

HPGL: High Perfomance Geostatistics Library

**версия 0.9.4**

Руководство пользователя

2009

Оглавление

[1. Основные сведения 4](#_Toc229892706)

[1.1. Требования к системе 4](#_Toc229892707)

[1.2. Описание 4](#_Toc229892708)

[1.3. Используемые компоненты 5](#_Toc229892709)

[1.4. Установка 5](#_Toc229892710)

[1.4.1. Windows 5](#_Toc229892711)

[1.4.2. Linux 5](#_Toc229892712)

[2. Основные функции 6](#_Toc229892713)

[2.1. Импорт библиотек 6](#_Toc229892714)

[2.2. Создание IJK сетки 6](#_Toc229892715)

[2.3. Загрузка свойств из файлов данных 6](#_Toc229892716)

[2.4. Сохранение свойств в файлы данных 7](#_Toc229892717)

[2.5. Удаление данных из памяти 7](#_Toc229892718)

[2.6. Расчет куба Vertical Proportion Curve (VPC) 8](#_Toc229892719)

[2.7. Функции работы с кубами свойств 9](#_Toc229892720)

[3. Использование алгоритмов 12](#_Toc229892721)

[3.1. Simple Kriging 12](#_Toc229892722)

[3.2. Ordinary Kriging 13](#_Toc229892723)

[3.3. Indicator Kriging 14](#_Toc229892724)

[3.4. LVM Kriging (Local Varying Mean) 16](#_Toc229892725)

[3.5. Sequential Indicator Simulation (SIS) (VPC, Corellogram) 17](#_Toc229892726)

[3.6. Sequential Gaussian Simulation (SGS, SGS LVM) 18](#_Toc229892727)

[4. Набор скриптов Python 21](#_Toc229892728)

[Авторы - Контакты 22](#_Toc229892729)

[Список изменений 23](#_Toc229892730)

[Лицензия 24](#_Toc229892731)

# 1. Основные сведения

## 1.1. Требования к системе

Для работы HPGL потребуется операционная система Windows (32 бита) или Linux (32/64 бита) c установленным Python версии не ниже 2.5 (скачать последнюю версию Python можно по адресу <http://www.python.org/download/>).

## 1.2. Описание

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

В версии **0.9.4** 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)
* Corellogram 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.

Поддерживается текстовый формат загрузки/выгрузки данных, в будущем планируется обеспечить совместимость в python numpy array.

## 1.3. Используемые компоненты

В библиотеке используются следующие свободно распространяемые компоненты:

- TNT (Template Numerical Toolkit) – библиотека для решения систем линейных уравнений (доступна по адресу http://math.nist.gov/tnt/overview.html)

## 1.4. Установка

### 1.4.1. Windows

Используйте инсталляционный файл HPGL-X.Y.Z.win32.exe.

В операционных системах Windows также потребуется установить Microsoft Visual C++ 2005 SP1 Redistributable Package (можно скачать по адресу <http://www.microsoft.com/downloads/details.aspx?familyid=200B2FD9-AE1A-4A14-984D-389C36F85647&displaylang=en>)

### 1.4.2. Linux

Ubuntu 8.10 (32-bit):

Установите пакет **hpgl\_x32\_X.Y.Z\_ubuntu\_8.10.deb.**

Ubuntu 8.10 (64-bit):

Установите пакет **hpgl\_x64\_X.Y.Z\_ubuntu\_8.10.deb.**

Все необходимые дополнительные пакеты (например библиотеки boost) будут установлены по зависимостям из репозитория.

К сожалению, пока нет сборок бинарных пакетов для других дистрибутивов, но если вы захотите собрать HPGL под другой дистрибутив, можете писать все вопросы в рассылку (см. Авторы – Контакты в конце документа). Инструкции по компиляции скоро будут на сайте проекта (http://hpgl.sourceforge.com)

# 2. Основные функции

## 2.1. Импорт библиотек

В начале каждого Python-скрипта, который будет использовать HPGL, необходимо прописать строки импорта библиотечных функций:

from geo import \*

## 2.2. Создание IJK сетки

Создание IJK сетки производится при помощи функции SugarboxGrid:

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

В результате выполнения команды будет создан объект сетки grid с размерностями I, J, K.

**Пример:**

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

## 2.3. Загрузка свойств из файлов данных

Поддерживаются текстовые файлы с данными следующего формата:

-- комментарий (игнорируется)

PROPERTY\_NAME

-- данные в порядке k,j,i (по слоям)

0

1

0

...

/

Файл формата данных должен содержать последовательно записанные значения из сетки, в порядке k,j,i (т.е. вертикальными слоями по k). Загрузка свойства из файла производится при помощи двух функций:

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

property\_object = load\_cont\_property(filename, undefined\_value)

property\_object = load\_ind\_property(filename, undefined\_value, [indicators])

В результате выполнения команды будет создан объект property\_object, в который будет загружено свойство из файла filename. Ячейки, значение в которых равно undefined\_value, будут считаться пустыми.

Последним параметров функции load\_ind\_property передаются коды индикаторов, находящихся в файле.

***Примечание:*** После загрузки индикаторного свойства из файла, все индикаторы преобразуются к виду 0,1,2,… в порядке, указанном в indicators.

**Пример:**

my\_cont\_property = load\_cont\_property("d:\CONTDATA.INC", -99)

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

## 2.4. Сохранение свойств в файлы данных

Сохранение свойства в файл производится при помощи функции write\_property:

write\_property(prop\_object, filename, prop\_name, undefined\_value, [indicators])

В результате выполнения команды будет создан файл filename, в который будет записано свойство из объекта prop\_object под именем prop\_name. На место пустых ячеек будет записано undefined\_value. Индикаторы будут записаны с индексами, указанными в indicators (если он не задан, то по порядку: 0, 1, 2…)

Тип свойства при записи (индикаторный/непрерывный) сохраняется автоматически.

**Пример:**

write\_property(my\_cont\_prop, "CON\_PROP.INC", "PROP\_CON", -99)

write\_property(my\_ind\_prop, "PR.INC", "PROP\_IND", -99, [0,1])

## 2.5. Удаление данных из памяти

Когда свойство уже не нужно, его можно удалить, освободив оперативную память. Делается это командой del:

del(prop\_object)

**Пример:**

del(my\_prop)

## 2.6. Расчет куба Vertical Proportion Curve (VPC)

Куб VPC – это куб, содержащий среднее значение свойства по каждому из вертикальных слоев. Он может быть использован в качестве источника средних значений в алгоритмах с изменяющимся средним (LVM SIS, LVM Corellogram SIS, LVM Kriging, и т.д.).

Расчет куба VPC по загруженному свойству prop выполняется следующим образом:

vpc\_cube = calc\_vpc(**prop**, **grid**, **marginal\_probs**)

где grid – это сетка, на которой выполняется расчет;

marginal\_probs – средние значения для каждого индикатора (это значение будет поставлено в те места, где по вертикали слой оказался пустым).

Нужно иметь в виду, что внутри куба vpc\_cube содержатся кубы средних для каждого из индикаторов, если индикаторов два — это будут соответственно vpc\_cube[0] и vpc\_cube[1].

Для сохранения куба VPC для любого из индикаторов, в файл, необходимо использовать функцию write\_mean\_data.

Куб, созданный при помощи calc\_vpc можно использовать для любого из алгоритмов с изменяющимся средним, передавая его в качестве параметра mean\_data.

**Пример:**

grid = SugarboxGrid(55, 52, 1)

prop = load\_ind\_property("IK\_HARD\_DATA.INC", -99, [0,1])

vpc\_cube = calc\_vpc( prop, grid, [0.8, 0.2] )

write\_mean\_data(vpc\_cube[0], "prob\_0.inc", "VPC\_PROB\_0");

write\_mean\_data(vpc\_cube[1], "prob\_1.inc", "VPC\_PROB\_1");

## 2.7. Функции работы с кубами свойств

Каждый куб свойств полученный в результате выполнения алгоритма, является объектом, и у него есть ряд методов.

1. **Создание пустого свойства заданного типа и размера**

Существует две возможность создать куб свойств – создать его с нуля (тогда он будет заполнен undefined-значениями) или осуществить копирование уже существующего.

Создание пустого свойства осуществляется функциями:

- для *непрерывного* свойства:

hpgl.cont\_property\_array(**size**)

- для *индикаторного* свойства:

hpgl.byte\_property\_array(**size**, [**indicators**])

Для копирования уже существующего свойства необходимо использовать функцию clone():

property\_copy = property.clone()

**Пример:**

prop\_cont = hpgl.cont\_property\_array(100\*100\*20)

prop\_ind = hpgl.byte\_property\_array(100\*100\*20, [0,1])

prop\_cont\_copy = prop\_cont.clone()

prop\_ind\_copy = prop\_ind.clone()

***Примечание:*** Если вам необходимо проверить, к какому типу (непрерывному или индикаторному) принадлежит какое-либо свойство, это можно сделать следующим образом:

if type(property) is hpgl.cont\_property\_array:

# действия, если property – непрерывное свойство

else:

# действия, если property – индикаторное свойство

***ОБРАТИТЕ ВНИМАНИЕ***, что конструкция вида:

property\_1 = property\_2

**не создает копию** свойства property\_2, а лишь делает property\_1 **указателем** на property\_2! Для создания полноценной копии нужно использовать функцию clone().

1. **Работа со значениями свойства через get\_at/set\_at**

Обращение к определенному значению свойства идет по индексу, от 0 до N-1, где N – это размер свойства. Индексирование идет в том порядке, в каком свойство было записано в файл данных, поэтому эти функции подходят в основном для изменения или получения значений в точках **независимо от их координат в сетке**. Плюс данного метода в том, что он очень быстрый и не занимает дополнительной памяти.

Все точки свойства делятся на два типа: известные (informed) и неизвестные (not informed) – т.е. те, в которых значение еще не смоделировано. Проверка на известность осуществляется функцией is\_informed():

result = property.is\_informed(index)

Если значение точки index известно, то resultбудет равен True, если нет, то False.

Для выполнения операций, в которых необходимо знать точные координаты изменяемых точек (a la работа с трехмерными массивами MATLAB), нужно воспользоваться функцией преобразования свойства в numpy-массив (описано ниже) – в этом случае с ним можно будет обращаться как с обыкновенным трехмерным массивом.

Для получения значения по индексу index из куба свойств property существует функция get\_at(). **Обратите внимание**, что перед использованием функции get\_at всегда необходимо проверять, является ли значение известным:

if(property.is\_informed(**index**) == True):

value = property.get\_at(**index**)

Для изменения значения по индексу index на value из куба свойств property существует функция set\_at():

property.set\_at(**index, value**)

После выполнения этой функции точка **index** будет считаться известной (informed).

**Пример:**

if(property.is\_informed(10) == True):

value = property.get\_at(10)

print ‘property value at 10 is’, value

else:

property.set\_at(10, 1000)

print ‘property value at 10 is 1000 now’

3) **Работа со значениями свойства через NumPy-массив**

Будет в следующем релизе.

# 3. Использование алгоритмов

## 3.1. Simple Kriging

Вызов алгоритма Simple Kriging 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

covariance\_type, # тип ковариации.

# может принимать значения

# covariance.spherical, covariance.exponential

# и covariance.gaussian

ranges, # радиусы вариограммы

sill, # пороговое значение вариограммы

nugget=None, # величина nugget-эффекта

angles=None, # углы поворота вариограммы

mean=None # среднее значение

# если не задано, вычисляется автоматически по

# изначальным данным

)

**Пример:**

grid = SugarboxGrid(55, 52, 1)

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

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

radiuses = (20, 20, 20),

max\_neighbours = 12,

covariance\_type = covariance.exponential,

ranges = (10, 10, 10),

sill = 1,

mean = 1.6)

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

del(prop\_result)

## 3.2. Ordinary Kriging

Вызов алгоритма Ordinary Kriging 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

covariance\_type, # тип ковариации.

# может принимать значения

# covariance.spherical, covariance.exponential

# и covariance.gaussian

ranges, # радиусы вариограммы

sill, # пороговове значение вариограммы

nugget=None, # величина nugget-эффекта

angles=None, # углы поворота вариограммы

)

**Пример:**

grid = SugarboxGrid(55, 52, 1)

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

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

radiuses = (20, 20, 20),

max\_neighbours = 12,

covariance\_type = covariance.exponential,

ranges = (10, 10, 10),

sill = 1)

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

del(prop\_result)

## 3.3. Indicator Kriging

Перед вызовом функции indicator\_kriging, выполняющей алгоритм, необходимо заполнить структуру, содержащую параметры моделирования.

Ниже приведен пример заполнения этой структуры параметров:

ik\_data = [

# Параметры вариограммы 1-го индикатора

{

"cov\_type": **cov\_type**, # тип вариограммы:

# 0 - сферическая

# 1 - экспоненциальная

# 2 - гауссовая

"ranges": (**R1**, **R2**, **R3**), # радиусы вариограммы

'sill': **sill**, # Sill — пороговое значение вариограммы

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

"max\_neighbours": **neigh\_count**, # максимальное количество соседей

# для интерполяции

"marginal\_prob": **marg\_prob**, # априорная вероятность индикатора

"value": **0** # значение индикатора

},

# Параметры вариограммы 2-го индикатора

{

"cov\_type": **cov\_type**, # тип вариограммы:

# 0 - сферическая

# 1 - экспоненциальная

# 2 - гауссовая

"ranges": (**R1**, **R2**, **R3**), # радиусы вариограммы

'sill': **sill**, # Sill — пороговое значение вариограммы

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

"max\_neighbours": **neigh\_count**, # максимальное количество соседей

# для интерполяции

"marginal\_prob": **marg\_prob**, # априорная вероятность индикатора

"value": **1** # значение индикатора

}

]

Для каждого существующего индикатора задается своя вариограмма.

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

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

def indicator\_kriging

(

ik\_prop, # список с параметрами алгоритма

grid, # сетка на которой производится кригинг

ik\_data, # свойства с исходными данными (hard data)

)

**Пример:**

grid = SugarboxGrid(55, 52, 1)

ik\_prop = load\_ind\_property("IK\_HARD\_DATA.INC", -99, [0,1])

ik\_data = [ {

"cov\_type": 0,

"ranges": (10, 10, 10),

'sill': 0.4,

"radiuses": (10, 10, 10),

"max\_neighbours": 12,

"marginal\_prob": 0.5,

"value": 0

},

{

"cov\_type": 0,

"ranges": (10, 10, 10),

"sill": 0.4,

"radiuses": (10, 10, 10),

"max\_neighbours": 12,

"marginal\_prob": 0.5,

"value": 1

}]

ik\_result = indicator\_kriging(ik\_prop, grid, ik\_data)

write\_property(ik\_result, "RESULT\_IK.INC", "PROP\_IK", -99)

## 3.4. LVM Kriging (Local Varying Mean)

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

def lvm\_kriging

(

lvm\_prop, # свойства с исходными данными (hard data)

grid, # сетка, на которой производится LVM Kriging

mean\_data, # куб, содержащий средние значения свойства для LVM

radiuses, # радиусы эллипсоида поиска

max\_neighbours, # максимальное число точек участвующее в интерполяции

covariance\_type, # тип ковариации.

# может принимать значения

# covariance.spherical, covariance.exponential

# и covariance.gaussian

ranges, # радиусы вариограммы

sill, # пороговове значение вариограммы

nugget=None, # величина nugget-эффекта

angles=None, # углы поворота вариограммы

)

Куб, используемый в качестве параметра mean\_data, загружается при помощи функции load\_mean\_data:

mean\_data\_obj = load\_mean\_data(filename)

где filename — путь к файлу со кубом средних значений;

mean\_data\_obj — куб средних значений, который будет использова в алгоритме.

**Пример:**

grid = SugarboxGrid(55, 52, 1)

mean\_data = load\_mean\_data("cube\_with\_local\_means.inc")

lvm\_prop = load\_ind\_property("LVM\_PROP.INC", [0,1])

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

radiuses = (20, 20, 20),

max\_neighbours = 12,

covariance\_type = covariance.exponential,

ranges = (10, 10, 10),

sill = 1)

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

del(mean\_data)

del(prop\_lvm)

## 3.5. Sequential Indicator Simulation (SIS) (VPC, Corellogram)

Параметры моделирования для алгоритма SIS задаются аналогично IK, заполнением той же структуры.

Вызов функции:

def sis\_simulation(

ik\_prop, # список с параметрами алгоритма

grid, # сетка на которой производится кригинг

ik\_data, # свойства с исходными данными (hard data)

seed, # Seed — номер стохастической реализации

mean\_data = None, # куб с вероятностями для LVM SIS

use\_corellogram = True,

# использование кореллограмного SIS или LVM SIS

# True — использовать кореллограмный SIS

# False — использовать LVM SIS

mask = False

**#** свойство, задающее регион моделирования (если необходимо

# смоделировать не все точки, а только часть).

# должно быть задано в виде индикаторного свойства со значениями

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

)

**Пример:**

grid = SugarboxGrid(55, 52, 1)

sis\_prop = load\_ind\_property("SIS\_HARD\_DATA.INC", -99, [0,1])

sis\_data = [ {

"cov\_type": 0,

"ranges": (10, 10, 10),

'sill': 0.4,

"radiuses": (10, 10, 10),

"max\_neighbours": 12,

"marginal\_prob": 0.5,

"value": 0

},

{

"cov\_type": 0,

"ranges": (10, 10, 10),

"sill": 0.4,

"radiuses": (10, 10, 10),

"max\_neighbours": 12,

"marginal\_prob": 0.5,

"value": 1

}]

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

write\_property(sis\_result, "RESULT\_SIS.INC", "PROP\_SIS", -99)

## 3.6. Sequential Gaussian Simulation (SGS, SGS LVM)

Вызов алгоритма SGS производится следующим образом.

Сначала нужно заполнить структуру с параметрами моделирования:

sgs\_params = {

"radiuses": (**SR1**, **SR2**, **SR3**),

# радиусы эллипсоида поиска (max, med, min)

"max\_neighbours": **max\_neigh**,

# максимальное количество соседних точек, используемых при

# интерполяции

"covariance\_type": **cov\_model**,

# тип вариограммы, может принимать значения:

# covariance.spherical — сферическая модель

# covariance.exponential — экспоненциальная модель

# covariance.gaussian — гауссовая модель

"ranges": (**R1**, **R2**, **R3**),

# радиусы вариограмм (max, med, min)

"sill": **sill**,

# пороговое значение вариограммы

"kriging\_type": **krig\_type**,

# тип кригинга, может принимать значения:

# “sk” - для Simple Kriging

# “ok” - для Ordinary Kriging

"mean": **mean**

**#** среднее значение свойства (при использовании Simple Kriging)

"mean\_data": **mean\_data\_obj**

**#** среднее значение свойства, заданное в виде объекта свойтсв

# (куба с данными)

"mask": **mask**

**#** свойство, задающее регион моделирования (если необходимо

# смоделировать не все точки, а только часть).

# должно быть задано в виде индикаторного свойства со значениями

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

}

После этого производится вызов алгоритма при помощи функции sgs\_simulation:

sgs\_result = sgs\_simulation(**property\_obj**, **grid\_obj**, **seed**, **mask**, \*\***sgs\_params**)

Параметры функции:

* property\_obj — свойство с изначальными данными (hard\_data);
* grid\_obj — сетка, на которой производится моделирование;
* seed — номер стохастической реализации;
* sgs\_params — заполненные параметры моделирования SGS
* mask - задает регион для моделирования (если нужно моделировать не все точки, а только часть). Задается в виде индикаторного куба, где на месте точек, подлежащих моделированию, стоит 1, а на месте не-моделируемых точек – 0. Если не задано, моделируются все неизвестные точки.

**Пример:**

grid = SugarboxGrid(55, 52, 1)

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

sgs\_params = {

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

"max\_neighbours": 12,

"covariance\_type": covariance.exponential,

"ranges": (10, 10, 10),

"sill": 0.4,

"kriging\_type": "sk",

"mean": 0.3}

sgs\_result = sgs\_simulation(prop\_con, grid, seed=12434, \*\*sgs\_params)

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

**Пример (LVM):**

grid = SugarboxGrid(55, 52, 1)

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

mean\_data = load\_mean\_data("SGS\_MEAN\_DATA.INC")

lvm\_sgs\_params = {

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

"max\_neighbours": 12,

"covariance\_type": covariance.exponential,

"ranges": (10, 10, 10),

"sill": 0.4,

"mean\_data": mean\_data}

sgs\_lvm = sgs\_simulation(prop, grid, seed=3439275, \*\*lvm\_sgs\_params)

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

del(sgs\_lvm)

# 

# 4. Набор скриптов Python

Набор скриптов на языке Python создан для того, чтобы показать примеры использования алгоритм и решения различных задач с использованием HPGL.

Для работы скриптов понадобится установить дополнение для Python, добавляющее работу с многомерными массивами — NumPy и SciPy (<http://www.scipy.org/Download>).

Для вывода на экран графиков (гистограмм и карт) необходимо также поставить модуль matplotlib (<http://sourceforge.net/projects/matplotlib>).

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 0.9.4** *- 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.