see the number of fields in the XML file (refer to the S-GeMS book described above for the meaning of all parameters):

```
>> S.XML.parameters

ans =

    algorithm: [1x1 struct]
   Grid_Name: [1x1 struct]
Property_Name: [1x1 struct]
Nb_Realizations: [1x1 struct]
      Seed: [1x1 struct]
```

```

        Kriging_Type: [1x1 struct]
        Trend: [1x1 struct]
    Local_Mean_Property: [1x1 struct]
        Assign_Hard_Data: [1x1 struct]
        Hard_Data: [1x1 struct]
    Max_Conditioning_Data: [1x1 struct]
        Search_Ellipsoid: [1x1 struct]
    Use_Target_Histogram: [1x1 struct]
        nonParamCdf: [1x1 struct]
        Variogram: [1x1 struct]

```

To see the number of realization :

```

>> S.XML.parameters.Nb_Realizations

ans =

    value: 10

```

To set the number of realization to 20 do:

```

>> S.XML.parameters.Nb_Realizations.value=20;

```

One also need to define the grid used for simulation. This is done through the `S.dim` data structure:

```

%grid size
S.dim.nx=70;
S.dim.ny=60;
S.dim.nz=1;
% grid cell size
S.dim.dx=1;
S.dim.dy=1;
S.dim.dz=1;
% grid origin
S.dim.x0=0;
S.dim.y0=0;
S.dim.z0=0;

```

All the values listed above for the `S.dim` data structure are default, thus if they are not set, they are assumed as listed.

Unconditional simulation is now performed using:

```

>> S=sgems_grid(S);
sgems_grid : Trying to run SGeMS using sgsm.py, output to SGSIM.out
'import site' failed; use -v for traceback
Executing script...

working on realization 1
|#          |      5%working on realization 2
|##         |     10%working on realization 3
|###        |     15%working on realization 4
|####       |     20%working on realization 5
|#####     |     25%working on realization 6
|#####    |     30%working on realization 7
|#####   |     35%working on realization 8
|#####  |     40%working on realization 9
|##### |     45%working on realization 10
|##### |     50%working on realization 11
|##### |     55%working on realization 12
|##### |     60%working on realization 13
|##### |     65%working on realization 14
|##### |     70%working on realization 15

```

```

|#####| 75%working on realization 16
|#####| 80%working on realization 17
|#####| 85%working on realization 18
|#####| 90%working on realization 19
|#####| 95%working on realization 20
|#####| 100%
sgems_read : Reading GRID data from SGSIM.sgems
sgems_grid : SGeMS ran successfully

S =

    xml_file: 'sgsim.par'
         XML: [1x1 struct]
         dim: [1x1 struct]
        data: [4200x20 double]
           O: [1x1 struct]
           x: [1x70 double]
           y: [1x60 double]
           z: 1
           D: [4-D double]

```

As seen above the following field have been added to the S-GeMS matlab structure: `S.x`, `S.y`, `S.z`, `S.data` and `S.D`.

`S.x`, `S.y`, `S.z` are 3 arrays defining the grid.

`S.data`, is the simulated data as exported from S-GeMS. Note the each realization is returned as a list of size `nx*ny*nz`.

`S.D`, is but a rearrangement of `S.data` into a 4D dimensional data structure, of size `(nx,ny,nz,nsim)`. To visualize for example the 3rd realization use for example:

```
imagesc(S.x,S.y,S.D(:,:,1,3));
```

Conditional simulation can be performed by setting the `S.d_obs` parameter. For example:

```

S.d_obs=[18 13 0 0; 5 5 0 1; 2 28 0 1];
S=sgems_grid(S);
imagesc(S.x,S.y,S.D(:,:,1,3));

```

### 6.3.1.1 Specification of variogram model

Using sequential Gaussian simulation the semivariogram model is specified in `S.XML.parameters.Variogram`:

```

>> S.XML.parameters.Variogram
ans=
    nugget: 1.0000e-003
structures_count: 1
    structure_1: [1x1 struct]

```

To run 10 simulations with increasing range do for example:

```

for i=1:1:10
    r=i*10;
    S.XML.parameters.Variogram.structure_1.ranges.max=[r];
    S.XML.parameters.Variogram.structure_1.ranges.medium=[r];
    S.XML.parameters.Variogram.structure_1.ranges.min=[r];
    S=sgems_grid(S);
    subplot(4,3,i);imagesc(S.x,S.y,S.D(:,:,1)');
end

```

The variogram model can also be specified using a shorter notation (same format as when using GSTAT):

```

S.XML.parameters.Variogram=sgems_variogram_xml('0.1 Nug(0) + 0.4 Exp(10) + 0.5 Sph ←
(40,30,0.2)');

```

### 6.3.2 Unconditional sequential Gaussian simulation using S-GeMS

A simple example of unconditional sequential simulation.

```
S=sgems_get_par('sgsim');
S.XML.parameters.Nb_Realizations.value=12;
S=sgems_grid(S);
for i=1:S.XML.parameters.Nb_Realizations.value;
    subplot(4,3,i);
    imagesc(S.x,S.y,S.D(:,:,1,i));
end
```

### 6.3.3 Conditional sequential Gaussian simulation

Conditioning data can be specified either as a data variable or as an sgems-binary formatted file (see Section 6.2).

#### 6.3.3.1 conditional data as a variable

A simple example of conditional sequential simulation (examples/sgems\_examples/sgems\_example\_sgsm\_conditional.m):

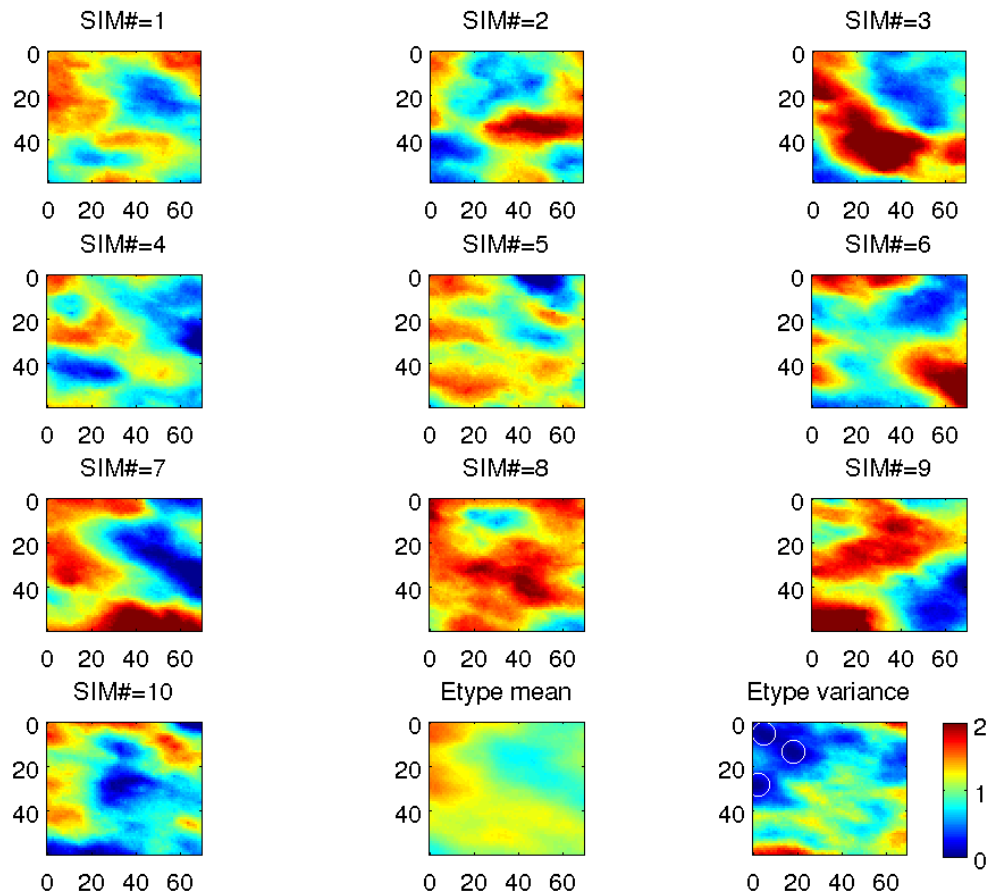
```
1 % sgems_sgsm_conditional :
2 %     Conditional SGSIM using hard data from variable
3
4 % GET Default par file
5 S=sgems_get_par('sgsim');
6
7 % Define observed data=
8 S.d_obs=[18 13 0 0; 5 5 0 1; 2 28 0 1];
9
10 S.XML.parameters.Nb_Realizations.value=10;
11 S=sgems_grid(S);
12
13 % PLOT DATA
14 cax=[-2 2];
15 for i=1:S.XML.parameters.Nb_Realizations.value;
16     subplot(4,3,i);
17     imagesc(S.x,S.y,S.D(:,:,1,i));axis image;caxis(cax);title(sprintf('SIM#=%d',i))
18 end
19
20 print('-dpng','sgems_sgsm_conditional')
```

#### 6.3.3.2 conditional data from file

A simple example of conditional sequential simulation (examples/sgems\_examples/sgems\_example\_sgsm\_conditional\_hard\_data\_from\_file.m):

```
1 % sgems_sgsm_conditional_hard_data_from_file :
2 %     Conditional SGSIM using hard data from file
3
4 % GET Default par file
5 S=sgems_get_par('sgsim');
6
7 % Define observed data=
8 d_obs=[18 13 0 2; 5 5 0 -1; 50 40 1 0];
9 % note that the third observation is located outside the 2D simulation plane!
10 sgems_write_pointset('obs.sgems',d_obs);
11 S.f_obs='obs.sgems';
```

```
12
13 S.XML.parameters.Max_Conditioning_Data.value=18;
14 S.XML.parameters.Nb_Realizations.value=100;
15 S.XML.parameters.Variogram.structure_1.ranges.max=30;
16 S.XML.parameters.Variogram.structure_1.ranges.medium=30;
17 S.XML.parameters.Variogram.structure_1.ranges.min=30;
18 S=sgems_grid(S);
19
20 %% PLOT DATA
21 cax=[-2 2];
22 for i=1:(min([10 S.XML.parameters.Nb_Realizations.value]));
23     subplot(4,3,i);
24     imagesc(S.x,S.y,S.D(:,:,1,i));axis image;caxis(cax);title(sprintf('SIM#=%d',i))
25 end
26 [m,v]=etype(S.D);
27 subplot(4,3,11);
28 imagesc(S.x,S.y,m');axis image;caxis(cax);title('Etype mean')
29 colorbar
30 hold on
31 plot(d_obs(:,1),d_obs(:,2),'ko','MarkerSize',10)
32 hold off
33 subplot(4,3,12);
34 imagesc(S.x,S.y,v');axis image;caxis([0 2]);title('Etype variance')
35 hold on
36 plot(d_obs(:,1),d_obs(:,2),'wo','MarkerSize',10)
37 hold off
38 colorbar
39
40 print('-dpng','sgems_sgsim_conditional_hard_data_from_file')
41
```



*Sequential Gaussian conditional simulation*

### 6.3.4 Unconditional SNESIM and FILTERSIM Gaussian simulation using S-GeMS

A simple example of unconditional SNESIM AND FILTERSIM simulation.

```

S1=sgems_get_par('snesim_std'); %
% Note that S1.ti_file is automatically set.
% simply change this to point to another training to use.
S1.XML.parameters.Nb_Realizations.value=4;

S2=sgems_get_par('filtersim_cont');
S2.XML.parameters.Nb_Realizations.value=4;

S1=sgems_grid(S1);
S2=sgems_grid(S2);

for i=1:S1.XML.parameters.Nb_Realizations.value;
    subplot(S1.XML.parameters.Nb_Realizations.value,2,i);
    imagesc(S1.x,S1.y,S1.D(:,:,1,i));axis image;

    subplot(S1.XML.parameters.Nb_Realizations.value,2,i+S1.XML.parameters.Nb_Realizations. ←
        value);

```



```

    imagesc(S2.x,S2.y,S2.D(:,:,1,i));axis image;
end

```

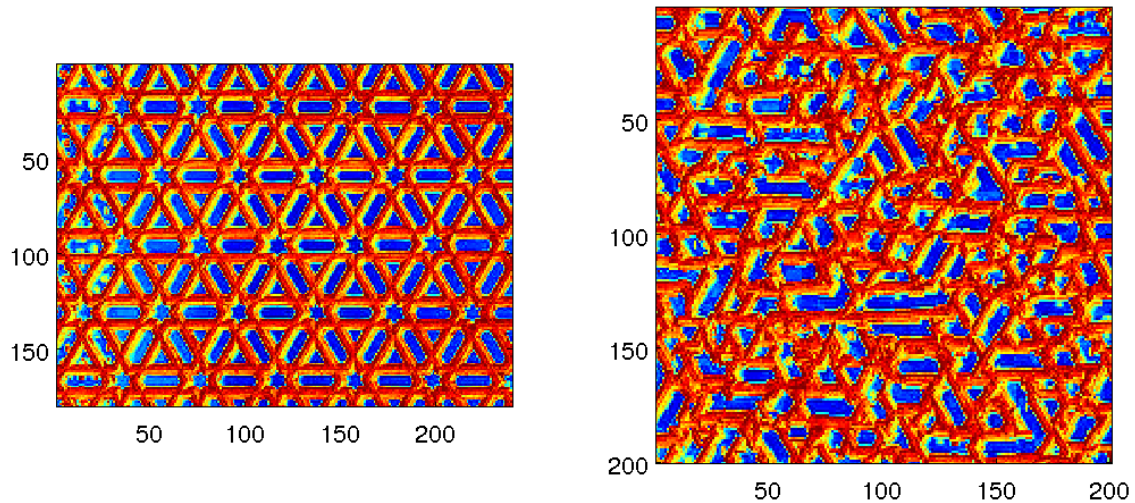
### 6.3.5 Convert image to training image;

Using a **JPG file from FLICKR** as training image:

```

1  % sgems_example_to_from_image : convert image and use as training image
2
3  % LOAD IMAGE AND CONVERT TO SGeMS binary TRAINING IMAGE
4  %file_img='1609350318_7300f07360_m_d.jpg'; % larger pattern
5  file_img='1609350318_7300f07360_m_d.jpg'; % smaller pattern
6  f_out=sgems_image2ti(file_img);
7  TI=sgems_read(f_out);
8
9
10 % SETUP FILTERSIM
11 S=sgems_get_par('filtersim_cont');
12 S.ti_file=f_out;
13 S.XML.parameters.PropertySelector_Training.grid=TI.grid_name;
14 S.XML.parameters.PropertySelector_Training.property=TI.property{1};
15 S.XML.parameters.Nb_Realizations.value=1;
16
17
18 S.dim.x=[1:1:200];
19 S.dim.y=[1:1:200];
20 S.dim.z=[0];
21
22 % RUN SIMULATION
23 S=sgems_grid(S);
24
25
26 % VISUALIZE RELIZATION
27 subplot(1,2,1);
28 imagesc(TI.x,TI.y,TI.D(:,:,1,1));axis image;
29 subplot(1,2,2);
30 imagesc(S.x,S.y,S.D(:,:,1,1));axis image;
31
32 print('-dpng','sgems_example_ti_from_image');

```



*Example of converting an image and using it for continuous filtersim simulation*

### 6.3.6 Simulation demonstration

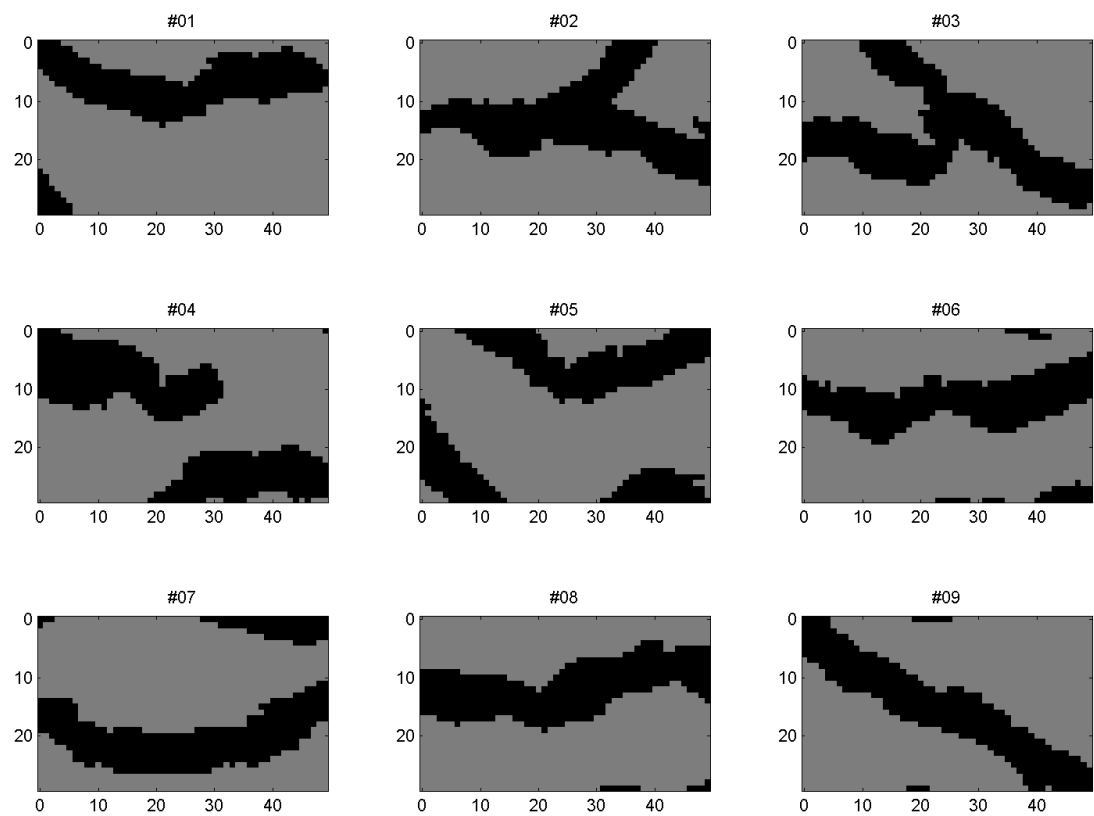
Demonstration simulation of mGstat supported simulation algorithms can be performed using Section 7.4.5. To see a list of supported simulation algorithms use:

```
1 >> sgems_get_par
2 sgems_get_par : available SGeMS type dssim
3 sgems_get_par : available SGeMS type filtersim_cate
4 sgems_get_par : available SGeMS type filtersim_cont
5 sgems_get_par : available SGeMS type lusim
6 sgems_get_par : available SGeMS type sgsim
7 sgems_get_par : available SGeMS type snesim_std
```

To run a demonstration of continuous filtersim simulation using the 'filtersim\_cont' algorithm do

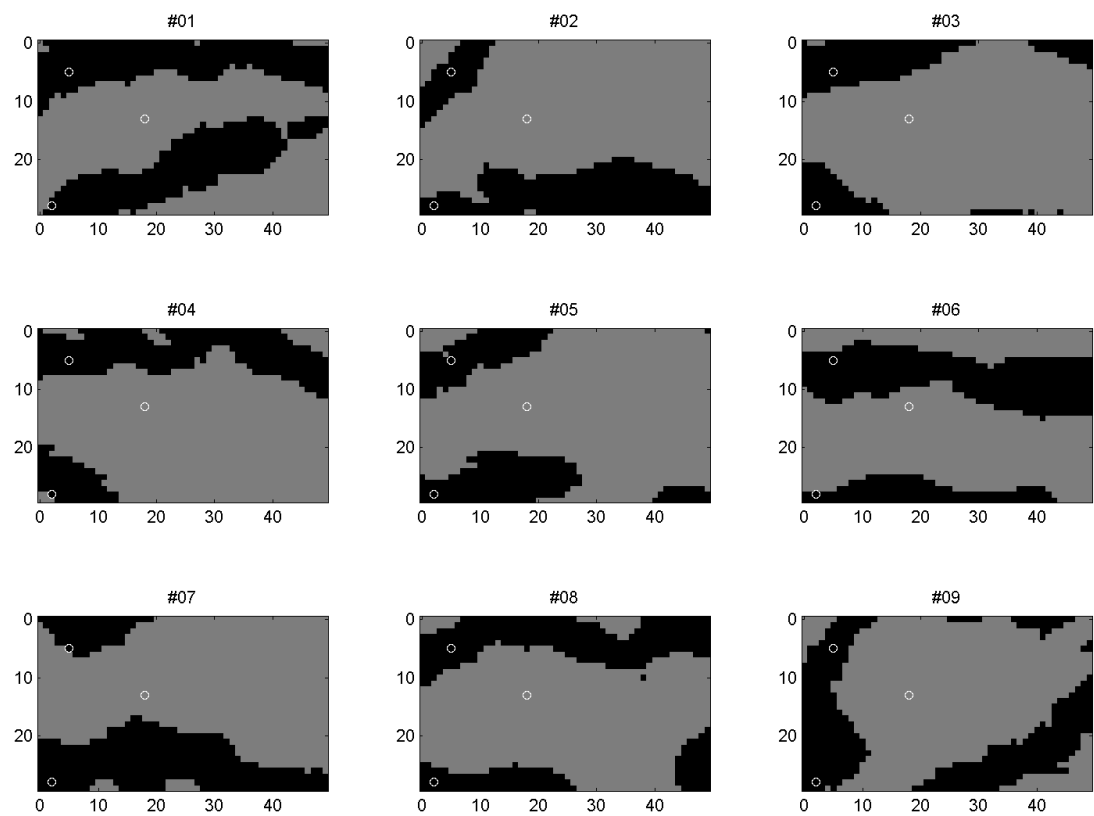
```
1 >> sgems_demo('filtersim_cont');
```

This will perform both unconditional and conditional simulation, and visualize the results as for example here below.



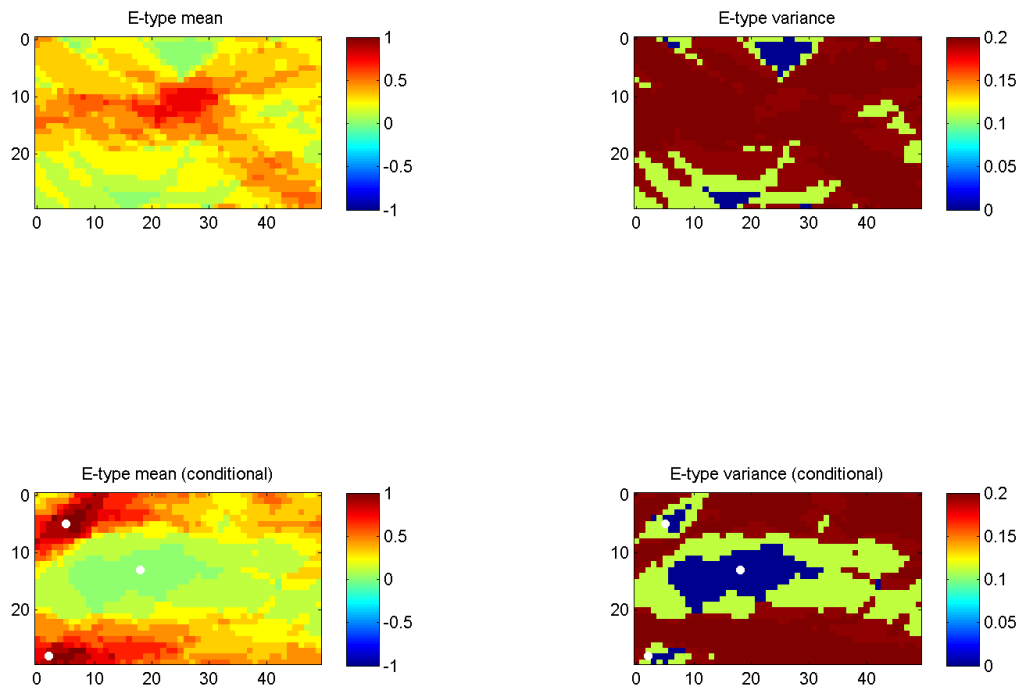
filtersim\_cont : unconditional simulation

*Unconditional simulation*



filtersim\_cont: conditional simulation

*Conditional simulation*



filtersim\_cont : etype

### *E-type on conditional simulations*

Running Section 7.4.5 without arguments will run the demonstration using all supported simulation algorithms.

## 6.3.7 Probability perturbation (PPM)

examples/sgems-examples/sgems\_example\_ppm.m is an example of applying the probability perturbation method, where one realization can be gradually deformed into another independent realization.

```

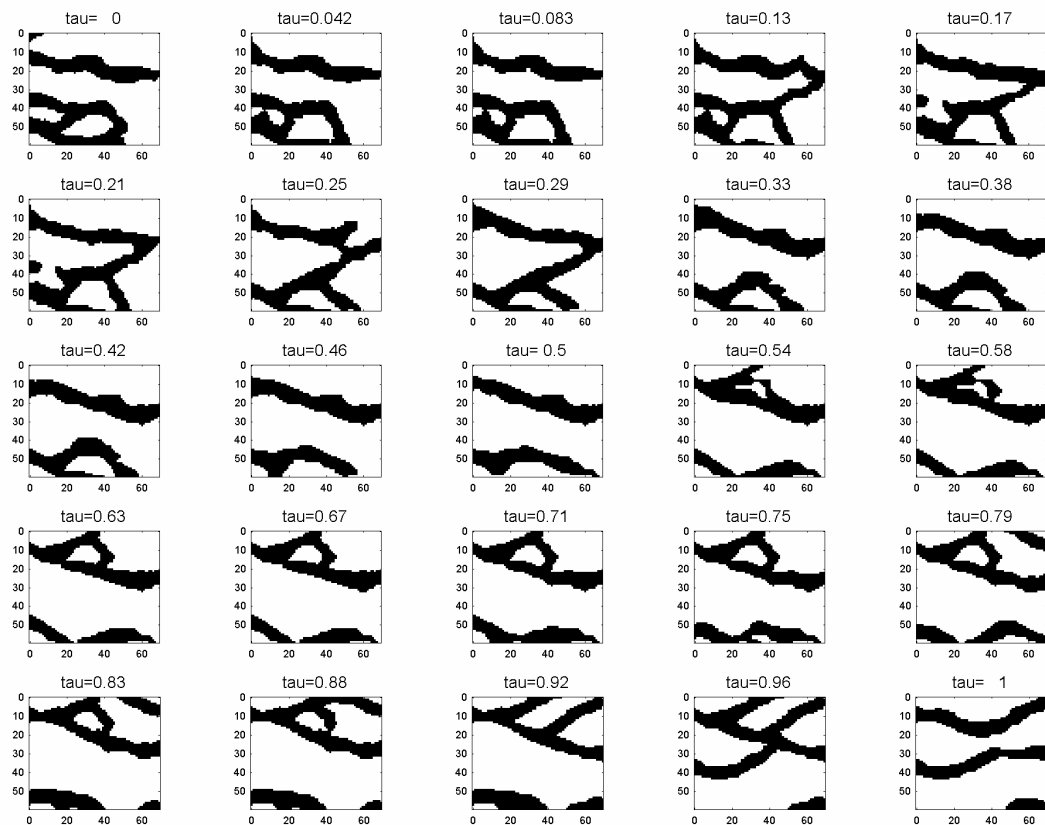
1  % sgems_example_ppm : example of using PPM
2  %
3
4  % get default snesim parameter file
5  S=sgems_get_par('snesim_std');
6  %S=sgems_get_par('filtersim_cate'); % SOFT PROB NOT YET IMPLEMENTED FOR
7  %FILTERSIM
8
9  % generate starting realization
10 S.XML.parameters.Nb_Realizations.value=1;
11 S=sgems_grid(S);
12
13
14 PPM_TARGET=sgems_read('snesim_std.sgems');
15
16 % loop over array of 'r' values
17 if ~exist('r_arr','var'),r_arr=linspace(0,1,25);end
18

```

```

19 nsp=ceil(sqrt(length(r_arr)));
20 for i=1:length(r_arr)
21     % perform PPM with rc=r_arr(i)
22     Sppm{i}=sgems_ppm(S,r_arr(i),PPM_TARGET);
23
24     figure_focus(1);
25     ax(i)=subplot(nsp,nsp,i);
26     imagesc(S.x,S.y,Sppm{i}.D');axis image;
27     colormap(1-gray)
28     set(gca,'FontSize',6)
29     title(sprintf('rc=%4.2g',r_arr(i)),'FontSize',10)
30     drawnow
31
32     figure_focus(2);
33     imagesc(S.x,S.y,Sppm{i}.D');axis image;
34     colormap(1-gray)
35     M(i)=getframe;
36
37 end
38
39 figure_focus(1);
40 colormap(1-gray)
41 print('-dpng','-r200','sgems_example_ppm.png');
42

```



Example of applying the Probability Perturbation Method using S-GeMS

## Chapter 7

# Matlab Reference

bla

### 7.1 Core functions

#### 7.1.1 Contents

```
mGstat Toolbox
Version 0.1 Feb 18, 2004

mGstat COMMANDS
mgstat_verbose - display verbose information
krig - simple/ordinary/tren kriging
precal_covar - precalculate covariance matrix
semivar_synth
semivar_exp
nscore : Normal socre transformation
inscore : Normal socre back transformation

sgsim      : Sequential Gaussian Simulation
dssim      : Direct sequential simulation
dssim-hr   : Direct sequential simulation with histogram reprod.
etype      : E-Type from reaslizations.

GSTAT SPECIFIC COMMANDS
gstat      - call gstat with parfile of mat-structure
gstat_convert - convert binary GSTAT output to ASCII
gstat_krig  - Point kriging
--gstat_cokrig - Point cokriging
--gstat_krig2d - 2D kriging
--gstat_cokrig2d- 2D cokriging
gstat_binary - returns the path to the binary gstat
gstat_demo   - mGstat demos
semivar_exp_gstat -

IO
read_petrel   - read petrel ascii formatted file
read_gstat_par - read gstat parameter file
write_gstat_par - write gstat parameter file
read_eas      - read EAS ascii formatted files
write_eas     - write EAS ascii formatted files
read_arcinfo_ascii - read ARCINFO ascii formatted files
```

```
write_arcinfo_ascii - write ARCINFO ascii formatted files
```

#### MISC

```
nanmean - mean of array, where NaN are excluded.
strip_space.m
format_variogram.m
deformat_variogram.m
vonk2d - random field generator
watermark - adds label to figure
progress_txt - ascii progress bar
```

#### Overloaded methods:

```
serial/Contents
mmreader/Contents
audiorecorder/Contents
audioplayer/Contents
VideoReader/Contents
instrument/Contents
rsmc/Contents
resultset/Contents
drivermanager/Contents
driver/Contents
dmd/Contents
dbtbx/Contents
database/Contents
cursor/Contents
```

### 7.1.2 CreateMisfitFunction

CreateMisfitFunction : Dynamically created function to be used for semivariogram optimization

### 7.1.3 MakeXmlRef

### 7.1.4 arcinfo2eas

arcinfo2eas : convert ArcInfo file to EAS

#### CALL

```
arcinfo2eas(file_arcinfo,file_eas);
```

#### or

```
arcinfo2eas(file_arcinfo);
using the same file name as the arcinfo file
but with the 'eas' file extension.
```

### 7.1.5 block\_log



```
block_log : block a log
```

Call:

```
[mz,z_out]=block_log(val,z,bz);
```

### 7.1.6 cokrig\_sk

```
cokrig_sk : Simple CoKriging
```

Call :

```
function [d_est,d_var,lambda_sk,K_sk,k_sk]=cokrig_sk(pos_known,val_known,pos_est,V,val_0 ←  
);
```

TMH/2005

### 7.1.7 colorbar\_shift

```
colorbar_shift : Adds a colorbar to the current figure, with no reshaping
```

Before printing to a PS file you may need to set :

```
set(gca,'ActivePositionProperty','Position')
```

example :

```
subplot(2,2,1)  
imagesc(peaks)  
subplot(2,2,2)  
imagesc(peaks)  
set(gca,'ActivePositionProperty','Position')  
colorbar_shift;
```

### 7.1.8 colormap\_nan

```
colormap_nan
```

Replaces all NaN values with a specific color, and rescales the colorbar appropriately;

example :

```
d=peaks(200);  
d(find(d<0))=NaN;  
figure(1);imagesc(d);  
colormap(hot);  
colormap_nan;  
drawnow;  
pause(2)  
  
figure(2);imagesc(d);  
colormap_nan(jet,[.2 .9 .1]);  
colorbar;drawnow;
```

```

pause(2);

figure(3);imagesc(d);
colormap_nan(jet(1000),[.2 .9 .1]);
colorbar

```

### 7.1.9 colormap\_squeeze

```

colormap_squeeze

Call :
  colormap_squeeze(dperc);
  dperc=[0 .. 0.5];

  imagesc(peaks);
  colormap_squeeze(.1);
  pause(1);
  colormap_squeeze(.1);

```

### 7.1.10 comb\_cprob

comb\_cprob : PDF combination using permancne of ratios

```

Call :
  pAgBC=comb_cprob(pA,pAgB,pAgC)

```

```

pA      : Prob(A)
pAgB    : Prob(A|B)
pAgC    : Prob(A|C)
pAgBC   : Prob(A|B,C)

```

Combination of conditional probabilities  
based on permanence of updating ratios.

Journal, An Alternative to Traditional Data Independence  
Hypotheses, Math Geol(34), 2002

### 7.1.11 comb\_cprob\_ind

comb\_cprob\_ind : Combination of two independent conditional PDF

```

Call :
  pAgBC=comb_cprob_ind(pA,pAgB,pAgC)

```

```

pA      : Prob(A)
pAgB    : Prob(A|B)
pAgC    : Prob(A|C)
pAgBC   : Prob(A|B,C)

```

TMH/2005

### 7.1.12 comb\_cprob\_nd

comb\_cprob\_nd : PDF combination using permanence of ratios

Combination of 'nd' conditional probabilities  
based on permanence of updating ratios.

Call :

```
pAgND=comb_cprob(pA,pAgND,wAgND)
```

```
pA      [scalar] : Prob(A)
pAgND   [array]  : Prob(A|N1),Prob(A|N2),...,Prob(A|ND)
wAgBC   [array]  : Weight of each cprob
pAgBC   [scala]  : Prob(A|ND)
```

Combination of conditional probabilities  
based on permanence of updating ratios.

Journel, An Alternative to Traditional Data Independence  
Hypotheses, Math Geol(34), 2002

### 7.1.13 covar\_exp

semivar\_exp : Calcualte experimental variogram

```
[hc,garr,h,gamma,hangc,head,tail]=semivar_exp(pos,val,nbin,nbinang)
```

```
pos : [ndata,ndims]
val : [ndata,ndata_types]
```

```
nbin : [integer] number of bins on distance axes
       [array] if specified as an array, this is used.
```

```
nbinang : [integer] number of arrays between 0/180 degrees
           (default 1)
```

Example isotrop:

```
[hc,garr]=semivar_exp(pos,val);
plot(garr,hc);
```

Example directional [0,45,90,135,180]:

```
[hc,garr,h,gamma,hangc]=semivar_exp(pos,val,20,4);
plot(garr,hc);
legend(num2str(hangc'))
```

TMH/2005

### 7.1.14 cpdf

cpdf : cumulative probability density function

```
[pk_obs]=cpdf(alldata,d_obs,doPlot)
```

finds pk quantiles for data (d\_obs),  
based on a series of data (alldata)

### 7.1.15 create\_nscore\_lookup

### 7.1.16 create\_nscore\_lookup\_old

### 7.1.17 csemivar\_exp

csemivar\_exp : Calculate experimental cross semivariogram

```
[hc,garr,h,gamma,hangc,head,tail]=semivar_exp(pos1,val1,pos2,val2,nbin,nbinang)
```

```
pos1 : [ndata,ndims] : attribute 1
val1 : [ndata,1]      : attribute 1
pos2 : [ndata,ndims] : attribute 2
val2 : [ndata,1]      : attribute 2
```

```
nbin : [integer] number of bins on distance axes
       [array] if specified as an array, this is used.
```

```
nbinang : [integer] number of arrays between 0/180 degrees
           (default 1)
```

Example isotrop:

```
[hc,garr]=semivar_exp(pos1,val1,pos2,val2);
plot(garr,hc);
```

Example directional [0,45,90,135,180]:

```
[hc,garr,h,gamma,hangc]=semivar_exp(pos1,val1,pos2,val2,20,4);
plot(garr,hc);
legend(num2str(hangc'))
```

TMH/2005

### 7.1.18 deformat\_variogram

deformat\_variogram : convert gstat variogram line into matlab structure

Call:

```
V=deformat_variogram(txt);
```

```
Example:
V=deformat_variogram('1 Sph(2)')
```

V =

```
    par1: 1
    par2: 2
    type: 'Sph'
    itype: 1
```

See also : format\_variogram

TMH /2004

### 7.1.19 dual

### 7.1.20 edist

edist : Euclidean distance

Call :

```
D=edist(p1,p2,transform,isorange)
```

p1,p2 : vectors

transform : GSTAT anisotropy and/or range information

isorange : [0] (default), transform is the usual GSTAT-anisotropy setting  
 isorange : [1] means that transform simply lists the range in  
 each dimensions, and that no rotation is performed

### 7.1.21 estim\_taran

estim\_taran, one data set : Tarantola equations (16-17)

```
CALL : [m_est,Cm_est]=estim_taran(G,Cm,Cd,m0,d0);
```

### 7.1.22 etype

### 7.1.23 f77strip

f77strip : Strips f77 characters from binary file.

```
[data]=f77strip(file,format,xskip,zskip);
```

IN :

required

file [string], optional, default='f77.bin'

```

optional :
format [string], 'float16','float32'[default],'float64'
xskip [scalar], Skip every xskip column
zskip [scalar], Skip every zskip row

```

```

OUT
data [matrix], required

```

Purpose : Reads a f77 style binary file

At the beginning and end of each row, an integer containing the number of bytes in the row is printed (like ftnunstrip/ftnstrip in the CWP SU package)

by Thomas Mejer Hansen, 05/2000  
Octave 2.0.15 and Matlab 5.3 compliant

### 7.1.24 fft\_ma\_3d

```

The FFT-MA algorithm in 3D
Call: out=FFT_MA_3D(ny,nx,nz,Nly,Nlx,Nlz,cell,h_min,h_max,h_z,gmean,gvar,it)
it: 1) Spherical, 2) Exponential, 3) Gaussian
ny, nx, nz: Number of model parameters in the x, y, and z directions,
respectively
Nlx, Nly, Nlz: Extensions of the grid in the x, y, and z directions in
order to avoid artefacts due to edge effects. Number of model parameters
in the extension = Nlx*range_x/cell, where cell is the size of the cells
and range_x is the range i the x-direction.
h_min, h_max, and h_z, are the ranges in the horizontal (x), vertical (y) and (z) ↔
direction.
gvar: Global variance
gmean: Global mean

```

### 7.1.25 format\_variogram

format\_variogram : Convert matlab style Variogram to Gstat style

```

Call :
txt=format_variogram(V);

```

See also: deformat\_variogram

### 7.1.26 fresnel\_punch

fresnel\_punch : computes the sensitivity kernel for a wave traveling from S to R.

```

CALL :
[K,RAY,timeS,timeR,raypath]=fresnel_punch(Vel,x,y,z,S,R,freq,alpha);

```

```

IN :
Vel : Velocity field
x [1:nx] :
y [1:ny] :

```

```

z [1:nz] :
S [1,3] : Location of Source
R [1,3] : Location of Receiver
freq : frequency
alpha: controls exponential decay away ray path

OUT :
K : Sensitivity kernel
R : Ray sensitivity kernel (High Frequency approx)
timeS : travel computed form Source
timeR : travel computed form Receiver
raypath [nraydata,ndim] : the center of the raypath

TMH/2006

```

### 7.1.27 gaussian\_simulation\_cholesky

gaussian\_simulation\_cholesky : generate realizations from a Gaussian 2D distribution mean m0 and covariance Cm

Very efficient for smaller models to generate a sample of the posterior PDF for least squares inversion problems :

For example :

```

[m_est,Cm_est]=least_squares_inversion(G,Cm,Cd,m0,d0);
z_uncond=gaussian_simulation_cholesky(m_est,Cm_est,nsim);
z_cond=gaussian_simulation_cholesky(m_est,Cm_est,nsim);

```

Choleksy decomposition can be calculated prior to calling

```

Cm=chol(Cm);
is_chol=1;
z_cond=gaussian_simulation_cholesky(m_est,Cm_est,nsim,is_chol);

```

% unconditional realization:

```

x=[1:1:40];
y=[1:1:40];
[xx,yy]=meshgrid(x,y);
Cm=precal_cov([xx(:) yy(:)],[xx(:) yy(:)],'1 Sph(45,.1,30)');
m0=xx.*0;
nsim=12;
[z_uncond,D]=gaussian_simulation_cholesky(m0,Cm,nsim);
for i=1:nsim;subplot(4,3,i);imagesc(x,y,D(:, :, i));axis image;end

```

see also gaussian\_simulation\_cholesky\_resim

### 7.1.28 gstat

gstat : call gstat from Matlab

CALL : gstat(G)

G : gstat data structure OR gstat parameter file on disk

```
[pred,pred_var,pred_covar,mask,G]=gstat(G)
```

### 7.1.29 gstat\_binary

`gstat_binary` : returns the path to the binary `gstat`

Call :  
`gstat_bin = gstat_binary;`

### 7.1.30 gstat\_convert

`gstat_convert` : convert between ascii/binary formats

CALL : `[data,x,y,dx,nanval]=gstat_convert(file,f,suf)`

### 7.1.31 gstat\_krig

`gstat_krig` : Simple/Ordinary Kriging using GSTAT

Call :  
`[d_est,d_var]=gstat_krig(pos_known,val_known,pos_est,V,options);`

`ndata` : number of data observations  
`ndims` : dimensions of data location ( $\geq 1$ )  
`nest` : number of data locations to be estimated

`pos_known [ndata,ndims]` : Locations of data observations  
`val_known [ndata,1 or 2]` : `col1` : Data value as measured at '`pos_known`'  
                               `col2` : Data uncertainty as measured at  
                                       '`pos_known`' (optional)  
`pos_est [nest ,ndims]` : Location of data to be estimated  
`V` : Variogram model, e.g. '`1 Sph(100)`'

```
%% Example 1 : 1D - NO DATA UNCERTAINTY
profile on
pos_known=10*rand(10,1);
val_known=rand(size(pos_known)); % adding some uncertainty
pos_est=[0:.01:10]';
V=deformat_variogram('1 Sph(1)');
[d_est,d_var]=gstat_krig(pos_known,val_known,pos_est,V);
plot(pos_est,d_est,'r.',pos_est,d_var,'b.',pos_known,val_known(:,1),'g*')
legend('SK estimate','SK variance','Observed Data')
title(['V = ',V])
profile viewer
```

```
%% Example 2 : 1D - Data Uncertainty
pos_known=[1;5;10];
val_known=[0 3 2;0.001 1 0.001]'; % adding some uncertainty
pos_est=[0:.01:10]';
V='1 Sph(2)';
[d_est,d_var]=gstat_krig(pos_known,val_known,pos_est,V);
```



```

plot(pos_est,d_est,'r.',pos_est,d_var,'b.',pos_known,val_known(:,1),'g*')
legend('SK estimate','SK variance','Observed Data')
title(['using data uncertainty, V = ',V])

%% Example 3 : 2D estimation
pos_known=[0 1;5 8;10 1];
val_known=[0 3 2]';
x=[0:.1:10];
y=[0:.1:10];
[xx,yy]=meshgrid(x,y);
pos_est=[xx(:) yy(:)];
V='1 Sph(7)';
[d_est,d_var]=gstat_krig(pos_known,val_known,pos_est,V);
subplot(1,2,1);scatter(pos_est(:,1),pos_est(:,2),10,d_est)
axis image;title('Kriging mean')
subplot(1,2,2);scatter(pos_est(:,1),pos_est(:,2),10,d_var)
axis image;title('Kriging variance')

%% Example 4 :SIMULATION
pos_known=[0 1;5 1;10 1];
val_known=[0 3 2]';
pos_est=linspace(-1,11,200)';pos_est(:,2)=1;
V='.0001 Nug(0) + .2 Gau(2)';
[d_est,d_var]=gstat_krig(pos_known,val_known,pos_est,V);
plot(pos_est(:,1),d_est,'k-',pos_known(:,1),val_known(:,1),'r*')

options.nsim=120;
[d_sim,d_varsim,pos_sim]=gstat_krig(pos_known,val_known,pos_est,V,options);
d=sortrows([pos_sim(:,1) d_sim],1);
d_sim=d(:,2:(options.nsim+1));

d=sortrows([pos_sim(:,1) d_varsim],1);
d_varsim=d(:,2);

plot(pos_est(:,1),d_sim,'r-');

hold on
plot(pos_est(:,1),d_est,'k-','linewidth',4)
plot(pos_known(:,1),val_known(:,1),'b.')

plot(pos_est(:,1),d_varsim-4,'k-')
plot(pos_est(:,1),d_var-4,'r-')
hold off

```

### 7.1.32 gstat\_krig\_blinderror

`gstat_krig_blinderror` : blind cross validation using `gstat`

Call as `gstat_krig` is called :

```
[d_est,d_var,be,d_diff]=gstat_krig_blinderror(pos_known,val_known,pos_est,V,options);
```

[d\_est,d\_var] : Cross validation prediction

[be] : Cross validation error

/TMH 12/2005

### 7.1.33 hpd\_2d

```
hpd_2d : highest posterior density

call :
  [levels]=hpd_2d(lik,hpd_level)cl

lik=abs(peaks);
levels=hpd_2d(lik,[.1:.2:.9])
contourf(lik,levels)
```

### 7.1.34 hpd\_2d\_point

```
hpd_2d_point : highest posterior density plot from scattered data

Call :
  [lik,levels,x,y]=hpd_2d_point(x_p,y_p,lik_p,x,y,hpd_levels,corner_type)

See also : hpd_2d

Example :
nd=1300;
x_p=randn(nd,1)*1;
y_p=randn(nd,1)*1;
lik_p = abs(peaks(x_p,y_p));
subplot(1,3,1);
[lik,levels,x,y]=hpd_2d_point(x_p,y_p,lik_p);
subplot(1,3,2);
[lik,levels,x,y]=hpd_2d_point(x_p,y_p,lik_p,[],[],[.1:.1:1]);
subplot(1,3,3);
[lik,levels,x,y]=hpd_2d_point(x_p,y_p,lik_p,-1:.1:1,-1:.1:1,[.2 0.5 1.0]);
```

### 7.1.35 icpdf

```
icdf : inverse cumulative density function

find data value associated to an pk quantile.

CALL : d_obs=icdf(data,pk_obs)
```

### 7.1.36 indicator\_transform\_con

```
indicator_transform_con : transform continous data into indicator

CALL :

[id,lev]=indicator_transform_con(d,lev)

[d] : data
[lev] : indicator transform of list lev#s : Prob(zi<lev(i))
```

```
if not specified level is chosen to match qantiles
.1,.2,...,.9
```

### 7.1.37 indicator\_transform\_dis

```
indicator_transform_dis : transform discrete data into indicator

CALL :

[id,lev]=indicator_transform_dis(d,ident)

[d] : data
[ident] : Transform data into a binary discrete data, such that Prob(zi=ident(i))
         if not specified level is chosen all unique4 discrete
         identifiers are chosen.
```

### 7.1.38 inscore

```
inscore : normal score BACK transform

CALL :
  d=inscore(d_nscore,o_nscore)

d_nscore : normal score values to be back transformed
          using the 'o_nscore' object, obtained using
          'nscore'

See also nscore.m
```

### 7.1.39 isorange

```
isorange : convert range scaling to gstat/gslib range settings

for example
  V = '1.0 Sph(0.7,0.8,0.9)';

  Vgstat=isorange(V);
  format_variogram(Vgstat,1)

Used when 'options.isorange=1'
```

### 7.1.40 krig

```
krig : Simple/Ordinar/Trend Kriging

Call :
[d_est,d_var,lambdak,K,k,inhood]=krig(pos_known,val_known,pos_est,V,options);

ndata : number of data observations
```

```

ndims : dimensions of data location (>=1)
nest  : number of data locations to be estimated

pos_known [ndata,ndims] : Locations of data observations
val_known [ndata,1 or 2] : col1 : Data value as measured at 'pos_known'
                           col2 : Data uncertainty as measured at
                           'pos_known' (optional)
pos_est   [N ,ndims] : Location of N data locations to be estimated
V : Variogram model, e.g. '1 Sph(100)'
val_0 : A priori assumed data value (default=mean(val_known))

```

Example 1D - NO DATA UNCERTAINTY

```

profile on
pos_known=10*rand(10,1);
val_known=rand(size(pos_known)); % adding some uncertainty
pos_est=[0:.01:10]';
V=deformat_variogram('1 Sph(1)');
[d_est,d_var]=krig(pos_known,val_known,pos_est,V);
plot(pos_est,d_est,'r.',pos_est,d_var,'b.',pos_known,val_known(:,1),'g*')
legend('SK estimate','SK variance','Observed Data')
%title(['V = ',V])
profile viewer

```

See source code for more examples

see also : krig\_npoint, krig\_blinderror

### 7.1.41 krig\_blinderror

```

krig_blinderror : Cross validation blind error
CALL :
    [d_est,d_var,be,d_diff,L,L2]=krig_blinderror(pos_known,val_known,pos_est,V,options, ←
        nleaveout)

```

### 7.1.42 krig\_covar\_lik

krig\_covar\_lik : Calculates the likelihood tha V is consistent with data observations

```

Call :
    L=krig_covar_lik(pos_known,val_known,V,options)

```

Can be used to infer covariance properties (range, sill, anisotropy,...)

### 7.1.43 krig\_crossval\_1d\_exh

```

krig_crossval_1d_exh
CALL :
    [V_L,B_be,ML,Mbe,ML2,par2_range,nugfrac_range]=krig_crossval_1d_exh(pos_known,val_known ←
        ,V,options);

```

---

```
function [V_L,V_be,ML,Mbe,ML2,par2_range,nugfrac_range]=krig_crossval_1d_exh(pos_known, ↵
    val_known,V,options);
```

#### 7.1.44 krig\_npoint

```
krig_npoint : as 'krig' but for multiple estimation position.

[d_est,d_var,options]=krig_npoint(pos_known,val_known,pos_est,V,options);

As krig, but allowing size(pos_known,1)>1

See also : krig
```

#### 7.1.45 krig\_optim\_1d\_exh

```
krig_optim_1d_exh
CALL :
    [V_L,B_be,ML,Mbe,ML2,par2_range,nugfrac_range]=krig_optim_1d_exh(pos_known,val_known,V, ↵
        options);
```

#### 7.1.46 krig\_optim\_mcmc

```
krig_optim_mcmc
CALL :
    [V_new,be_acc,L_acc,par2,nugfrac_acc,V_acc,options]=krig_optim_mcmc(pos_known,val_known ↵
        ,V,options)
```

#### 7.1.47 krig\_optim\_ml

```
krig_optim_ml : MCMC Maximum likelihood optimization

Call :

    [Vop2,Vop1,be,L,par2,nugfrac,Vall]=krig_optim_ml(pos_known,val_known,V,options)
```

#### 7.1.48 krig\_optim\_range

```
krig_optim_range
CALL :
    [V,be]=krig_optim_range(pos_known,val_known,V,options)
```

---

### 7.1.49 krig\_volume

krig\_volume : kriging with volume average data

See hansen et. al. 2005.

### 7.1.50 least\_squares\_inversion

least\_squares\_inversion, one data set : Tarantola equations (16-17)

CALL : [m\_est,Cm\_est]=least\_squares\_inversion(G,Cm,Cd,m0,d0);

See also : gaussian\_simulation\_cholesky

### 7.1.51 least\_squares\_oneone

least\_squares\_oneone : Least squares inversion using only scalar operations. (only valid when offdiag(Cd)=0)

See Tarantola (2005), Inverse Problem Theory, page 198, Ch 6.

Call :

[m\_est,sigma2,Cm\_est]=least\_squares\_oneone(G,Cm,Cd,m0,d\_obs)

### 7.1.52 least\_squares\_partition\_data

least\_squares\_partition\_data : least sq. inversion using partitioning

See Tarantola (2005), page 197, eqn. 6.211 or 6.212.

Least squares inversion by partitioning into to data subsets with independent data covariance !

This can be very fast if the number of data observations is large, and large compared to the number of model parameters.

CALL : [m\_est,Cm\_est]=least\_squares\_partition\_data(G,Cm,Cd,m0,d\_obs,nsubsets,use\_eq);

### 7.1.53 least\_squares\_slice

least\_squares\_slice : least sq. inversion using partitioning

CALL : [m\_est,Cm\_est]=least\_squares\_slice(G,Cm,Cd,m0,d0,id,im);

### 7.1.54 mgstat\_clean

mgstat\_clean : clean up temporary files from visim, sgems, fast

### 7.1.55 mgstat\_demo

mgstat\_demo : demos illustrating the use of mGstat

Try some of the following demos:

```
mgstat_demo('mgstat'); % mgstat demo, illustrating the native matlab
                        % algorithms
mgstat_demo('gstat');  % demos using gstat
mgstat_demo('visim');  % demos using visim
mgstat_demo('sgems');  % demos using sgems
mgstat_demo('snesim'); % demos using snesim

mgstat_demo('all'); % runs all of the available demos.
```

### 7.1.56 mgstat\_dir

mgstat\_dir : return the install directory for mGstat

### 7.1.57 mgstat\_set\_path

mgstat\_set\_path : set path to all mGstat

```
set path to :
  mGstat_Install_Dir/snesim
  mGstat_Install_Dir/visim
  mGstat_Install_Dir/sgems
  mGstat_Install_Dir/fast
  mGstat_Install_Dir/misc
```

### 7.1.58 mgstat\_verbose

mgstat\_verbose : list verbose information to the console

Call:

```
mgstat_verbose(txt,verbose)
```

txt [string] : text to be displayed

verbose [integer] (def=0) : increase to see more information

'vlevel' must be set in the mgstat\_verbose.m m-file.

All entries with vebose>vlevel are displayed

### 7.1.59 nanmean

nanmean : mean of data, ignoring NaN's

call : meandata=nanmean(data)

```
data [n-dimensional array]
meandata [scalar]
```

TMH(tmh@gfy.ku.dk), 2001

### 7.1.60 nanstd

nanstd : std of data, ignoring NaN's

call : stddata=nanstd(data)

```
data [n-dimensional array]
stddata [scalar]
```

TMH(tmh@gfy.ku.dk), 2001

### 7.1.61 nanvar

nanvar : var of data, ignoring NaN's

call : vardata=nanvar(data)

```
data [n-dimensional array]
vardata [scalar]
```

TMH(tmh@gfy.ku.dk), 2001

### 7.1.62 nhood

nhood : Neighborhood selection

TMH/2005

### 7.1.63 normcdf

NORMCDF returns normal cumulative distribution function

```
cdf = normcdf(x,m,s);
```

Computes the CDF of a the normal distribution

with mean m and standard deviation s

default: m=0; s=1;

x,m,s must be matrices of same size, or any one can be a scalar.

see also: NORMPDF, NORMINV



### 7.1.64 norminv

NORMINV returns inverse cumulative function of the normal distribution

```
x = norminv(p,m,s);
```

Computes the quantile (inverse of the CDF) of a the normal cumulative distribution with mean m and standard deviation s  
default: m=0; s=1;

p,m,s must be matrices of same size, or any one can be a scalar.

see also: NORMPDF, NORMCDF

### 7.1.65 normpdf

NORMPDF returns normal probability density

```
pdf = normpdf(x,m,s);
```

Computes the PDF of a the normal distribution  
with mean m and standard deviation s  
default: m=0; s=1;

x,m,s must be matrices of same size, or any one can be a scalar.

see also: NORMCDF, NORMINV

### 7.1.66 nscore

nscore : Normal score transform

CALL :

```
[d_nscore,o_nscore]=nscore(d,w1,w2,dmin,dmax,DoPlot)
```

INPUT PARAMETERS :

Required :

d : array of data to transformed into normal scorres.

Optional :

w1,dmin : Extrapolation options for lower tail.

w1=1 -> linear interpolation

w1>1 -> Gradual power interpolation

w2,dmax : Extrapolation options for lower tail.

w1=1 -> linear interpolation

w1<1 -> Gradual power interpolation

See Goovaerts page 280-281 for details

DoPlot : ==1 --> The choice of CCPDF to be used for normal score transformation is plotted

OUTPUT PARAMETERS

d\_nscore : normal score transform of input data

o\_nscore : normal socre object containing information  
needed to perform normal score backtransform.

See also : inscore

### 7.1.67 plot\_scale

plot\_scale : plot scale to figure

Call:

```
plot_scale(ax,len,pos,FontSize)
```

ax: axis (gca)

len [1,2]: length of scale length in each direction

pos [1],[2],[3] or [4]: Position of scale plot.  
[NW],[SW],[NE],[SE]

Example

```
imagesc(peaks);
hold on;
plot_scale(gca,[15 15],3);
hold off;
axis image
```

### 7.1.68 pos2index

```
[ix,iy]=pos2index(xpos,ypos,x,y);
```

### 7.1.69 ppp

file=ppp.m : Creates a lx,ly cm plot

call function ppp(lx,ly,Fsize,x1,y1),

(lx,ly) : WIDTH and HEIGHT of plot in cm

Fsize : Font Size

(x1,y1) : Lower left corner of plot (relative to lower left corner of paper)

(C) Thomas Mejer Hansen, 1997-2001, tmh@gfy.ku.dk

### 7.1.70 precal\_cov

precal\_cov : Precalculate covariance matrix

CALL :

```
cov=precal_cov(pos1,pos2,V,options);
```

pos1 [ndata1,ndims] : Location of data to be estimated

pos2 [ndata2,ndims] : Location of data to be estimated

V [struct] : Variogram structure

cov [ndata1,ndata1] : Covariance matrix

```

Ex:
x=[1:1:10];
y=[1:1:20];
[xx,yy]=meshgrid(x,y);
cov=precal_cov([xx(:) yy(:)],[xx(:) yy(:)],'1 Sph(5,.1,0)');

```

### 7.1.71 print\_mul

print\_mul : prints both EPS and PNG figures of current plot

Call :  
 print\_mul('test') : prints test.eps and test.png

In case 'mogrify' is available on the system  
 the png file will be trimmed and optionall  
 A specific color will be made transparent :

```

print_mul('test',red)
    also creates trim_test.png

print_mul('test',1)
    also creates trim_test.png, with transparent white color

print_mul('test','red')
    also creates trim_test.png, with transparent red color

```

/TMH 12/2005

### 7.1.72 progress\_txt

progress\_txt : console based progress bar

```

Ex1 :
for i=1:10000;
    progress_txt(i,10000,'Ciao');
end

```

```

Ex1 :

for i=1:10;
for j=1:10;
for k=1:10;
    progress_txt([i j k],[10 100 1000],'i','j','k');
end
end
end

```

TMH/2005, thomas@cultpenguin.com

### 7.1.73 rank\_transform

```
rank_transform : rank transform data
```

```
Call :  
[r]=rank_transform(d);
```

### 7.1.74 read\_arcinfo\_ascii

```
read_arcinfo_ascii : Reads ascii formatted ArcInfo files
```

```
Call :  
[data,x,y,dx,nanval,x0,y0,xll,yll]=read_arcinfo_ascii(filename);
```

### 7.1.75 read\_bin

```
read_bin : Reads a binary file to matlab
```

```
CALL : function [dataout]=read_bin(fileid,nx,nz,fchar,format,b_order)
```

```
REQUIRED
```

```
fileid  
nx
```

```
OPTIONAL
```

```
nz      : Number of samples in 2nd direction  
fchar (scalar) : (==1)Remove F77 chracters  
format (string) : 'float32' [default] or 'int16' or 'int32',...  
b_order : set byteorder : '0' : Little Endian  
          '1' : Big endian
```

```
/TMH 2006
```

### 7.1.76 read\_eas

```
read_eas : reads an GEO EAS formatted file into Matlab.
```

```
Call [data,header,title]=read_eas(filename);
```

```
TMH (tmh@gfy.ku.dk)
```

### 7.1.77 read\_emm

```
read_emm : read emm output file from emldiv
```

```
CALL :  
[emm]=read_emm(filename);
```

### 7.1.78 read\_gstat\_ascii

### 7.1.79 read\_gstat\_par

```
read_gstat_par : Reads gstat.par file into Matlab data structure

CALL :
    G = read_gstat_par('ex01.cmd');

KNOWN BUGS (FEB 2004)
    Cannot load covariogram : covariogram(data1,data2)
    Semivariogram line : Can only contain verilogram, not filename
```

### 7.1.80 read\_gstat\_semivar

### 7.1.81 read\_petrel

```
read_peterl : reads an PETREL ascii point file

Call [data,header]=read_eas(filename);

TMH (tmh@gfy.ku.dk)
```

### 7.1.82 read\_punch\_par

```
FUNCTION : [fxs,fys,fzs,nx,ny,nz,x0,y0,z0,h,timefile,velfile,reverse,maxoff]= ←
    read_punch_par(filename);
Purpose : Reads PAR file from PUNCH program (HOLE PROGRAM)

TMH 09/1999;
```

### 7.1.83 read\_surfer\_grid

```
read_surfer_grid : Read Surfer ASCII GRD file

CALL :
    [data,x,y]=read_surfer_grid(filename);

IN:
    filename [char] :string
OUT:
    data [ny,nx]
    x [nx]
    y [ny]
```

### 7.1.84 rot2

```

rot2 : 2D coordiante transformation

Call :
  htrans=rot2(h,ang,ani,dir)

h :[hx,hy] location
ang : angle in radians
ani : anisotropy factor

dir : 'direction' =1, normal transform, <>1, inverse transform

TMH/2005

```

### 7.1.85 scatter\_dot

```

scatter_dot : A black dot beneith scatter dots

Call :
  scatter_dot(x,y,MS,v,option)

```

### 7.1.86 scatter\_hpd

```

scatter_hpd : calculate 2D HPD region

[prob,levels,x_arr,y_arr,f2,c2]=scatter_hpd(x,y,p,p_levels,x_arr,y_arr)

```

### 7.1.87 semivar

```

semivar : calcualte semivariogram

[binc,sv,bin_array,svM]=semivar(loc,val,bin_array);

loc : [ndim,n]

```

### 7.1.88 semivar\_exp

```

semivar_exp : Calculate experimental variogram

[gamma,h,ang_center,gamma_cloud,h_cloud]=semivar_exp(pos,val,nbin,nbinang)

pos : [ndata,ndims]
val : [ndata,ndata_types]

nbin : [integer] number of bins on distance anxes
       [array] if specified as an array, this is used.

```

```

nbinang : [integer] number of arrays between 0/180 degrees (def=1)
          [array] array of angles.

Example : load jura data
dwd=[mgstat_dir,filesep,'examples',filesep,'data',filesep,'jura',filesep];
[p,pHeader]=read_eas([dwd,'prediction.dat']);
idata=6;dval=pHeader{idata};
pos=[p(:,1) p(:,2)];
val=p(:,idata);
figure;scatter(pos(:,1),pos(:,2),10,val(:,1),'filled');
        colorbar;title(dval);xlabel('X');ylabel('Y');axis image;

Example isotrop:
[garr,hc]=semivar_exp(pos,val);
plot(hc,garr);
xlabel('Distance (m)');ylabel('semivariance');title(dval)

Exmple directional
[garr,hc,hangc,gamma,h]=semivar_exp(pos,val,20,4);
plot(hc,garr);
legend(num2str(180*hangc'./pi))
xlabel('Distance (m)');ylabel('semivariance');title(dval)

```

### 7.1.89 semivar\_exp\_gstat

```

semivar_exp_gstat : Experimental semivariance using GSTAT

CALL :

[gamma,hc,np,av_dist]=semivar_exp_gstat(pos,val,angle,tol,width,cutoff)

IN :
pos : [ndata,ndims] : location of data
val : [ndata,1] : data values
angle [1] : angle (degrees)
tol [1] : angle tolerance around 'angle' (degrees)
width[1] : width of bin use to average semivariance
cutoff[1] : max distance for whoch to compute semivariance

'angle' and 'tol' are optional

defaults: angle=0;
          tol=180

OUT :
gamma : semivariance
hc : Seperation distance
np : Number of points for each seperation distance
av_dist : Average distance

EXAMPLE :
% GET JURA DATA
dwd=[mgstat_dir,filesep,'examples',filesep,'data',filesep,'jura',filesep];
[p,pHeader]=read_eas([dwd,'prediction.dat']);
idata=6;dval=pHeader{idata};

```

```

pos=[p(:,1) p(:,2)];
val=p(:,idata);
figure;scatter(pos(:,1),pos(:,2),10,val(:,1),'filled');
    colorbar;title(dval);xlabel('X');xlabel('Y');axis image;

% ISOTROP SEMIVARIOGRAM
[gamma,hc]=semivar_exp_gstat(pos,val);
figure;plot(hc,gamma);
title(dval);xlabel('Distance (m)');ylabel('Semivariance');

% ANISOTROPIC SEMIVARIOGRAM
hang=[0 45 90];
tol=10; % Angle tolerance
clear gamma;
figure
for ih=1:length(hang);
    [gamma,hc]=semivar_exp_gstat(pos,val,hang(ih),tol);
end
figure;plot(hc,gamma);
title(dval);xlabel('Distance (m)');ylabel('Semivariance');
legend(num2str(hang));

% ANISOTROPIC SEMIVARIOGRAM (2)
width=[0.1];
cutoff=[4];
hang=[0 45 90];
tol=10; % Angle tolerance
clear gamma;
for ih=1:length(hang);
    [gamma,hc]=semivar_exp_gstat(pos,val,hang(ih),tol,width,cutoff);
    p(ih)=plot(hc,gamma);hold on
    if ih==1, set(p(ih),'color',[0 0 0]);end
    if ih==2, set(p(ih),'color',[0 1 0]);end
    if ih==3, set(p(ih),'color',[0 0 1]);end
end
hold off
title(dval);xlabel('Distance (m)');ylabel('Semivariance');
legend(num2str(hang));

```

### 7.1.90 semivar\_map

semivar\_map : create 2D semivariogram map

See Goovaerts, p. 99

### 7.1.91 semivar\_optim

### 7.1.92 semivar\_synth

semivar\_synth : synthethic semivariogram

```

[sv,d]=semivar_synth(V,d,gstat);
V : Variogram model

```



```
d : seperation (array or matrix)
gstat : [0] use SGeMS semivariogram definitions (default)
gstat : [1] use GSTAT semivariogram definitions
```

Call ex :

```
[sv,d]=semivar_synth('0.1 Nug(0) + 1 Gau(1.5)', [0:.1:6]);plot(d,sv)
or :
V(1).par1=1;V(1).par2=1.5;V(1).type='Gau';
V(2).par1=0.1;V(2).par2=0;V(2).type='Nug';
[sv,d]=semivar_synth(V, [0:.1:6]);plot(d,sv)
```

### 7.1.93 set\_mgstat\_path

### 7.1.94 sgssim

sgssim : sequential Gaussian simulation

Call :

```
[sim_mul]=sgssim(pos_known,val_known,pos_sim,V,options);%
```

all arguments are the same as for 'krig.m', except the number of generated realizations can be set using :

```
options.nsim=10; (default is options.nsim=1)
```

note: this algorithm is very slow and for teaching purposes if you intend to simulate large fields use either the 'gstat' or 'mgstat' simulation options.

see also: krig

### 7.1.95 space2char

space2char : replace oen character with another in string

```
txtout=space2char(txt,charout,charin);
```

Example :

```
txt='Hello nice world';
space2char(txt)
ans = Hello_nice_world
space2char(txt,'+')
ans = Hello+nice+world
space2char(txt,'+','l')
ans = He++o nice wor+d
```

### 7.1.96 spherical\_spreading

```
sperical_spreading(r,type);
```

### 7.1.97 strip\_space

```
strip_space : strip leading/trailing spaces from string

CALL :
    txt=strip_space(txt,type);

    txt[string]
    type[integer] : [0] strip leading and trailing space (default);
    type[integer] : [1] strip leading space;
    type[integer] : [2] strip trailing space;

EX :
    a='      Hei Ho Here We Go      ';
    [''',strip_space(a),'']
    ans =
    'Hello World'

% strip leading space
[''',strip_space(a,1),'']
% strip trailing space
[''',strip_space(a,2),'']
```

### 7.1.98 suptitle

suptitle : Puts a title above all subplots.  
 SUPTITLE('text') adds text to the top of the figure  
 above all subplots (a "super title"). Use this function  
 after all subplot commands.

### 7.1.99 title\_alt

```
title_alt : Alternate title positioning

Call :
    title_alt(string,isub,location,dw,w_out)

title      [str] : title string
isub       [int] : Number of subplot. 1-->'a)' is prepended to the title
                                2-->'b)' is prepended to the title..
                                0--> use original string [Default]

location   [str] : 'NorthWestInside'
                    'NorthWestOutside' [Default]
                    'NorthEastInside'
                    'NorthEastOutside'

dw         [rea] : distance from edge to label, relative to plot size
                    [default dw=0.01];

w_out      [rea] : distance from edge to horizontal edge of label,
                    when location='*Outside', relative to plot size.
                    [default dw=0.2];
```

EXAMPLE :

```

figure
for i=1:5;
    subplot(2,3,i)
    imagesc(peaks(i*10))
    title_alt('Title',i);
end

figure
subplot(2,2,1);title_alt('NorthWestInside',i,'NorthWestInside');
subplot(2,2,2);title_alt('NorthWestOutside',i,'NorthWestOutside');
subplot(2,2,3);title_alt('NorthEastInside',i,'NorthEastInside');
subplot(2,2,4);title_alt('NorthEastOutside',i,'NorthEastOutside');

(C) TMH/2007

```

### 7.1.100 vonk2d

VONK2D.M : 2D Von Karman Distribution

Call : [randdata,x,z,data,expcorr]=vonk2d(rseed,dx,dz,ax,az,ix,iz,pop,med,nu,vel,frac)

rseed : Random Seed number

dx,dz : Spatial distance

ax,az : Horizontal, vertical lengthscale

ix,iz : size of model in same scale as ax,az

nu : Hurst Number

pop : Population, [1]:Gaussian [2]:PDF

med : Medium , [1]:Gaussian [2]:Exponential [3]: Von Karman  
[4]:Pink [5]:Brown

vel : Scalar or vector of velocities, For pop=pdf,v(1) is used as  
+/-max velocity of input field

frac : fraction assigned to each velocity (normalized), same size  
as vel

(C) 1998-2001 Thomas Mejer Hansen (tmh@gfy.ku.dk)

UPDATED APR 05 1999 /TMH

Octave 2.0.15 and Matlab 5.3 compliant

### 7.1.101 watermark

watermark : \_add watermark to figure : watermark(txt,FontSize);

Call

watermark(txt);

watermark(txt,FontSize);

ax=watermark(txt,FontSize,position);

### 7.1.102 write\_arcinfo\_ascii

write\_arcinfo\_ascii : Writes ascii formatted ArcInfo files

CALL :

```
[data,x,y,dx,nanval]=write_arcinfo_ascii(filename,data,x,y,nannumber,xll,yll);

filename [char] :string
data [ny,nx]
x [nx]
y [ny]
nannumber [1] : can be left empty []. Optional.
xll [char] : 'CENTER' (def) or 'CORNER'. Optional.
yll [char] : 'CENTER' or 'CORNER'. if 'xll' is set, yll=xll.
```

### 7.1.103 write\_bin

```
write_bin.m

CALL :
    write_bin(filename,variable,fchar,format,b_order);

REQUIRED :
    filename (string)
    variable : to be written to a binary file;

OPTIONAL
    fchar (scalar) : [1] Remove F77 chracters [0,default] do nothing
    format (string) : 'float32' [default] or 'int16' or 'int32',...
    b_order : set byteorder : '0' : Little Endian
```

### 7.1.104 write\_eas

```
write_eas : writes a GEO EAS formatted file into Matlab.

Call write_eas(filename,data,header,title,nanValue);

filename [string]
data [ndata,natts]
header [structure{natts}] : header values for data columns
title [string] : optional title for EAS file
nanValue [float] : NaN value

TMH (tmh@gfy.ku.dk)
```

### 7.1.105 write\_gstat\_ascii

### 7.1.106 write\_gstat\_par

```
write_gstat_par : write gstat.par file from Matlab structure

CALL :

    filename=write_gstat_par(G,filename);

input --:
G [struct]: gstat matlab structure
```

```

filename [string] : optional filename

output --:
filename [string] : filename of command file written to disk

```

### 7.1.107 write\_punch\_par

```

write_punch_par(filename,timefile,velfile,fxs,fys,fzs,nx,ny,nz,x0,y0,z0,h,reverse,maxoff) ←
;
Purpose : WRITES OUT PAR-FILE FOR USE WITH 'PUNCH' (John Hole)

CALL write_punch_par(filename,timefile,velfile,fxs,fys,fzs,nx,ny,nz,x0,y0,z0,h,reverse, ←
maxoff);
filename [string] : Name of par-file
timefile [string] : Name of output-time file
velfile [string] : Name of input velocity file
fxs [scalar] :
fys [scalar] :
fzs [scalar] :
nx [scalar] :
ny [scalar] :
nz [scalar] :
x0 [scalar] :
y0 [scalar] :
z0 [scalar] :
h [scalar] :
reverse [scalar] :
maxxoff [scalar] : Maximum offset

TMH 09/1999

```

### 7.1.108 write\_surfer\_grid

```

write_surfer_grid : Writes ascii formatted Surfer GRD file

CALL :
[data,x,y,dx,nanval]=write_surfer_grid(filename,data,x,y);

filename [char] :string
data [ny,nx]
x [nx]
y [ny]

```

## 7.2 VISIM functions

### 7.2.1 G\_to\_visim

### 7.2.2 calc\_gstat\_semivar

### 7.2.3 covar\_prob

---

### 7.2.4 read\_visim

---

### 7.2.5 semivar\_mat

---

### 7.2.6 visim

---

### 7.2.7 visim\_calc\_semivar\_prob

---

### 7.2.8 visim\_cholesky

---

### 7.2.9 visim\_clean

---

### 7.2.10 visim\_error\_sim

---

### 7.2.11 visim\_format\_variogram

---

### 7.2.12 visim\_init

---

### 7.2.13 visim\_lsq

---

### 7.2.14 visim\_make\_movie

---

**7.2.15 visim\_mask**

---

**7.2.16 visim\_merge\_volume\_data**

---

**7.2.17 visim\_movie**

---

**7.2.18 visim\_plot**

---

**7.2.19 visim\_plot\_condtab**

---

**7.2.20 visim\_plot\_etype**

---

**7.2.21 visim\_plot\_hist**

---

**7.2.22 visim\_plot\_kernel**

---

**7.2.23 visim\_plot\_kernel2**

---

**7.2.24 visim\_plot\_rpath**

---

**7.2.25 visim\_plot\_semivar**

---

**7.2.26 visim\_plot\_semivar2**

---

**7.2.27 visim\_plot\_semivar\_compare**

---

**7.2.28 visim\_plot\_semivar\_mul**

---

**7.2.29 visim\_plot\_semivar\_real**

---

**7.2.30 visim\_plot\_sim**

---

**7.2.31 visim\_plot\_stat**

---

**7.2.32 visim\_plot\_volfit**

---

**7.2.33 visim\_plot\_volume**

---

**7.2.34 visim\_precal\_linearization**

---

**7.2.35 visim\_prior\_prob**

---

**7.2.36 visim\_prior\_prob\_gibbs**

---

**7.2.37 visim\_prior\_prob\_mcmc**

---

**7.2.38 visim\_prior\_prob\_mcmc\_tomolin**

---



---

**7.2.39 visim\_prior\_prob\_sill**

---

**7.2.40 visim\_semivar**

---

**7.2.41 visim\_semivar\_pdf**

---

**7.2.42 visim\_semivar\_uncon**

---

**7.2.43 visim\_set\_broadband\_covariance**

---

**7.2.44 visim\_set\_conditional\_point**

---

**7.2.45 visim\_set\_multiscale\_covariance**

---

**7.2.46 visim\_set\_resim\_data**

---

**7.2.47 visim\_setup\_punch**

---

**7.2.48 visim\_setup\_tomo\_kernel**

---

**7.2.49 visim\_slice\_volume**

---

**7.2.50 visim\_to\_G**

---

### **7.2.51 visim\_tomo\_setup**

---

### **7.2.52 visim\_tomography**

---

### **7.2.53 visim\_tomography\_forward**

---

### **7.2.54 visim\_tomography\_linearize**

---

### **7.2.55 vvisim**

---

### **7.2.56 write\_visim**

---

## **7.3 SNESIM functions**

### **7.3.1 read\_snesim**

---

### **7.3.2 snesim**

---

### **7.3.3 snesim\_init**

---

### **7.3.4 snesim\_prior\_sampler**

---

### **7.3.5 snesim\_set\_resim\_data**

---

### **7.3.6 write\_snesim**

---

## 7.4 S-GeMS functions

### 7.4.1 eas2sgems

---

### 7.4.2 sgems

---

### 7.4.3 sgems2eas

---

### 7.4.4 sgems\_clean

---

### 7.4.5 sgems\_demo

---

### 7.4.6 sgems\_get\_par

---

### 7.4.7 sgems\_grid

---

### 7.4.8 sgems\_grid\_py

---

### 7.4.9 sgems\_image2ti

---

### 7.4.10 sgems\_plot\_project

---

### 7.4.11 sgems\_plot\_structure

---

### 7.4.12 sgems\_ppm

---

#### **7.4.13 sgems\_read**

---

#### **7.4.14 sgems\_read\_project**

---

#### **7.4.15 sgems\_read\_xml**

---

#### **7.4.16 sgems\_set\_resim\_data**

---

#### **7.4.17 sgems\_variogram\_xml**

---

#### **7.4.18 sgems\_write**

---

#### **7.4.19 sgems\_write\_grid**

---

#### **7.4.20 sgems\_write\_pointset**

---

#### **7.4.21 sgems\_write\_xml**

---

### **7.5 Misc functions**

#### **7.5.1 LowPass1**

---

#### **7.5.2 LowPass1nan**

---

#### **7.5.3 LowPass2**

---

#### 7.5.4 LowPass2nan

---

#### 7.5.5 accept\_rate

---

#### 7.5.6 caxis\_squeeze

---

#### 7.5.7 cmap\_quantile

---

#### 7.5.8 cmap\_rwb

---

#### 7.5.9 conv2\_strip

---

#### 7.5.10 conv\_strip

---

#### 7.5.11 despike

---

#### 7.5.12 figure\_focus

---

#### 7.5.13 find\_row\_array

---

#### 7.5.14 gamma\_coefficients

---

#### 7.5.15 gaussian\_likelihood

---

---

#### 7.5.16 generalized\_gaussian

---

#### 7.5.17 generalized\_gaussian\_2d

---

#### 7.5.18 isoctave

---

#### 7.5.19 jura

---

#### 7.5.20 kernel\_buursink\_2d

---

#### 7.5.21 kernel\_finite\_2d

---

#### 7.5.22 kernel\_fresnel\_2d

---

#### 7.5.23 kernel\_fresnel\_monochrome\_2d

---

#### 7.5.24 kernel\_multiple

---

#### 7.5.25 kernel\_slowness\_to\_velocity

---

#### 7.5.26 kml\_line

---

#### 7.5.27 kml\_points

---

---

### 7.5.28 mspectrum

---

### 7.5.29 munk\_fresnel\_2d

---

### 7.5.30 munk\_fresnel\_3d

---

### 7.5.31 pick\_first\_arrival

---

### 7.5.32 plot\_mh\_error\_hist

---

### 7.5.33 plot\_mh\_loglikelihood

---

### 7.5.34 punch

---

### 7.5.35 rayinv\_grid\_v

---

### 7.5.36 rayinv\_load\_tx

---

### 7.5.37 rayinv\_load\_v

---

### 7.5.38 rayinv\_plot\_tx

---

### 7.5.39 rickerwavelet

---

---

#### **7.5.40 sample\_generalized\_gaussian**

---

#### **7.5.41 set\_paper**

---

#### **7.5.42 set\_resim\_data**

---

#### **7.5.43 statusbar**

---

#### **7.5.44 su\_write\_model**

---

#### **7.5.45 test\_fd\_su\_gpr**

---

---