

HPGL: High Performance Geostatistics Library

**версия 0.9.6 BSD**

Руководство пользователя

2009

Оглавление

[1. Основные сведения 4](#_Toc240169002)

[1.1. Описание 4](#_Toc240169003)

[1.2. Требования к системе и установка 4](#_Toc240169004)

[1.2.1. Windows 5](#_Toc240169005)

[1.2.2. Ubuntu Linux 5](#_Toc240169006)

[1.2.3. Другие ОС 5](#_Toc240169007)

[1.3. Используемые компоненты 5](#_Toc240169008)

[2. Основная информация 6](#_Toc240169009)

[2.1. Модули библиотеки HPGL 6](#_Toc240169010)

[2.2. Создание IJK сетки 6](#_Toc240169011)

[2.3. Формат данных свойств 6](#_Toc240169012)

[2.4. Работа с файлами формата Eclipse Property 7](#_Toc240169013)

[2.4.1. Чтение файлов Eclipse Property 8](#_Toc240169014)

[2.4.2. Запись файлов Eclipse Property 8](#_Toc240169015)

[2.5. Работа с файлами формата GSLIB 9](#_Toc240169016)

[2.5.1. Чтение файлов GSLIB 9](#_Toc240169017)

[2.5.2. Запись файлов GSLIB 9](#_Toc240169018)

[2.6. Объект ковариация (вариограмма) 10](#_Toc240169019)

[2.7. Удаление данных из памяти 10](#_Toc240169020)

[3. Использование алгоритмов 11](#_Toc240169021)

[3.1. Simple Kriging 11](#_Toc240169022)

[3.2. Ordinary Kriging 12](#_Toc240169023)

[3.3. Indicator Kriging 13](#_Toc240169024)

[3.4. LVM Kriging (Local Varying Mean) 14](#_Toc240169025)

[3.5. Sequential Indicator Simulation (SIS) 15](#_Toc240169026)

[3.6. Sequential Gaussian Simulation (SGS) 16](#_Toc240169027)

[4. Описание подмодулей 18](#_Toc240169028)

[4.1. geo.routines 18](#_Toc240169029)

[4.1.1. Расчет средних значений 18](#_Toc240169030)

[4.1.2. Расчет VPC (Vertical Proportion Curve) 18](#_Toc240169031)

[4.1.3. Работа с GSLIB-файлами 19](#_Toc240169032)

[4.1.4. Расчет «ползущего среднего» 20](#_Toc240169033)

[Авторы - Контакты 22](#_Toc240169034)

[Список изменений 23](#_Toc240169035)

[Лицензия 25](#_Toc240169036)

# 1. Основные сведения

## 1.1. Описание

HPGL является С++/Python библиотекой, содержащей геостатистические алгоритмы. Использование алгоритмов осуществляется командами на языке Python, благодаря чему возможно создание необходимых сценариев геологического моделирования.

В версии **0.9.6 BSD** реализованы следующие алгоритмы:

* Simple Kriging (SK)
* Ordinary Kriging (OK)
* Indicator Kriging (IK)
* Local Varying Mean Kriging (LVM Kriging)
* Simple CoKriging (Markov Models 1 & 2)
* Sequential Indicator Simulation (SIS)
* Correlogram Local Varying Mean SIS (CLVM SIS)
* Local Varying Mean SIS (LVM SIS)
* Sequential Gaussian Simulation (SGS)
* Local Varying Mean SGS (LVM SGS)
* Truncated Gaussian Simulation (GTSIM)\*

\* в наборе скриптов на Python

Распространение свойств производится в IJK пространстве, поэтому все параметры (например, радиусы вариограмм и эллипсоида) задаются в ячейках сетки.

Поддерживаются текстовые форматы загрузки/выгрузки данных Eclipse и GSLIB, форматом данных для алгоритмов являются NumPy-массивы (numpy.ndarray).

## 1.2. Требования к системе и установка

Для работы HPGL потребуется операционная система Windows (32 бита) или Linux (32/64 бита), с установленными в системе Python версии 2.5/2.6 и пакетами NumPy/SciPy для соответствующей версии Python.

### 1.2.1. Windows

В операционных системах Windows потребуется установить Microsoft Visual C++ 2005 SP1 Redistributable Package (можно скачать по адресу <http://www.microsoft.com/downloads/details.aspx?familyid=2051A0C1-C9B5-4B0A-A8F5-770A549FD78C&displaylang=en>).

**ВНИМАНИЕ!** Обратите внимание на дату этого обновления (7/28/2009). Более старые версии Visual C++ Redistributable Package работать не будут!

Для установки HPGL, используйте файл **HPGL-X.Y.Z-BSD-[py2.5/py2.6].win32.exe**, соответствующий установленной в системе версии Python.

### 1.2.2. Ubuntu Linux

Ubuntu (32-bit / 64-bit):

Установите пакет **hpgl\_X.Y.Z-BSD-[x32/x64].deb,** соответствующий архитектуре вашей системы.

Для работы HPGL также понадобится установить библиотеки Boost (в частности, boost::python).

### 

### 1.2.3. Другие ОС

Если вам понадобятся сборки под другие ОС (в частности, под другие версии Linux), обращайтесь к авторам по электронной почте. (см. Авторы – Контакты в конце документа).

## 1.3. Используемые компоненты

В библиотеке используются следующие свободно распространяемые компоненты:

- TNT (Template Numerical Toolkit) - <http://math.nist.gov/tnt/overview.html>;

- библиотека boost (в частности, boost::python).

# 2. Основная информация

## 2.1. Модули библиотеки HPGL

Каждый скрипт на языке Python, в котором будет использоваться HPGL, должен начинаться строкой импорта модуля **geo**:

from geo import \*

Кроме основного модуля **geo**, HPGL также содержит подмодуль **geo.routines**.

Модуль **geo.routines** содержит различные функции для дополнительной работы со свойствами, такие как расчет «ползущего» среднего, VPC (Vertical Proportion Curve), в т.ч. в виде VPC-кубов, а также функции для работы с данными формата GSLIB (PointSet и CartesianGrid).

Для импорта данных модулей в начале скрипта нужно написать:

from geo.routines import \*

Более подробную информацию о содержании каждого модуля смотрите в соответствующем разделе документации (п. 4).

## 2.2. Создание IJK сетки

IJK-сетка (т.е. сетка, заданная ячейками), является основой для выполнения геостатистических алгоритмов и ее необходимо создавать перед вызовом любого из них.

Создание IJK сетки производится при помощи функции **SugarboxGrid**:

**grid\_object** = SugarboxGrid(**I**, **J**, **K**)

В результате выполнения команды будет создан объект - сетка **grid\_object** с размерностями **I, J, K.**

**Пример:**

my\_griddy = SugarboxGrid(42, 42, 10)

## 2.3. Формат данных свойств

Все свойства, заданные на сетке, в HPGL являются кортежем (tuple) Python из двух массивов (numpy.ndarray) NumPy и имеют следующую структуру:

property = (array\_prop, array\_informed)

где **array\_prop** – NumPy-массив типа *float32* [для непрерывных свойств] или *uint8* [для индикаторных] с данными;

**array\_informed** – NumPy-массив типа *uint8*, показывающий, какие ячейки массива **array\_prop** имеют определенное значение (**array\_informed = 1**), а какие - не определены (**array\_informed = 0**),

Индикаторные свойства, в дополнение к этим двум массивам, должны содержать еще один элемент:

property = (array\_prop, array\_informed, **indicators\_number**)

где **indicators\_number** – количество индикаторов в **array\_prop**.

***ВНИМАНИЕ!*** В numpy-массивах, которые передаются в алгоритмы, нужно использовать фортрановский порядок хранения данных. Фортрановский порядок хранения для numpy-массивов можно задать следующим образом:

- при создании новых массивов:

a = array([], **order=’F’**)

- для преобразования ранее созданных массивов:

a = **require(**a, **requirements=’F’)**

Если в алгоритм будет передан numpy-массив с не-фортрановским порядком данных, он будет автоматически преобразован в нужный формат. Таким образом, нужно иметь в виду, что *возвращаемые алгоритмами массивы всегда имеют фортрановский порядок данных*.

Подробнее о фортрановском порядке хранения массивов можно узнать здесь - <http://www.ibiblio.org/pub/languages/fortran/ch2-6.html>

## 2.4. Работа с файлами формата Eclipse Property

В HPGL поддерживается чтение и запись данных в текстовом формате файлов Eclipse Property, которые имеют следующую структуру:

-- комментарий (игнорируется)

PROPERTY\_NAME

0

1

0

...

/

Файл формата данных должен содержать последовательно записанные значения из сетки, в порядке *i=0…max, j=max…0, k=0…max*.

### 2.4.1. Чтение файлов Eclipse Property

Чтение данных из файлов формата Eclipse Property производится при помощи двух функций:

* load\_ind\_property - для индикаторных величин;
* load\_cont\_property - для непрерывных величин.

prop = load\_cont\_property(filename, undefined\_value, size)

prop = load\_ind\_property(filename, undefined\_value, [indicators], size)

В результате выполнения команды будет создан объект **prop**, являющийся свойством HPGL *(см. 2.3),* куда будут загружены данные из файла **filename**. Ячейки со значением **undefined\_value** будут считаться пустыми (*undefined*), и **array\_informed** для них будет равен 0.

В качестве предпоследнего параметра функции **load\_ind\_property** передаются коды индикаторов, находящихся в файле.

**ВНИМАНИЕ!** После загрузки индикаторных данных из файла, все индикаторы будут переименованы в 0,1,2,3… по порядку, заданному в **indicators**.

Последним параметром передаются размеры сетки **i,j,k** для моделирования, в виде кортежа (tuple) Python:

size = (i,j,k)

**Пример:**

size = (50, 50, 100)

cont\_prop = load\_cont\_property("d:\CONT.INC", -99, size)

ind\_prop = load\_ind\_property("d:\IND.INC", -99, [0,1], size)

### 2.4.2. Запись файлов Eclipse Property

Сохранение свойства HPGL в файл Eclipse Property производится при помощи функции **write\_property**:

write\_property(prop\_object, filename, prop\_name, undefined\_value, indicator\_values=[])

В результате выполнения команды будет создан файл **filename**, в который будет записано свойство из объекта **prop\_object** под именем **prop\_name**. На место пустых (**array\_informed = 0**) ячеек будет записано значение **undefined\_value**. Индикаторы будут записаны с индексами, указанными в **indicator\_values** (если не задано, они будут сохранены как 0,1,2,… по порядку).

**Пример:**

write\_property(cont\_prop, "CON\_PROP.INC", "PROPCON", -99)

write\_property(i\_prop, "INDP.INC", "PROP\_IND", -99, [0,1])

## 2.5. Работа с файлами формата GSLIB

Подробное описание формата GSLIB можно прочитать на <http://www.gslib.com/gslib_help/format.html>. Функции для работы с файлами GSLIB содержаться в модуле **geo.routines**., поэтому его нужно импортировать перед их использованием:

from geo.routines import \*

### 2.5.1. Чтение файлов GSLIB

Чтение свойств из файла формата GSLIB производится при помощи функции **LoadGslibFile**:

dict\_gslib = LoadGslibFile(**filename**)

где **filename** – это имя файла;

**dict\_gslib** – словарь (dictionary) Python с содержимым файла (каждый элемент этого словаря будет NumPy-массивом).

Обращение к свойству **property\_1** осуществляется следующим образом:

dict\_gslib[“property\_1”]

### 2.5.2. Запись файлов GSLIB

Запись свойств в файл формата GSLIB производится при помощи функции **SaveGSLIBCubes**:

SaveGSLIBCubes(dict\_gslib, **filename**, сaption, Format = "%d")

где **filename** – это имя файла;

**dict\_gslib** – словарь (dictionary) Python со свойствами (каждый элемент словаря должен быть NumPy-массивом);

**caption** – первая строка в записываемом файле (заголовок)

Подробно работа со словарями описана в соответствующем разделе справки Python (например, здесь –

[http://ru.wikibooks.org/wiki/%D0%A3%D1%87%D0%B5%D0%B1%D0%BD%D0%B8%D0%BA\_Python\_2.6#.D0.A1.D0.BB.D0.BE.D0.B2.D0.B0.D1.80.D0.B8](http://ru.wikibooks.org/wiki/%D0%A3%D1%87%D0%B5%D0%B1%D0%BD%D0%B8%D0%BA_Python_2.6%23.D0.A1.D0.BB.D0.BE.D0.B2.D0.B0.D1.80.D0.B8)).

## 2.6. Объект ковариация (вариограмма)

Все геостатистические алгоритмы HPGL используют единый тип ковариации (вариограммы), который задается при помощи объекта класса **CovarianceModel**:

cov = CovarianceModel(

**type** = 0,

**ranges**=(0,0,0),

**angles**=(0,0,0),

**sill**=1.0,

**nugget**=0.0)

где **type** – тип вариограммы:

**0** – сферическая, **1** – экспоненциальная, **2** – гауссовая;

**ranges** – радиусы эллипсоида вариограммы (0⁰, 90⁰, вертикальный);

**angles** – углы наклона эллипсоида вариограммы;

**sill** – пороговое значение вариограммы;

**nugget** – величина “nugget”-эффекта.

Созданный таким образом объект **cov** можно использовать для всех алгоритмов HPGL.

## 2.7. Удаление данных из памяти

Когда свойство уже не нужно, его можно удалить, освободив оперативную память. Это можно сделать при помощи команды **del:**

del(**prop\_object**)

# 3. Использование алгоритмов

## 3.1. Simple Kriging

Вызов алгоритма Simple Kriging (SK) осуществляется при помощи функции **simple\_kriging:**

def simple\_kriging(

prop, # свойство с изначальными данными (hard data)

grid, # сетка, на которой производится SK

radiuses, # радиусы эллипсоида поиска.

max\_neighbours, # максимальное число точек, участвующих в интерполяции

cov\_model, # объект ковариации (см п. 2.6)

mean=**None** # среднее значение

# если не задано, вычисляется автоматически по

# изначальным данным

)

**Пример:**

size = (55, 52, 100)

grid = SugarboxGrid(55, 52, 100)

prop = load\_cont\_property("HARD\_DATA.INC", -99, size )

cov\_krig = CovarianceModel(type=1, ranges=(10,10,10), sill=1)

prop\_result = simple\_kriging(prop, grid,

radiuses = (20, 20, 20),

max\_neighbours = 12,

cov\_model = cov\_krig,

mean = 1.6)

write\_property(prop\_result, "SK.INC", "SK\_RESULT", -99)

del(prop\_result)

del(prop\_result)

## 3.2. Ordinary Kriging

Вызов алгоритма Ordinary Kriging (OK) осуществляется при помощи функции **ordinary\_kriging:**

def ordinary\_kriging(

prop, # свойство с изначальными данными (hard data)

grid, # сетка, на которой производится OK

radiuses, # радиусы эллипсоида поиска

max\_neighbours, # максимальное число точек участвующее в интерполяции

cov\_model, # объект ковариации (см п. 2.6)

)

**Пример:**

size = (55, 52, 100)

grid = SugarboxGrid(55, 52, 100)

prop = load\_cont\_property("HARD\_DATA.INC", -99, size )

cov\_krig = CovarianceModel(type=1, ranges=(10,10,10), sill=1)

prop\_result = ordinary\_kriging(prop, grid,

radiuses = (20, 20, 20),

max\_neighbours = 12,

cov\_model = cov\_krig)

write\_property(prop\_result, "R\_OK.INC", "OK\_RESULT", -99)

del(prop\_result)

## 3.3. Indicator Kriging

Перед вызовом функции **indicator\_kriging**, выполняющей алгоритм Indicator Kriging (IK), необходимо создать список Python (list), содержащий параметры моделирования.

Ниже приведен пример такого списка:

data = [

# Параметры 0-го индикатора

{

"cov\_model": **cov0**, # объект ковариации (см. 2.6)

"radiuses": (**SR1**, **SR2**, **SR3**), # радиусы эллипсоида поиска

"max\_neighbours": **neigh\_count**,

# максимальное количество соседей

# для интерполяции

},

# Параметры 1-го индикатора

{

"cov\_model": **cov1**, # объект ковариации (см. 2.6)

"radiuses": (**SR1**, **SR2**, **SR3**), # радиусы эллипсоида поиска

"max\_neighbours": **neigh\_count**,

# максимальное количество соседей

# для интерполяции

}

]

Для каждого существующего индикатора задаются свои параметры.

***Примечание:*** При использовании только двух индикаторов автоматически выполняется алгоритм Median IK.

Когда структура параметров заполнена, можно вызывать на исполнение алгоритм, используя функцию **indicator\_kriging**:

def indicator\_kriging

(

ik\_prop, # свойства с исходными данными

grid, # сетка, на которой выполняется алгоритм

data, # список параметров для индикаторов

marginal\_probs # кортеж (tuple) Python с априорными

# вероятностями для каждого индикатора

)

**Пример:**

size = (55, 52, 100)

grid = SugarboxGrid(55, 52, 100)

prop = load\_ind\_property("HARDDATA.INC", -99, [0,1], size)

cov1 = CovarianceModel(type=1, ranges=(10,10,10), sill=1)

data = [ {

"cov\_model": cov1,

"radiuses": (20, 20, 20),

"max\_neighbours": 12,

},

{

"cov\_model": cov1,

"radiuses": (20, 20, 20),

"max\_neighbours": 12,

}]

ik\_result = indicator\_kriging(prop, grid, data, (0.8, 0.2))

write\_property(ik\_result, "RESIK.INC", "PROP\_IK", -99, [0,1])

## 3.4. LVM Kriging (Local Varying Mean)

Кригинг с изменяющимся средним (LVM) производится при помощи функции **lvm\_kriging**:

def lvm\_kriging

(

prop, # свойства с исходными данными (hard data)

grid, # сетка, на которой производится LVM Kriging

mean\_data, # numpy array (типа float32),

# содержащий средние значения свойства для LVM

radiuses, # радиусы эллипсоида поиска

max\_neighbours, # максимальное число точек участвующее в интерполяции

cov\_model, # объект ковариации (см п. 2.6)

)

**Пример:**

grid = SugarboxGrid(55, 52, 100)

size = (55, 52, 100)

mean\_data = load\_cont\_property("cube\_local\_means.inc", size)[0]

cov1 = CovarianceModel(type=1, ranges=(10,10,10), sill=1)

lvm\_prop = load\_cont\_property("LVM.INC", -99, size)

prop\_lvm = lvm\_kriging(lvm\_prop, grid, mean\_data,

radiuses = (20, 20, 20),

max\_neighbours = 12,

cov\_model = cov1)

write\_property(prop\_lvm, "lvmresult.inc", "lvm\_kriging", -99)

del(mean\_data)

del(prop\_lvm)

## 3.5. Sequential Indicator Simulation (SIS)

Параметры моделирования для алгоритма SIS задаются аналогично Indicator Kriging (см. п. 3.3), путем создания списка с параметрами для индикаторов.

Вызов функции:

def sis\_simulation(

prop, # свойство с исходными данными (hard data)

grid, # сетка, на которой выполняется алгоритм

data, # список с параметрами алгоритма (см. п. 3.3)

seed, # номер стохастической реализации

marginal\_probs, # априорные вероятности для индикаторов.

# если заданы в виде кортежа (tuple) с вероятностями,

# то выполняется обычный SIS; если в виде кортежа с Numpy

# массивами (вероятностные кубы) – то SIS LVM.

use\_correlogram = **True**,

# задает использование коррелограмного SIS LVM

# или «классического» SIS LVM

# **True** — использовать коррелограмный SIS

# **False** — использовать LVM SIS

mask = **None**,

#свойство, задающее регион моделирования (если необходимо

# смоделировать не все точки сетки, а только часть).

# должно быть задано в виде numpy array (типа uint8) со значениями

# **0** (не моделируется) и **1** (моделируется).

)

***Примечание:*** При использовании только двух индикаторов автоматически выполняется алгоритм Median SIS.

**Пример:**

size = (55, 52, 100)

grid = SugarboxGrid(55, 52, 100)

sis\_prop = load\_ind\_property("HARD.INC", -99, [0,1], size)

cov1 = CovarianceModel(type=1, ranges=(10,10,10), sill=1)

sis\_data = [ {

"cov\_model": cov1,

"radiuses": (20, 20, 20),

"max\_neighbours": 12,

},

{

"cov\_model": cov1,

"radiuses": (20, 20, 20),

"max\_neighbours": 12,

}]

sis\_result = sis\_simulation(sis\_prop, grid, sis\_data, seed=3241347)

write\_property(sis\_result, "RESSIS.INC", "P\_SIS", -99, [0,1])

## 3.6. Sequential Gaussian Simulation (SGS)

Вызов алгоритма SGS производится при помощи функции **sgs\_simulation**:

def sgs\_simulation(

prop, # свойство с исходными данными (hard data)

grid, # сетка, на которой выполняется алгоритм

radiuses, # радиусы эллипсоида поиска.

max\_neighbours, # максимальное число точек, участвующих в интерполяции

cov\_model, # объект ковариации (см п. 2.6)

seed, # номер стохастической реализации

kriging\_type = **“sk”**, # тип Кригинга, используемого в алгоритме

# **sk** – Simple Kriging

# **ok** – Ordinary Kriging

# игнорируется, если среднее задано в виде куба (для LVM)

mean = **None,** # среднее значение моделируемого свойства

# если задано в виде числа, то выполняется «обычный» SGS

# если в виде свойства HPGL – выполняется SGS LVM.

use\_harddata = **True,**

#если **False**, исходные данные использоваться не будут, в этом случае

# будет произведено безусловное моделирование, с гистограммой,

# полученной из массива **cdf\_data**

cdf\_data = **None,**

# используется только если **use\_harddata = True**

#numpy-массив (типа float32), который используется в качестве

# источника гистограммы при некондиционном моделировании

mask = **None**,

#свойство, задающее регион моделирования (если необходимо

# смоделировать не все точки сетки, а только часть).

# должно быть задано в виде numpy array (типа uint8) со значениями

# **0** (не моделируется) и **1** (моделируется).

)

**Пример:**

size = (55, 52, 100)

grid = SugarboxGrid(55, 52, 100)

prop = load\_cont\_property("SGS\_HARD\_DATA.INC", -99, size)

cov1 = CovarianceModel(type=1, ranges=(10,10,10), sill=1)

sgs\_result = sgs\_simulation(prop, grid,

radiuses = (20,20,20),

max\_neighbours = 12,

cov\_model = cov1,

seed=3439275)

write\_property(sgs\_result, "RSGS.INC", "PROP\_SGS", -99)

## Пример(LVM):

grid = SugarboxGrid(55, 52, 100)

size = (55, 52, 100)

prop = load\_cont\_property("HARD\_DATA.INC", -99, size )

mean\_data = load\_cont\_property("MEAN.INC", -99, size )[0]

sgs\_lvm\_result = sgs\_result = sgs\_simulation(prop, grid,

radiuses = (20,20,20),

max\_neighbours = 12,

cov\_model = cov1,

seed=3439275,

mean = mean\_data)

write\_property(sgs\_lvm, "SGS\_LVM\_RESULT.INC", "SGS\_LVM", -99)

del(sgs\_lvm)

# 4. Описание подмодулей

## 4.1. geo.routines

Подмодуль **geo.routines** содержит различные дополнительные функции для работы со свойствами HPGL.

### 4.1.1. Расчет средних значений

1. **CalcMean** - возвращает среднее значение в Numpy-массиве **Cube** для определенных (**Mask = 1**) ячеек:

**mean** = CalcMean(**Cube**, **Mask**)

1. **CalcMarginalProbsIndicator** – возвращает Numpy-массив с долями (т.е. априорными вероятностями) для каждого индикатора куба **Cube**, заданного в **Indicators**, для определенных (**Mask = 1**) ячеек:

**MProbs** = CalcMarginalProbsIndicator(**Cube**, **Mask**, **Indicators**)

### 4.1.2. Расчет VPC (Vertical Proportion Curve)

1. **CalcVPC** - возвращает Numpy-массив c VPC (*Vertical Proportion Curve*) - средними значениями по вертикальным слоям Numpy-массива **Cube**, для определенных (**Mask = 1**) ячеек:

**VPC** = CalcVPC(**Cube**, **Mask**, **MarginalMean**)

В качестве последнего параметра - **MarginalMean** - нужно передать среднее значение свойства, по которому строится VPC. Это значение будет записано в VPC для тех слоев, где свойство не имеет определенных (**Mask = 1**) ячеек.

1. **CalcVPCsIndicator** - возвращает список (list) Python c Numpy-массивами VPC (*Vertical Proportion Curve*) - средними значениями по вертикальным слоям Numpy-массива **Cube** – для каждого из индикаторов, заданных в **Indicators**, для определенных (**Mask = 1**) ячеек:

**Result** = CalcVPCsIndicator(**Cube**, **Mask**, **Indicators**, **MarginalProbs**)

В качестве последнего параметра - **MarginalProbs** - нужно передать средние значения (т.е. априорные вероятности) для каждого из индикаторов, по которым строится VPC. Эти значения будут записаны в VPC для тех слоев, где свойство не имеет определенных (**Mask = 1**) ячеек.

с) **CubeFromVPC** – создает трехмерный Numpy-массив размерности **NX**, **NY**, len(**VPC**), заполненный значениями из **VPС** для каждого вертикального слоя.

**VPC\_Cube** = CubeFromVPC(**VPC**, **NX**, **NY**)

Полученный таким образом массив **VPC\_Cube** можно использовать в качестве куба средних значений (*mean / marginal\_probs*) для непрерывных алгоритмов с изменяющимся средним (SGS LVM, LVM Kriging).

Данная функция предназначена для использования совместно с функцией **CalcVPC**.

с) **CubesFromVPCs** – создает список (list) трехмерных Numpy-массивов размерности **NX**, **NY**, len(**VPC**), заполненных значениями из соответствующих **VPСs** для каждого вертикального слоя.

**VPC\_Cubes** = CubesFromVPCs(**VPCs**, **NX**, **NY**)

Полученный таким образом массив **VPC\_Cubes** можно использовать в качестве куба средних значений (*mean / marginal\_probs*) для индикаторных алгоритмов с изменяющимся средним (SIS LVM).

Данная функция предназначена для использования совместно с функцией **CalcVPCsIndicator**.

### 4.1.3. Работа с GSLIB-файлами

Функции чтения и записи GSLIB-файлов, находящиеся в данном модуле, подробно описаны в п. 2.5. Ниже представлены дополнительные функции для работы со свойствами этого типа.

1. **Cubes2PointSet** – функция для преобразования словаря (*dictionary*) с GSLIB-свойствами в формат GSLIB Point Set:

**PointSets** = Cubes2PointSet(**CubesDictionary**, **Mask**)

Здесь:

**-**  **CubesDictionary – словарь со свойствами GSLIB;**

**- Mask** – массив, задающий определенные **(Mask = 1)** и не определенные **(Mask = 0)** ячейки**.**

1. **Cube2PointSet** – функция для преобразования определенных ячеек **(Mask = 1)** Numpy-массива (**Cube)** в GSLIB Point Set:

**PointSet** = Cube2PointSet(**Cube**, **Mask**)

**с) PointSet2Cube** – функция для преобразования GSLIB Point Set’а в свойство HPGL:

**Cube**, **Mask** = PointSet2Cube(**X**, **Y**, **Z**, **Property**, **Cube**)

Здесь:

- **Cube** – Numpy-массив, куда нужно записать точки Point Set’а;

- **Mask** – Numpy-массив, показывающий какие из ячеек Cube определены (Mask = 1), а какие нет (Mask = 0);

- **X** – координаты X для всех точек Point Set’a;

- **Y** – координаты Y для всех точек Point Set’a;

- **Z** – координаты Z для всех точек Point Set’a;

- **Property** – Numpy-массив, содержащий значения свойства в точках Point Set’а.

***Примечание:*** параметр Cube должен передаваться в эту функцию уже инициализированным с необходимой размерностью. После выполнения алгоритма, значения в нем будут перезаписаны значениями из Point Set’a.

e) **SaveGSLIBPointSet** – функция для сохранения GSLIB Point Set (**PointSet**) в файл (**FileName**) формата GSLIB с заданным заголовком (**Caption**):

SaveGSLIBPointSet(**PointSet**, **FileName**, **Caption**)

### 4.1.4. Расчет «ползущего среднего»

Функция расчета «ползущего среднего» позволяет получить numpy-массив, который можно использовать как куб средних в алгоритмах с изменяющимся средним (SIS LVM, SGS LVM, LVM Kriging).

Для того, чтобы рассчитать массив «ползущего среднего» **MACube** по определенным **(Mask = 1)** ячейкам numpy-массива **Cube**, нужно вызвать функцию **MovingAverage3D**:

**MACube** = MovingAverage3D((**Cube**, **Mask**), **Radiuses**, **undefined**\_value, **MaskCalcFunction**)

Здесь:

- **Radiuses** – кортеж (tuple) из трех чисел, задающих окрестность для расчета «ползущего среднего» по каждой из осей (I, J, K);

- **undefined\_value** – значение, которое будет записано в ячейки массива **MACube**, для которых не удалось рассчитать «ползущее среднее»;

- **MaskCalcFunction** – указатель на функцию, создающую шаблон «ползущего среднего», может принимать значения:

- **GetCubicalMask** – для кубического шаблона;

- **GetEllipseMask** – для эллипсоидного шаблона;

**Пример:**

size\_prop = [166, 141, 20]

undef = -99

prop = load\_cont\_property("DATA.INC", undef, size\_prop)

Radiuses = (10, 10, 10)

# Расчет кубического «ползущего среднего»

MACube = MovingAverage3DP(prop, Radiuses, undef, GetCubicalMask)

Авторы - Контакты

Савичев Владимир

Безруков Андрей

Мухарлямов Артур

Барский Константин

Насибуллина Дина

По всем вопросам обращайтесь по адресу:

hpgl-support-rus@﻿lists.sourceforge.net.

# Список изменений

**HPGL 0.9.6** *- 14/09/2009*

* Появился подмодуль **geo.routines**
* Новая структура модуля **geo** (множество изменений в интерфейсе алгоритмов) – большая часть документации переписана
* **SGS LVM**: изменена логика работы алгоритма, теперь заданное среднее воспроизводится точно
* **IK/SIS**: Теперь для двух индикаторов в IK/SIS всегда выполняется Median-версии этих алгоритмов
* **SGS:** Исправлена ошибка с использованием внешнего свойства для CDF в SGS – работало неверно
* Исправлена ошибка в генераторе случайного пути, использующемся для всех алгоритмов (выдавал ошибочные значения для маленьких сеток – меньше 100 ячеек)
* Изменена схема компиляции проекта
* Добавлены автоматические сборки пакетов под Python 2.5 и 2.6 (Windows + Linux)
* Требование фортрановского порядка к входным массивам алгоритмов теперь опционально (массивы будут автоматически преобразованы внутри алгоритмов).
* Новые функции чтения GSLIB-файлов и расчета VPC – гораздо более быстрые
* Добавлена проверка Sill > Nugget

**HPGL 0.9.5** *- 22/05/2009*

* Свойства теперь хранятся в формате массивов NumPy array.
* Поддержка формата файлов GSLIB
* Добавлена возможность моделирования без изначальных данных (некондиционное моделирование)
* Почти все алгоритмы (кроме Ordinary Kriging) теперь при решении СЛАУ используют метод разложения Холецкого, благодаря чему производительность алгоритмов повышена в 2 раза.
* boost::python теперь слинкован статически.

**HPGL 0.9.4** *- 12/05/2009*

* GsTL больше не используется.
* Библиотека переведена на лицензию BSD.
* Добавлена поддержка параметра nugget и анизотропных вариограмм.
* Новая внутренняя структура алгоритмов.
* Поддержка регионов моделирования в SIS/SGS

**HPGL 0.9.3** *- 06/04/2009*

* Первая открытая версия

# Лицензия

HPGL распространяется на условиях лицензии BSD.

Далее приведен полный текст лицензии.

\* Copyright (c) 2009, HPGL Team

\*

\* Разрешается повторное распространение и использование как в виде исходного

\* кода, так и в двоичной форме, с изменениями или без, при соблюдении

\* следующих условий:

\*

\* \* При повторном распространении исходного кода должно оставаться

\* указанное выше уведомление об авторском праве, этот список условий и

\* последующий отказ от гарантий.

\* \* При повторном распространении двоичного кода должна сохраняться

\* указанная выше информация об авторском праве, этот список условий и

\* последующий отказ от гарантий в документации и/или в других

\* материалах, поставляемых при распространении.

\* \* Ни название HPGL, ни имена ее сотрудников не могут быть

\* использованы в качестве поддержки или продвижения продуктов,

\* основанных на этом ПО без предварительного письменного разрешения.

\*

\* ЭТА ПРОГРАММА ПРЕДОСТАВЛЕНА ВЛАДЕЛЬЦАМИ АВТОРСКИХ ПРАВ И/ИЛИ ДРУГИМИ

\* СТОРОНАМИ "КАК ОНА ЕСТЬ" БЕЗ КАКОГО-ЛИБО ВИДА ГАРАНТИЙ, ВЫРАЖЕННЫХ ЯВНО

\* ИЛИ ПОДРАЗУМЕВАЕМЫХ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ИМИ, ПОДРАЗУМЕВАЕМЫЕ

\* ГАРАНТИИ КОММЕРЧЕСКОЙ ЦЕННОСТИ И ПРИГОДНОСТИ ДЛЯ КОНКРЕТНОЙ ЦЕЛИ. НИ В

\* КОЕМ СЛУЧАЕ, ЕСЛИ НЕ ТРЕБУЕТСЯ СООТВЕТСТВУЮЩИМ ЗАКОНОМ, ИЛИ НЕ УСТАНОВЛЕНО

\* В УСТНОЙ ФОРМЕ, НИ ОДИН ВЛАДЕЛЕЦ АВТОРСКИХ ПРАВ И НИ ОДНО ДРУГОЕ ЛИЦО,

\* КОТОРОЕ МОЖЕТ ИЗМЕНЯТЬ И/ИЛИ ПОВТОРНО РАСПРОСТРАНЯТЬ ПРОГРАММУ, КАК БЫЛО

\* СКАЗАНО ВЫШЕ, НЕ НЕСЁТ ОТВЕТСТВЕННОСТИ, ВКЛЮЧАЯ ЛЮБЫЕ ОБЩИЕ, СЛУЧАЙНЫЕ,

\* СПЕЦИАЛЬНЫЕ ИЛИ ПОСЛЕДОВАВШИЕ УБЫТКИ, ВСЛЕДСТВИЕ ИСПОЛЬЗОВАНИЯ ИЛИ

\* НЕВОЗМОЖНОСТИ ИСПОЛЬЗОВАНИЯ ПРОГРАММЫ (ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ

\* ПОТЕРЕЙ ДАННЫХ, ИЛИ ДАННЫМИ, СТАВШИМИ НЕПРАВИЛЬНЫМИ, ИЛИ ПОТЕРЯМИ

\* ПРИНЕСЕННЫМИ ИЗ-ЗА ВАС ИЛИ ТРЕТЬИХ ЛИЦ, ИЛИ ОТКАЗОМ ПРОГРАММЫ РАБОТАТЬ

\* СОВМЕСТНО С ДРУГИМИ ПРОГРАММАМИ), ДАЖЕ ЕСЛИ ТАКОЙ ВЛАДЕЛЕЦ ИЛИ ДРУГОЕ

\* ЛИЦО БЫЛИ ИЗВЕЩЕНЫ О ВОЗМОЖНОСТИ ТАКИХ УБЫТКОВ.