HYVR Documentation

Release 0.1

Jeremy Bennett

Jan 03, 2018

CONTENTS

Welcome to the Hydrogeological Virtual Reality (HyVR) simulation package.

Contents:

CHAPTER

ONE

INTRODUCTION

HyVR: Turn your geofantasy into reality!

The Hydrogeological Virtual Reality simulation package (HyVR) is a Python module that helps researchers and practitioners generate subsurface models with multiple scales of heterogeneity that are based on geological concepts. The simulation outputs can then be used to explore groundwater flow and solute transport behaviour. This is facilitated by HyVR outputs in common flow-and-transport simulation input formats. As each site is unique, HyVR has been designed that users can take the code and extend it to suit their particular simulation needs.

The original motivation for HyVR was the lack of tools for modelling sedimentary deposits that include bedding structure model outputs (i.e. dip and azimuth). Such bedding parameters were required to approximate full hydraulic-conductivity tensors for groundwater flow and solute transport modelling. HyVR is able to simulate these bedding parameters and generate spatially distributed parameter fields, including full hydraulic-conductivity tensors.

I hope you enjoy using HyVR much more than I enjoyed putting it together! I look forward to seeing what kind of funky fields you created in the course of your work.

1.1 Installing the HYVR package (Windows)

1.1.1 With conda

To install HyVR we recommend first installing the Anaconda distribution of Python 3. This distribution has the majority of dependencies that HyVR requires.

It is a good idea to install the HyVR package into a virtual environment. Do this by opening a command prompt window and typing the following:

```
python -m virtualenv hyvr_env
```

You need to then activate this environment:

```
call hyvr_env/scripts/activate
```

Install the necessary python packages by downloading the `requirements.txt` file in the HyVR repository and then running:

```
pip install -r <path to>requirements.txt
```

Once this is completed install HyVR using pip:

```
(hyvr_env) <working directory> pip install hyvr
```

You can test whether HyVR is working by running the default parameters in the Python console:

```
>>> import hyvr
>>> hyvr.sim.main(0)
```

A number of messages should then be displayed in the console, including the location where the simulation outputs have been saved.

1.2 Source

The most current version of HyVR will be available at the github repository; a version will also be available on the PyPI index which can be installed using `pip`.

1.3 Requirements

1.3.1 Python

HyVR was developed for use with Python 3.4 or greater. It may be possible to use with earlier versions of Python 3, howver this has not been tested.

1.3.2 Modules

- scipy
- pyevtk
- pandas
- numpy
- matplotlib
- flopy

1.4 Development

You can contact the developer(s) of HyVR by email. HyVR is currently being developed by Jeremy Bennett (website) as part of his doctoral research at the University of Tübingen.

CHAPTER

TWO

HYVR COMPUTATIONAL METHODS

The first step in the HyVR algorithm is to load the model parameters, as defined in the `*.ini` initialisation file. Sequence contact surfaces are generated first. Architectural element assemblage contact surfaces are then simulated within each sequence, either based on input parameters or loaded from a user-defined lookup table of mean contact surface depths. The external and internal geometries of architectural elements and associated hydrofacies are then simulated within each architectural element assemblage. Internal heterogeneity of the features is simulated at the end.

Note that in this section model input parameters are denoted in the following manner: `parameter-section.parameter`.

2.1 Simulation of sequence and architectural element unit contact surfaces

Sequences are defined in the parameter file by their upper mean elevations and the architectural elements that are to be included within them. The upper contact surface is then generated and all model cells between the lower and upper contact surface are assigned to the sequence.

Contact surfaces can either be flat or random. Multi-Gaussian random contact surfaces are generated using the spectral methods outlined by *[DN93]*. These methods require structural statistical parameters (i.e. mean and variance) for the quantity of interest, and a geostatistical covariance model. We used a Gaussian covariance model in the present study:

$$R_{ss}(h) = \sigma_s^2 \exp\left[-\left[\frac{\Delta x}{\lambda}\right]^2\right]$$

where s is the random quantity of interest, σ_s^2 is the variance of s, Δx is the distance between the two points, and λ is the correlation length.

Simulation at the architectural element hierarchical level begins once all sequence units have been assigned. Units in which AEs will be simulated are defined using an architectural element lookup table with the following information:

Architectural element identifier	Mean bottom elevation	Mean top elevation	Architectural element type	Sequence identifier

If the architectural element lookup table is not defined before initializing the simulation, then the architectural element units will be simulated based on input parameters defined for each sequence. This starts with the random choice of an architectural element from those defined; the probability of each architectural element being chosen is also defined in the model parameter file. The thickness of the architectural element unit is then drawn from a random normal distribution that is defined for each sequence in the parameter file. To account for the erosive nature of many sedimentary environments the algorithm may erode the underlying units: here the 'avulsion' thickness $th_{avulsion}$ is subtracted from the bottom and top of the architectural element unit $z_{AE}^{bot}, z_{AE}^{top}$. Once the architectural element lookup table has been defined, unit contact surfaces are generated using the same procedure as used for sequence contact surfaces. When the architectural element units have been generated, the algorithm begins to simulate external architectural element geometries and hydrofacies.

2.2 Simulation of architectural elements and hydrofacies geometries

The generation of architectural elements and internal hydrofacies occurs sequence- and architectural-element-wise, beginning with the lowest architectural element unit in the lowest sequence. The simulation of individual architectural elements is object-based, with random placement of features within the architectural element units. Object-based methods have been implemented widely in subsurface simulation *[JSD94][BHC17]* as they are generally computationally efficient and relatively easy to parameterize. The HyVR program approximates architectural elements with simple geometric shapes. Currently, three *shapes* are supported: truncated ellipsoids, channels, and sheets. Truncated ellipsoids and channels are 'erosive' architectural elements: this means that within the HyVR algorithm they are able to erode underlying units, and therefore the architectural element (and sequence) boundaries may be altered during the course of the simulation.

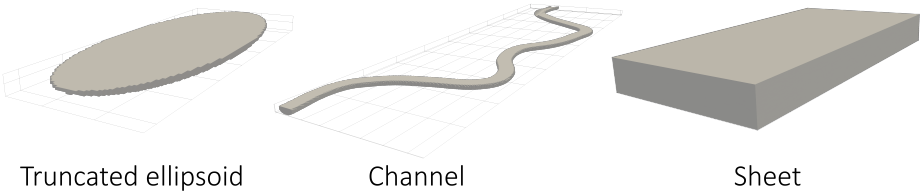

Fig. 2.1: *Geometries implemented in HyVR.*

Four properties are assigned to each model grid cell during this simulation step: material, facies, azimuth, and dip. The material property is a unique identifier for each individual architectural element generated. The facies property denotes which hydrofacies has been assigned to a model grid cell. The azimuth κ and dip ψ properties are associated with the bedding structure at each model grid cell and denote the angle of the bedding plane from the mean direction of flow and horizontal, respectively.

2.2.1 Truncated ellipsoids

Truncated ellipsoids are generated as a proxy for trough-like features. The method for generating the boundaries of these featu

- trough-wise homogeneous, with constant azimuth and dip;
- bulb-type, with azimuth and dip values based on the three-dimensional gradient at the ellipsoid boundary;
- nested-bulb-like, comprising nested alternating hydrofacies with κ and ψ values generated as for bulb-type;
- dip-set internal structure, where the features have a constant κ and ψ but the assigned hydrofacies alternate throughout the truncated ellipsoid.

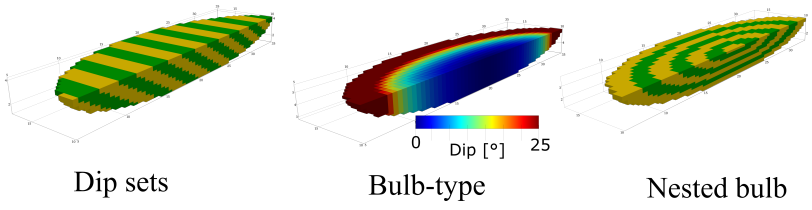

Fig. 2.2: *Internal structure of truncated ellipsoid architectural elements.*

Bulbs

Bulb internal heterogeneity is simulated by calculating the three-dimensional gradient at the boundary of the truncated ellipsoid and then the angle between the gradient and a horizontal plane. This angle is then compared with a 'maximum dip angle' (`r_dip`) and the smaller of these two values is assigned to all model grid cells within the architectural element with equivalent x, y-coordinates (i.e. column-wise).

Nested bulbs

Nested-bulb-like layers are simulated by subdividing the depth of the truncated ellipsoid into a series with a set thickness `trunc_ellip.bulbset_d`. Truncated ellipsoids are simulated consecutively with the same center point and paleoflow α value, starting with the deepest element. With each simulation, a scaling factor is calculated by dividing the new depth with the total depth of the element. This scaling factor is applied to the length and width parameters of the truncated ellipsoid. Each newly generated ellipsoid subsumes the previous. Each nested element represents a constant hydrofacies, however the orientation of these hydrofacies may differ within the entire architectural element, to create bulb-like features that have been reported in the field. The dip of the nested ellipsoids defaults to that determined by the three-dimension gradient at the nested-ellipsoid boundary.

Dipset

Refer to _Dipset section.

Once a truncated ellipsoid has been generated, an aggradation thickness (`trunc_ellip.agg`) is added to the current simulation elevation z_{sim} and the next element is simulated. This occurs until $z_{sim} = z_{AE}^{top}$.

2.2.2 Channels

Channels are sinuous features that are in-filled with sediment. Channel centerlines in HyVR are parameterized using the disturbed periodic model implemented by *[Fer76]*:

$\theta + \frac{2h}{k}\frac{d\theta}{ds} + \frac{1}{k^2}\frac{d^2\theta}{ds^2} = \epsilon(s)$

with channel direction θ, damping factor $h \in [0, 1]$, $k = 2\pi/\lambda$ is the wavenumber with λ the frequency of the undamped sine wave, and s is the distance along the channel. This model can be approximated using the following second-order autoregressive model described in Equation 15 of *[Fer76]* (this method was also used by [Pyrcz2009] for the simulation of alluvial depositional features). Model grid cells are assigned to the channel if the following conditions are met:

$D^2 \leqslant \frac{w_{ch}^2}{4} - \left[\frac{(z_{ch} - z_{cell}) \cdot \Delta z \cdot w_{ch}}{d_{ch}}\right]^2 \ \wedge \ z_{cell} \leqslant z_{ch}$

where D^2 is the two-dimensional (x, y) distance from the cell to the channel centerline, w_{ch} and d_{ch} are the channel width and depth respectively, z_{ch} and z_{cell} are the elevations of the channel top and node respectively, and Δz is the model grid cell z dimension. Two-dimensional channel velocities $\vec{v}$ are evaluated at the centerline and then interpolated to grid cells using an inverse-distance-weighted interpolation. Azimuth values are calculated by taking the arctangent of the two-dimensional channel velocity at a given point. Dip values of grid cells within the channel are assigned based on input parameters. If alternating hydrofacies are to be simulated they are constructed by creating planes that are evenly spaced along the channel centerline.

The HyVR algorithm generates channels starting from $z_{AEunit}^{bot} + AE_{depth} \cdot \beta$, as for truncated ellipsoids. However, to account for the multiple channels that are often concurrently present in many river systems, multiple channels can be generated at each simulation depth (`channel.channel_no`). The starting x, y coordinates for the centerlines are drawn from a random uniform distribution such that $x \in [-50, 0]$ and $y \in [0, Y]$. Channel geometries are then assigned sequentially to the model grid cells; note that in HyVR there is no interaction of channels, and subsequent channels will supersede (or 'erode') those previously generated. Once the predefined number of channels stipulated

by `channel.channel_no` has been simulated a three-dimensional migration vector `channel.r_mig` is added to the channel centerlines and the channel assignment to model grid cells begins again. The reuse of the channel centerline trajectories is more efficient than re-simulating these values at each z_{sim}. This continues until $z_{sim} = z_{seq}^{top}$.

2.2.3 Sheets

Sheets are comparatively simple to generate as they are laterally continuous across the entire model domain (depending on sequence boundaries). The internal structure of sheet features may be massive (i.e. without internal structure), or laminations can be generated. In the HyVR algorithm laminations are simulated sequentially by assigning all model grid cells between a specific elevation interval the appropriate hydrofacies codes. Dipping set structures can also be incorporated into these features. Sheets may differ in orientation, as specified in the input parameters.

2.2.4 Internal structure

The internal structure of the architectural elements is distinguished by hydrofacies. The internal structure of an architectural element may be homogeneous, dipping or elliptic (for truncated ellipsoid only). Additionally, lag surfaces composed of different hydrofacies may be simulated in erosive (i.e. channel, truncated ellipsoid) architectural elements.

Dipset

Architectural elements may be populated with dipping hydrofacies structures label{para:dipsets}. Such structures are generated by creating planes at regular intervals throughout the architectural element, as defined by `element.dipset_d`. In truncated ellipsoids the planes are constructed along the centerline of the element, perpendicular to the paleoflow angle α. In channel elements, the planes are constructed along the centerline and are perpendicular to $\vec{v}(x)$. The distance from the centre of each model grid cell to all planes is calculated and then the model grid cells between planes are assigned a hydrofacies value.

Lag surfaces

Lag surfaces can be set for erosive architectutral elements by setting the `element.l_lag` parameter. This parameter consists

- The thickness of the lag surface from the element base; and
- The hydrofacies identifier to be assigned.

Lag surfaces cannot have any internal dipping structure.

Sedimentary deposits can often exhibit cyclicity in their features; therefore, HyVR allows alternating hydrofacies to be simulated. This is controlled by sequentially assigning hydrofacies within each architectural element, starting with a hydrofacies randomly selected from those to be simulated in the architectural element (`element.l_facies`). The hydrofacies which is assigned next is drawn from a subset of hydrofacies specified in the `element.ll_altfacies` input parameter. For each hydrofacies in `element.l_facies`, a list of alternating hydrofacies (i.e. which hydrofacies can follow the present one) is stipulated. By only specifying one hydrofacies ID in the `element.ll_altfacies`, it guarantees that that ID will be selected. The figure below gives three examples of different input parameters.

2.2.5 Linear trends

The HyVR algorithm allows for linear trends in geometry sizes with increasing elevation by setting the `element.r_geo_ztrend` parameter. This parameter comprises a bottom and top factor that multiply the usual

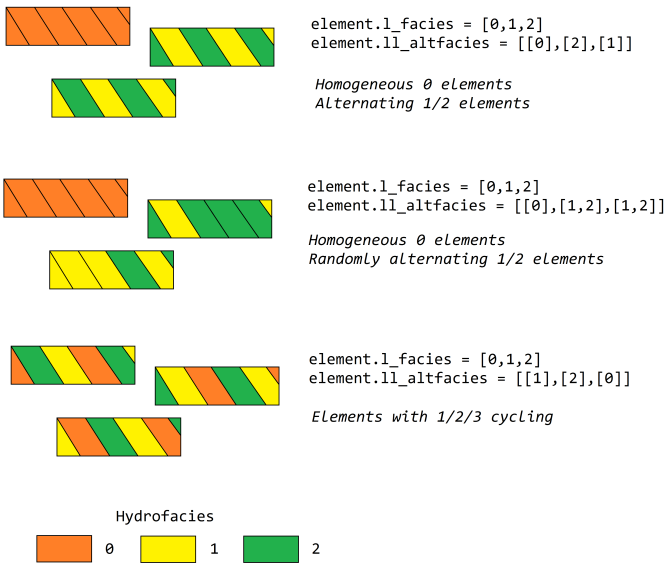

Fig. 2.3: *Variations on alternating hydrofacies in architectural elements*

geometry dimensions. For intermediate elevations the z factor is calculated through a linear interpolation of the top and bottom z factors. The parameters of each geometry may be set for each individual architectural element included in the model parameter file.

2.3 Simulation of hydraulic parameters

Hydraulic parameters are simulated once all features have been generated. The distributed hydraulic parameter outputs of HyVR are: the isotropic hydraulic conductivity $K_{iso}(x, y, z)$; porosity $\theta(x, y, z)$; and the full hydraulic conductivity tensor $\mathbf{K}(x, y, z)$, defined for each model grid cell.

Internal heterogeneity of hydraulic parameters is first simulated for each individual architectural element (as present in the `mat` storage array) simulated in the previous steps. Spatially varying $\ln(K_{iso})$ and θ fields are generated for each hydrofacies present in an architectural element using spectral methods to simulate random multi-Gaussian fields with an exponential covariance model:

$R_{ss}(h) = \sigma_s^2 \exp\left[-\left|\frac{\Delta x}{\lambda}\right|\right]$

An anisotropic ratio is also assigned to each model grid cell according to the hydrofacies present; these ratios are globally constant for each hydrofacies.

Internal heterogeneity may also be assigned to model grid cells that are not within architectural elements. This background heterogeneity is simulated for each architectural element unit using values defined for each architectural element type (`element.r_bg`). Simulation methods are the same as for within-element heterogeneity.

Once isotropic hydraulic-conductivity values have been assigned to all model grid cells then spatial trends may also be applied. As for trends in architectural element geometry, K_0 trends are assigned using a linearly-interpolated factor in the x- and/or z-direction. The K_0 value of each model grid cell is then multiplied by the trend factors.

Full hydraulic-conductivity tensors for each model grid cell are calculated by multiplying the isotropic hydraulic conductivity K^{iso}, with a a rotated anisotropy matrix $\mathbf{M}$:

$\mathbf{K}_i = K_i^{iso} \mathbf{R}_i \mathbf{M}_i \mathbf{R}_i^T$

$$\mathbf{R}_i = \begin{bmatrix} \cos(\kappa_i)\cos(\psi_i) & \sin(\kappa_i) & \cos(\kappa_i)\sin(\psi_i) \\ -\sin(\kappa_i)\cos(\psi_i) & \cos(\kappa_i) & -\sin(\kappa_i)\sin(\psi_i) \\ -\sin(\psi_i) & 0 & \cos(\psi_i) \end{bmatrix}$$

Parameters ψ_i and κ_i are the simulated bedding structures (dip and azimuth, respectively). The anisotropy matrix $\mathbf{M}_i$ is diagonal with lateral terms set as equivalent (i.e. $K_{xx} = K_{yy}$). This approach is identical to that of *[BHC17]*. Once this has been completed, the simulated parameter files are saved and can be used for groundwater flow and solute transport simulations.

CHAPTER

THREE

HYVR INPUTS AND OUTPUTS

3.1 Parameter inputs and model outputs

Model input parameters are contained in `*.ini` files which can be easily customized using your favourite text editor. This format was chosen for its readability and the ability to be used with numerous programming languages.

HYVR simulations are structured in the following way:

Model -> Run -> Realisation

The default operation of HYVR is to save simulation outputs in the same directory as the `.ini` parameter input file. However, it is possible to save the simulation outputs into another directory by specifying "run.modeldir" in the parameter file.

3.2 Model outputs

3.2.1 MODFLOW input files

CHAPTER

FOUR

HYVR EXAMPLE

In this section we start to use HYVR. The model parameter input files are explained.

4.1 The `.ini` configuration file

The key piece of information requried by HYVR is the parameter file. This is an `.ini` configuration file that contains all the parameters required to run a HYVR simulation.

The parameter file is separated into sections denoted by headers surrounded by brackets (e.g. `[section_name]`). Parameters (or keys) and their associated values are then stipulated using the equals sign (e.g. `key = value`). Each key is associated with the section in which it is located. Section and variable names should be in lower case. String values do nore require quotation marks in `.ini` files.

In HYVR there following sections are necessary:

- `[run]` - This contains parameters related to the model run.
- `[model]` - This contains details about the model dimensions
- `[sequences]` - In this section the sequence parameters are set.
- `[*architectural_elements*]` - Each architectural element to be simulated in HYVR is included in its own section. Please see the subsection below for more information.
- `[hydraulics]` - This section includes information for setting the hydraulic properties of simulated features.

An additional section `[flowtrans]` is included in some parameter files - this section included parameters used in groundwater flow and solute transport simulation packages not implemented in HYVR. `.ini` files are readable by a range of programming languages so the user can also store and read flow and transport parameters from the configuration file.

A number of key prefixes are used in the configuration files to assist identification when the HYVR package reads the configuration file:

- `r_` - This prefix denotes the lower and upper bounds of a range of float values.
- `l_` - This denotes a list of string values (no punctuation marks are required).
- `ll_` - This prefix denote a list of lists, each enclosed in brackets.
- `flag_` - keys with this prefix will be intepreted as true/false (boolean) values.

4.2 Model setup sections

4.2.1 `[run]` section

- `runname` - Name of the model simulation run
- `numsim` - Number of realisations to be generated
- `l_dataoutputs` - Simulation output data formats [vtk, mat, py]
- `l_modeloutputs` - Simulation output model formats [mf, hgs]
- `fp` - filepath/directory for simulation outputs
- `flag_ow` - overwrite previous simulation with same name?
- `flag_anisotropy` - Generate anisotropy parameters?
- `flag_het` - Generate heterogeneity?

4.2.2 `[model]` section

- `dx`, `dy`, `dz` - Model grid cell dimensions
- `lx`, `ly`, `lz` - Model domain dimensions
- `flag_periodic` - Periodic model domain? (Sheets/truncated ellipsoids only)
- `flag_display` - 'Display'-type simulation? If this flag is set to `true`, the simulated architectural elements are centred in the model domain so they can be viewed easily.
- `hetlev` - Hierarchical level at which heterogeneity should be simulated (ae, facies, internal)

4.3 `[sequences]` section

- `l_seq` - List of sequences
- `r_seq_top` - List of mean sequence contact elevations
- `ll_seq_contact_model` - Statistical parameters for sequence contact model
- `ae_table` - Filepath for architectural element lookup table
- `seq_contact` - Contact surface type (flat, random, user)
- `ll_seq_ae` - Which architectural elements are in each sequence
- `ll_ae_prob` - Probability of an architectural element occuring
- `ll_ae_z_mean` - Mean thickness of architectural element unit
- `ll_avul_prob` - Probability of avulsion
- `ll_avul` - Avulsion depth range
- `r_bg` - Background values for unassigned cells

4.4 [element] sections for architectural elements

Sections that describe architectural elements are entitled with an identifying name (e.g. [sparse_scour]). Note that section names should not include spaces. The first parameter to be set it the geometry. The current implementation of HYVR includes three geometries: truncated ellipsoids (trunc_ellip), channels (channel), and sheets (sheet).

4.4.1 General [*element] parameters

- geometry - Geometry of architectural element (trunc_ellip, channel, sheet)
- structure - Internal structure of architectural element

4.4.2 Truncated ellipsoid parameters

4.4.3 Channel parameters

4.4.4 Sheet parameters

4.5 [hydraulics] section

The input parameters in this section are associated with the simulation of hydraulic parameters. It is also possible to only simulate the geometries of architectural elements and hydrofacies if required.

- flag_gen - Generate hydraulic parameters (i.e. hydraulic conductivity)?
- l_hydro - List of hydrofacies codes
- r_k_h - Mean horizontal hydraulic conductivity
- r_sig_y - Variance of log hydraulic conductivity
- ll_ycorlengths - Default correlation lengths for $\log(K_{iso})$ in each hydrofacies in x, y, z-directions
- r_k_ratio - List of perpendicular anisotropy ratios (i.e $\frac{K_h}{K_v}$)
- r_n - List of mean porosity values
- r_sig_n - Variance of porosity values
- ll_ncorlengths - Default correlation lengths for porosity in each hydrofacies in x, y, z-directions

4.6 [flowtrans] section

This section contains parameters to be used for groundwater flow and solute transport simulations. This allows all input parameters for field generation and subsequent modelling to be stored in the same .ini file.

4.7 Example parameter file

An example of a parameter file is included below.

```
## Example HyVR model parameter input file
# HyVR 0.1 simulation package
# https://github.com/driftingtides/hyvr/
# Jeremy P. Bennett, University of Tübingen, 2017-2018

# Notes:
#        - String values do not require quotation marks
#         - All section and variable names should be lower case

# Suffixes
# r_:                 range of float values
# l_:                 list of string values
# ll_:                list of tuples
# flag_:         flag value (boolean)

[run]
# ------------------
# Run parameters
# ------------------

# Name of model simulation run
runname = testing_small1

# Number of realisations
numsim = 1

## Outputs
# Required outputs
# vtk: *.vtk
# py: python pickle
# mat *.mat
l_dataoutputs = [vtk,mat,py]
#l_modeloutputs = [mf,hgs]

# Full Filepath/directory for outputs
# Default is the directory of the parameter initialization file
# fp = made

# Overwrite parameter files
flag_ow = true

# Will anisotropy be assigned?
flag_anisotropy = true
flag_het = true


[model]
# ------------------
# Model parameters
# ------------------

# Grid cell dimensions [m]
dx = 0.5
dy = 0.5
dz = 0.1

# Model dimensions [m]
#lx = 200
```

```
lx = 40
#ly = 70
ly = 20
lz = 11

# Is domain periodic?
flag_periodic = false
flag_display =  false

# Lowest hierarchical level of heterogeneity to assign
#        -ae
#        -facies
#         -internal
hetlev = internal

[sequences]
# ------------------
# Sequence parameters
# ------------------

## List of sequences
l_seq = [clay, transition, glaflu, meander]

## List of sequence top contact depths
r_seq_top = [1.5, 3, 8, 11]

#  [variance, correlation length x, corr. length. y]
ll_seq_contact_model = [[0.05,6,6],[0.05,6,6],[0.05,6,6],[0.05,6,6]]

## Architectural element lookup table
#ae_table = ae_lu_19-09-2017_10.39.18.txt

## Contact surfaces
# flat:                 horizontal contacts <default>
# random: random surfaces
#                                 - requires geostatistical model "l_contact_model"
# user:         Use-defined contact surfaces
#                                 - requires input path "contact_file"
seq_contact = random

# Which architectural elements are included in each sequence
# - Must have same length as l_seq,
# - Architectural elements must be identical to section names (except␣
↪[model],[hydraulics])
ll_seq_ae = [[clay_sheet],[sand_sheet,clay_lens],[crossbedded_scour, sandy_
↪gravel],[mc_sheet, meander_channel]]

# The probability of an architecutral element occuring
ll_ae_prob = [[1.0],[0.4,0.6],[0.8,0.2],[0.3, 0.7]]

# Mean thickness of architectural element
ll_ae_z_mean = [[3.0],[0.3,0.3],[1.7, 0.2],[1.0, 2.0]]

## Erosion / deposition rules
# Avulsion probability
ll_avul_prob = [[0],[0],[0.7],[0]]

# Avulsion depth range [m]
```

```
ll_avul = [[0.0,0.0],[0.0,0.0],[0.2, 0.4],[0.0,0.0]]

# Background parameters for unassigned cells
# [fac, azim, dip]
r_bg = [5, 4, 0, 6]


[crossbedded_scour]
# ------------------
# Scour pool element
# ------------------
geometry = trunc_ellip

# Internal structure
structure = random
agg = 0.2

# Contact type
contact = random
#  [variance, correlation length x, corr. length. y]
r_contact_model = [0.01,6,6]

# Number of elements per simulation elevation
el_z = 3e-3

# Migration of troughs [mean & var migration in x, y]
#r_migrate = [10, 1, 10, 1]

# Do not generate troughs close to bottom contact
# Value is proportion of trough depth
buffer = 0.8

# Mean trough geometry [m]
length = 22
width = 10.4
depth = 1.2

# Mean angles [°]
r_paleoflow = [-45, 45]
r_dip = [10, 25]
r_azimuth = [-45, 45]

# Hydrofacies (refer to [hydraulics]l_hydro; 0-indexed)
l_facies = [1,2,3]

# Alternating facies
# List of what hydrofacies can follow those listed in l_facies
# To generate cyclical facies each list entry should have only one facies value
ll_altfacies = [[1,2],[1,2],[3]]

# Thickness of lenses (or) spatial period (lambda) of inclined set [m]
bulbset_d = 0.1
dipset_d = 0.7

# Background parameters for unassigned cells
# [fac, azim, dip]
r_bg = [0, 0, 0]
```

```
# Geometry trend with elevation
# Trends are linear, moving from bottom to top of domain
# Percentage change of mean value with dx = 1m
r_geo_ztrend = [2, 0.5]

[meander_channel]
# ------------------
# Meander channel element
# ------------------
geometry = channel
width = 10
depth = 1.5

# Internal structure
structure = massive
agg = 5

# Contact type
contact = random
#  [variance, correlation length x, corr. length. y]
r_contact_model = [0.001,12,6]

#  Migration vector (x,y,z-difference between two channels per dz)
r_mig = [2, 20, 0.5]

# Channel shape parameters
h = 0.4
# Wavenumber
k = 0.3
# Channel distance for calculations
ds = 1
eps_factor = 0.1

# Channels per iteration
channel_no = 1

# Dip range ([0,0] = massive bedding without any dip)
r_dip = [0, 0]

# Do not generate troughs close to bottom contact
# Value is proportion of trough depth
buffer = 0.4

# Hydrofacies (refer to [hydraulics]l_hydro; 0-indexed)
l_facies = [5]

# Lag surface at bottom of feature
# [lag depth, hydrofacies]
l_lag = [0.3, 0]

# Background parameters for unassigned cells
# [mat, fac, azim, dip]
r_bg = [6, 0, 0]

# Global hydraulics trend with elevation
# Trends are linear, moving from bottom to top of domain
r_k_ztrend = [1.5, 0.9]
```

```
[sandy_gravel]
# ------------------
# Sandy gravel sheet element
# ------------------
# Geometry
geometry = sheet
lens_thickness = -1
structure = massive

# Contact type
contact = random
#  [variance, correlation length x, corr. length. y]
r_contact_model = [0.05,6,6]

# Hydrofacies (refer to [hydraulics]l_hydro)
l_facies = [0]

[sand_sheet]
# ------------------
# Sand sheet element
# ------------------
# Geometry
geometry = sheet
lens_thickness = 0.3
structure = massive

# Contact type
contact = flat

#  [variance, correlation length x, corr. length. y]
r_contact_model = [0.01,6,6]

# Dip range ([0,0] = massive bedding without any dip)
r_dip = [0, 0]

# Spatial period (lambda) of inclined set [m]
setlamb = 0.3

# Hydrofacies (refer to [hydraulics]l_hydro)
l_facies = [4]

# Global hydraulics trend with elevation
# Trends are linear, moving from bottom to top of domain
r_k_ztrend = [0.5, 5]

[clay_sheet]
# ------------------
# Clay sheet element
# ------------------
# Geometry
geometry = sheet
lens_thickness = 0.2
structure = massive

# Contact type
contact = flat
#  [variance, correlation length x, corr. length. y]
r_contact_model = [0.01,6,6]
```

```
# Dip range ([0,0] = massive bedding without any dip)
r_dip = [0, 0]

# Spatial period (lambda) of inclined set [m]
setlamb = 3

# Hydrofacies (refer to [hydraulics]l_hydro)
l_facies = [5]

# Global hydraulics trend with elevation
# Trends are linear, moving from bottom to top of domain
r_k_ztrend = [0.1, 10]

[mc_sheet]
# ------------------
# silt/clay sheet element
# ------------------
# Geometry
geometry = sheet
lens_thickness = 0.1
structure = massive

# Contact type
contact = flat
r_contact_model = [0.01,6,6]

# Spatial period (lambda) of inclined set [m]
setlamb = 3

# Dip range ([0,0] = massive bedding without any dip)
r_dip = [0, 0]

# Hydrofacies (refer to [hydraulics]l_hydro)
l_facies = [6]

# Global hydraulics trend with elevation
# Trends are linear, moving from bottom to top of domain
r_k_ztrend = [0.5, 10]

[clay_lens]
# ------------------
# Clay/silt lens
# ------------------
geometry = trunc_ellip

# Internal structure
structure = flat
agg = 0.2

# Contact type
contact = flat
#  [variance, correlation length x, corr. length. y]
r_contact_model = [0.01,6,6]

# Number of elements per simulation elevation
el_z = 1e-3
```

```
# Migration of troughs [mean & var migration in x, y]
#r_migrate = [20, 1, 10, 1]

# Do not generate troughs close to bottom contact
# Value is proportion of trough depth
# buffer = 0.2

# Mean trough geometry [m]
length = 10
width = 8
depth = 0.3

# Mean angles [°]
r_paleoflow = [-90, 90]
r_dip = [0, 0]
r_azimuth = [0, 0]

# Hydrofacies (refer to [hydraulics]l_hydro; 0-indexed)
l_facies = [6]

# Alternating facies
# List of what hydrofacies can follow those listed in l_facies
# To generate cyclical facies each list entry should have only one facies value
ll_altfacies = [[6],[6]]

# Thickness of lenses (or) spatial period (lambda) of inclined set [m]
setlamb = 0.2

# Background parameters for unassigned cells
# [mat, fac, azim, dip]
r_bg = [4, 0, 0]

# Geometry trend with elevation
# Trends are linear, moving from bottom to top of domain
# Percentage change of mean value with dx = 1m
r_geo_ztrend = [1, 1]

[hydraulics]
# ------------------
# Hydraulic parameters
# ------------------
# Simulation of hydraulic parameters?
flag_gen = true

# List of hydrofacies codes
l_hydro = [sG, scG, oG, S, fS, C, mS]

# mean horizontal hydraulic conductivity [m/s]
r_k_h = [1e-5, 1e-7, 1e-1, 1e-4, 1e-5, 2e-9, 3e-9]

# variance of log hydraulic conductivity [-]
r_sig_y = [1, 1, 1, 1, 1, 1, 1]

# default correlation lengths for log(K) in each hydrofacies in x,y,z-directions
ll_ycorlengths = [[13,13,1.6],[13,13,1.6],[13,13,1.6], [13,13,1.6],[13,13,1.
↪6],[13,13,1.6],[13,13,1.6]]

# List of perpendicular anisotropy ratios (i.e K_h/K_v) [-]
```

```
r_k_ratio =  [1, 0.25, 0.025, 1, 2.3, 2.3, 2.3, 1.7]

# list of mean porosity values [-]
r_n = [0.2, 0.17, 0.35, 0.43, 0.43, 0.52, 0.45]

# variance of porosity values [-]
r_sig_n = [0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005,0.0005]

# default correlation lengths for n in each hydrofacies in x,y,z-directions
ll_ncorlengths = [[3,3,0.3],[3,3,0.3],[3,3,0.3],[3,3,0.3],[3,3,0.3],[3,3,0.3],[3,3,0.
↪3]]

# Global hydraulics trend with elevation
# Trends are linear, moving from bottom to top of domain
#r_k_ztrend = [1.5, 0.9]
#r_k_xtrend = [1.5, 0.9]


[flowtrans]
# -------------------
# Flow and transport modelling parameters
# -------------------

# Boundary conditions (head in/out [m])
hin = [1, 0, 0]
hout = [0, 0, 0]
```

CHAPTER

FIVE

EXTENDING HYVR

The HYVR package is a work in progress. It has been implemented in Python in order to make it accessible for researchers, easily customisable, and hopefully somewhat easy to extend to suit the needs of future research. In this section I have included some tips on ways to extend the HYVR source code.

5.1 Adding more geometries

HYVR has been set up in such a way to facilitate the implementation of additional architectural element geometries.

In order to generate new types of geometries a new function needs to be written in the `hyvr` module that will be called from `hyvr.hyvr_main()` where individual architectural elements and hydrofacies are simulated (around line 188 of `hyvr.hyvr_main()`).

Any new geometry function needs to return the following properties:

- `mat` (numpy array)
- `azim` (numpy array)
- `dip` (numpy array)
- `fac` (numpy array)
- `ae_arr_i` (numpy array)

5.2 The HYVR wish list

Any modelling project will have 'areas for growth' (as opposed to weaknesses). I have identified some things that I would like HYVR to have, but that are outside of the scope of my PhD research (and funds...). Perhaps you have the time and need to complete these yourself?

- Extensions in `C` programming language to speed up bottlenecks in simulations, particularly in `hyvr.scale_rotate'',``hyvr.reindex`, and `hyvr.planepoint`.
- Some level of conditioning, or improved interfacing with multiple-point geostatistical packages.
- Interaction of channels, as well as more complex/realisitic configurations of channel deposits (e.g. point bars).
- Utilities for deriving HYVR simulation parameters from transitional probability geostatistics.
- Simulation of chemofacies.

CHAPTER

SIX

REFERENCES

CHAPTER

SEVEN

HYVR

7.1 hyvr package

7.1.1 Submodules

7.1.2 hyvr.hyvr.grid module

Grid class

A module containing some classes and function useful to work with structured grids.

Usage Import as a normal python module.

Version 0.1 , 01-09-2016 : Forked from Alessandro Comunian

Authors Jeremy P. Bennett

Notes: The grids for the moment are always considered as 3D.

class hyvr.hyvr.grid.**Grid**(*ox=0.0*, *oy=0.0*, *oz=0.0*, *dx=1.0*, *dy=1.0*, *dz=1.0*, *nx=200*, *ny=200*, *nz=10*, *gtype='points'*, *gname='image'*, *periodicity=False*)

Bases: object

Grid class

A simple class that contains the *Origin*, *Delta* and *Size* of a simulation. It can also be used as container for some information contained in a VTK structured grid file.

Notes

•By default, the size of the grid is considered as points.

See also:

vtknumpy

cart_coords()
Get x,y,z coordinates in a tuple

Parameters:

Returns A Tuple containing the x,y and z-coordinates of the grid

cart_coords2d()
Get x,y coordinates in a tuple

Parameters **self** – An instance of the Grid class

Returns A tuple containing the x,y-coordinates of the grid

cells
Number of cells

cellsize_2d()
Compute the cell size in 2D

Parameters **self** – An instance of the Grid class

Returns Cell size in 2D

compute_max()
Compute the max values for *x*, *y* and *z* of the grid.

Parameters **self** – An instance of the Grid class

Returns Max values of x,y and z

cs2
Some helpful things for getting grid indices

del_cells()
Delete the cells for a cells grid

Parameters **self** – An instance of the Grid class

Returns:

del_lx()
Delete the x-dimension of a grid

Parameters **self** – An instance of the Grid class

Returns Tuple containing the size of a grid in x-direction

del_ly()
Delete the y-dimension of a grid

Parameters **self** – An instance of the Grid class

Returns Tuple containing the size of a grid in y-direction

del_lz()
Delete the z-dimension of a grid

Parameters **self** – An instance of the Grid class

Returns Tuple containing the size of a grid in z-direction

del_points()
Delete the points of a points grid

Parameters **self** – An instance of the Grid class

Returns:

get_cells()
Update the number of cells for a cells grid

Parameters **self** – An instance of the Grid class

Returns Number of cells for x, y and z

get_lx()
Provide as output a tuple containing the x-size of a grid. Useful for the implementation of 'property'.

Parameters **self** – An instance of the Grid class

Returns Tuple containing the size of a grid in x-direction

get_ly()
Provide as output a tuple containing the y-size of a grid. Useful for the implementation of 'property'.

Parameters **self** – An instance of the Grid class

Returns Tuple containing the size of a grid in y-direction

get_lz()
Provide as output a tuple containing the z-size of a grid. Useful for the implementation of 'property'.

Parameters **self** – An instance of the Grid class

Returns Tuple containing the size of a grid in z-direction

get_points()
Update the number of points for a points grid

Parameters **self** – An instance of the Grid class

Returns Number of points for a points grid

idx_z(*zval*)
Return the index of an elevation for the k-dimension of a 3D array

Parameter: zval: Elevation value

Returns **iz** – Index of elevation

Return type int

lx
'size' along *x* of the grid.

ly
'size' along *y* of the grid.

lz
'size' along *z* of the grid.

meshup(*ind='ij'*)
Create a meshgrid representation of the grid

Parameters **self** – An instance of the Grid class

Returns A tuple containing the x,y,z-coordinates of the grid

meshup2d(*ind='ij'*)
Create a 2D meshgrid representation of the grid

Parameters **self** – An instance of the Grid class

Returns A tuple containing the x,y-coordinates of the grid

origin()
To print out the origin of the grid as a tuple

Parameters **self** – An instance of the Grid class

Returns A tuple containing the origin defined in the grid

points
Number of points

print_intervals(*axis='xyz'*)
Print the intervals that constitute the simulation domain in a format like:

[ox, ox+nx*dx] [oy, oy+ny*dy] [oz, oz+nz*dz]

where *ox* is the origin, *nx* is the number of points and *dx* is the delta between points (*idem* for *y* and *z*).

Parameters **axis** (*string*) – containing ['x','y','z'], [optional] If the default value "xyz" is used, then all the intevals are printed.

Returns print intervals that constitute the simulation domain

set_cells(*val=None*)
Set the number of cells for a cells grid

Parameters **self** – An instance of the Grid class

Returns Number of cells for x, y and z

set_lx(*val=None*)
Set the x-size of a grid

Parameters **self** – An instance of the Grid class

Returns Tuple containing the size of a grid in x-direction

set_ly()
Set the y-size of a grid

Parameters **self** – An instance of the Grid class

Returns Tuple containing the size of a grid in y-direction

set_lz()
Set the z-size of a grid

Parameters **self** – An instance of the Grid class

Returns Tuple containing the size of a grid in z-direction

set_points(*val=None*)
Set the number of points for a points grid

Parameters **self** – An instance of the Grid class

Returns Number of points for a points grid

shape()
To print out the shape of the grid as a tuple

Parameters

- **self** – An instance of the Grid class
- **gtype** (*string*) – 'points' or 'cells'. String to decide to print the shape in terms of points or in terms of cells.

Returns A tuple containing the shape defined in the grid

spacing()
To print out the spacing of the grid as a tuple

Parameters **self** – An instance of the Grid class

Returns A tuple containing the spacing defined in the grid

vec()
Create vectors of spatial coordinates

Parameters **self** – An instance of the Grid class

Returns Vectors of spatial coordinates

Return type xv, yv, zv

vec_node()
Create vectors of spatial coordinates of bounding nodes

Parameters **self** – An instance of the Grid class

Returns Vectors of spatial coordinates of bounding nodes

Return type xv, yv, zv

vec_x()
Create vector of spatial x-coordinate

Parameters **self** – An instance of the Grid class

Returns Vector of spatial x-coordinate

Return type xv

vec_y()
Create vector of spatial y-coordinate

Parameters **self** – An instance of the Grid class

Returns Vector of spatial y-coordinate

Return type yv

vec_z()
Create vector of spatial z-coordinate

Parameters **self** – An instance of the Grid class

Returns Vector of spatial z-coordinate

Return type zv

7.1.3 hyvr.hyvr.sim module

Hydrogeological Virtual Reality simulation package.

Hydrogeological virtual reality (HYVR) simulator for object-based modelling of sedimentary structures

Notes

Grid nodes are cell-centred!

hyvr.hyvr.sim.**angle**(*v1*, *v2*)
Return angle between two vectors in [°]

Parameters

- **v1** – Vector 1
- **v2** – Vector 2

Returns Angle between v1 and v2

hyvr.hyvr.sim.**channel_checker**(*param_file*, *ae_name*, *no_channels=1*, *dist=0*)
channel_checker function for quickly assessing the shape of channel inputs

Parameters

- **param_file** (*str*) – Parameter file location
- **ae_name** (*str*) – Name of architectural element
- **no_channels** (*float*) – Number of channels
- **dist** (*float*) – Distance to generate channels - defaults to mg.lx

Returns Plots showing shape of Ferguson channels

hyvr.hyvr.sim.**curve_interp**(*xc*, *yc*, *spacing*)

Interpolate evenly spaced points along a curve. This code is based on code in an answer posted by 'Unutbu' on http://stackoverflow.com/questions/19117660/how-to-generate-equispaced-interpolating-values (retrieved 17/04/2017)

Parameters

- **xc** – x coordinates of curve
- **yc** – y coordinates of curve
- **spacing** – Spacing between points

Returns x coordinates of interpolated points yn: y coordinates of interpolated points

Return type xn

hyvr.hyvr.sim.**dip_rotate**(*azimuth_in*, *dip_in*)

Rotate dip angle based on azimuth Note that inputs and outputs are in degrees

Parameters

- **azimuth_in** – Azimuth input angle
- **dip_in** – Dipping input angle

Returns Azimuth output angle

Return type dip_out

hyvr.hyvr.sim.**dip_sets**(*mg*, *aep*, *znow*, *channel=[]*, *select=[]*, *azimuth_z=0*)

Generate dip angles and assign to the dip matrix

Parameters

- **mg** – Mesh grid object class
- **aep** – Architectural element parameters (dict)
- **channel** – Tuple of x,y coordinates of channel (omitted for linear flows) - x, y coordinates of channel - vx, vy of channel flow
- **select** – Model grid nodes to assign

Returns Array of assigned dip values fac_out: Array of assigned hydrofacies

Return type dip_out

hyvr.hyvr.sim.**ellipsoid_gradient**(*x*, *y*, *z*, *a*, *b*, *c*, *alpha*, *select*, *tr*)

Calculate dip and strike values in rotated ellipsoids

Parameters

- **y, z** (*x,*) – Distances to centre of ellipsoid
- **b, c** (*a,*) – Majox/minor axes of ellipsoid

- **alpha** – Rotation of ellipsoid from mean flow direction

Returns Dipping in 3D azimuth_g: Azimuth in 3D

Return type dip_g

hyvr.hyvr.sim.**facies**(*run*, *model*, *sequences*, *hydraulics*, *flowtrans*, *elements*, *mg*)
Generate hydrofacies fields

Parameters

- **run** – Model run parameters like `runname`, `rundir`, `l_dataoutputs`, `l_modeloutputs`, etc.
- **model** – Model domain parameters
- **sequences** – Details about the sequences
- **hydraulics** – Details about the hydraulics
- **flowtrans** – Flow & transport simulation parameters
- **elements** – Architectural elements and parametersX
- **mg** – Mesh grid object class

Returns

probs –

Contains data of architectural element units and associated hydrofacies if model parameter 'flag_anisotropy':

azim = Azimuth angles mat = Material values dip = Dipping angles fac = Facies values ae_arr = Array with architectural element unit details seq_arr = Array with sequence details

else

mat = Material values fac = Facies values

params (list): Contains parameters of model domain, sequence, hydraulics, etc.

run (dict) = Model run parameters model (dict) = Model domain parameters sequences (dict) = Sequence parameters hydraulics (dict) = Hydraulic properties parameters flowtrans (dict) = Flow & transport simulation parameters elements (dict) = Architectural elements and parameters mg = Mesh grid object class ae_lu = Architectural element lookup table

Return type list

hyvr.hyvr.sim.**ferguson_channel**(*mg*, *h*, *k*, *ds*, *eps_factor*, *dist=0*, *disp=False*)
Simulate channel centrelines using the Ferguson (1976) disturbed meander model Implementation of AR2 autoregressive model http://onlinelibrary.wiley.com/doi/10.1002/esp.3290010403/full

Parameters

- **mg** (*object class*) – Mesh grid object class
- **h** (*float*) – Height
- **k** (*float*) – Wave number
- **ds** (*float*) – Channel distance for calculations
- **eps_factor** (*float*) – Random background noise
- **dist** (*float*) – Distance to generate channels - defaults to mg.lx

- **disp** (*bool*) – Creating display channel - channel begins at (0,0)

Returns **outputs** – Simulated channel centerlines: storage array containing values for x coordinate, y coordinate, vx and vy

Return type float array

hyvr.hyvr.sim.**ferguson_theta**(*s*, *eps_factor*, *k*, *h*)

Calculate channel direction angle

Parameters

- **s** – Steps within generated channel distance
- **eps_factor** – Random background noise
- **k** – Wave number
- **h** – Height

Returns **th_store** – Channel direction angle

Return type array

hyvr.hyvr.sim.**gen_channel**(*ch_par*, *mg*, *model*, *seq*, *ae_array*, *count*, *ani=True*)

Generate channels architectural element:

- Flow regime is assumed to be reasonably constant so the major geometry of the channels doesn't change so much
- 'Migration' of the channels according to a shift vector

Parameters

- **ch_par** – Channel parameters
- **mg** – Mesh grid object class
- **model** (*dict*) – Model domain parameters
- **seq** (*dict*) – Sequence parameters
- **ae_array** – Array with architectural element unit details
- **count** (*int*) – Material number and/or identifier
- **ani** (*bool*) – Boolean if anisotropy is to be generated
- **(z_in** – starting depth)
- **(thickness** – Thickness of architectural element)

Returns

Contains data of architectural element units and associated hydrofacies

if 'ani':

mat = Material values azim = Azimuth angles dip = Dipping angles fac = Facies values ae_arr_i = Array with architectural element unit details

else

mat = Material values fac = Facies values ae_arr_i = Array with architectural element unit details

count (int): Material number and/or identifier

Return type probs

hyvr.hyvr.sim.**gen_sheet**(*sh*, *mg*, *ae_i*, *ae_array*, *count*, *ani=True*)
Generate gravel sheet with internal heterogeneity

Parameters

- **sh** – Sheet parameters
- **mg** – Model grid class
- **ae_i** – Architectural element lookup details [sequence number, z_bottom, z_top, architectural element, geometry]
- **ae_array** – Architectural element array
- **count** (*int*) – Material number and/or identifier
- **ani** (*bool*) – Boolean if anisotropy is to be generated

Returns **probs** – Contains data of architectural element units and associated hydrofacies (e.g. values of azimuth, material, dipping, etc.) count (int): Material number and/or identifier

Return type dict

hyvr.hyvr.sim.**gen_trough**(*tr*, *mg*, *model*, *ae*, *ae_arr*, *count*, *ani=True*)
Create trough shapes

Parameters

- **tr** (*dict*) – Trough parameters
- **mg** (*grid class*) – Model grid
- **ae** (*list*) – Architectural element unit details
- **ae_arr** (*ndarray*) – 3D array of sequeunce numbers
- **count** (*int*) – Material number and/or identifier
- **ani** (*bool*) – Boolean if anisotropy is to be generated

Returns **probs** – Grid properties count (int): Material number and/or identifier

Return type numpy array

hyvr.hyvr.sim.**heterogeneity**(*props*, *params*)
Generate internal heterogeneity

Parameters

- **probs** (*list*) – Data of architectural element units and associated hydrofacies (e.g. values of azimuth, material, dipping, etc.)
- **params** (*list*) – Parameters of model domain, sequence, hydraulics, etc.

Returns **probs** – Data of architectural element units and associated hydrofacies params (list): Input parameters, assigned with heterogenity

Return type list

hyvr.hyvr.sim.**main**(*param_file*)
Main function for HYVR generation

Parameters **param_file** (*str*) – Parameter file location

Returns Save data outputs as parameter file

hyvr.hyvr.sim.**planepoint**(*dip_norm*, *x_dip*, *y_dip*, *znow*, *xtemp*, *ytemp*, *ztemp*, *select=[]*)
Compute number of planes

Parameters

- **dip_norm** –
- **x_dip** – X coordinates of points on dip planes
- **y_dip** – Y coordinates of points on dip planes
- **znow** – Current coordinates of Z, needed to compute Z coordinates of points on dip planes
- **xtemp** – X dimension of model grid nodes
- **ytemp** – Y dimension of model grid nodes
- **ztemp** – Z dimension of model grid nodes
- **select** – Model grid nodes to consider

Returns Number of planes with selected model grid nodes

Return type set_no

hyvr.hyvr.sim.**prob_choose**(*choices*, *probs*)
Get random values of an architectural element

Parameters

- **choices** – Fixed number of choices
- **probs** – Contains data of architectural element units and associated hydrofacies

Returns Random value of architectural elements

Return type choice

hyvr.hyvr.sim.**rand_trough**(*tr*, *mg=False*, *ztr=[]*)
Randomly generate ellipsoid geometry parameters:

Parameters

- **tr** – Ellipsoid parameters
- **mg** – Meshgrid object
- **ztr** – Elevation of ellipsoid centre point

Returns Length, width and depth of ellipsoid

Return type a, b, c

hyvr.hyvr.sim.**reindex**(*inray*)
Reindex array from 0

Parameters **inray** – Array with indices

Returns Vectorized function of inray

Return type vecmat

hyvr.hyvr.sim.**save_arrays**(*arr_size*, *bg=False*, *mat_count=0*, *ani=True*)
Generate arrays for material properties storage

Parameters

- **arr_size** – Size of array
- **bg** – List of background values for each array

- **ani** (*bool*) – Boolean if anisotropy is to be generated

Returns Material values fac: Facies values

Return type mat

hyvr.hyvr.sim.**save_models**(*realdir*, *realname*, *mg*, *outputs*, *flowtrans*, *k_iso*, *ktensors*, *poros*, *anirat*, *dip*, *azim*)

Save HYVR outputs to standard modelling codes

Parameters

- **run** (*dict*) – Model run parameters
- **mg** – Mesh grid object class
- **flowtrans** (*dict*) – Flow & transport simulation parameters
- **k_iso** – Horizontal hydraulic conductivity array
- **ktensors** – Array with tensor values of K
- **poros** – Porosity array
- **anirat** – Anistropic ratio (K_h/K_v)

Returns Save data outputs as .mf (MODFLOW) or .hgs (HydroGeoSphere)

hyvr.hyvr.sim.**save_outputs**(*realdir*, *realname*, *outputs*, *mg*, *outdict*)

Save data arrays to standard formats

Parameters

- **realdir** (*str*) – File path to save to
- **realname** (*str*) – File name
- **run** (*dict*) – Model run parameters
- **mg** – Mesh grid object class
- **outdict** – Output directory

Returns Save data outputs as .vtk (Paraview), .mat (Matlab) or .dat (Python pickle output)

hyvr.hyvr.sim.**scale_rotate**(*x*, *y*, *z*, *alpha=0*, *a=1*, *b=1*, *c=1*)

Scale and rotate three-dimensional trough

Parameters

- **y, z** (*x,*) – Spatial coordinates
- **alpha** (*float*) – Rotation angle about the z-axis
- **b, c** (*a,*) – Axis lengths in x, y, z directions (ellipsoid length, width, depth)

Returns Grid cells within ellipsoid R2: Grid of scaled and rotated values

Return type select

hyvr.hyvr.sim.**thetaAR2**(*t1*, *t2*, *k*, *h*, *eps*)

Implementation of AR2 autoregressive model (Ferguson, 1976, Eq.15) http://onlinelibrary.wiley.com/doi/10.1002/esp.3290010403/full

Parameters

- **t1** – theta(i-1)
- **t2** – theta(i-2)

- **k** – Wavenumber
- **h** – Height
- **eps** – Random background noise

Returns 2nd-order autoregression (AR2)

7.1.4 hyvr.hyvr.utils module

Some utility functions for HFM modelling

Authors Jeremy P. Bennett, with help from Alessandro Comunian

Notes

hyvr.hyvr.utils.**calc_norm**(*x*)
Calculate norm (compute the complex conjugate from 'x')

Parameters **x** – Input parameter

Returns Complex conjugate of x

hyvr.hyvr.utils.**dem_load**(*fn*)
Load data from ESRI-style ASCII-file.

Parameters **fn** (*str*) – Directory and file name for save

Returns **data** – Data from ERSI-style ASCII-file meta (dict): Dict with grid metadata

Return type numpy array

hyvr.hyvr.utils.**dem_save**(*fn*, *data*, *gro*)
Save DEM data to ESRI-style ASCII-file

Parameters

- **fn** (*str*) – Directory and file name for save
- **data** (*numpy array*) – DEM data
- **gr** (*object class*) – grid.Grid() object class

Returns Save DEM data to ESRI-style ASCII-file

hyvr.hyvr.utils.**get_boreholes**(*bh_loc*, *fin*, *fout=None*)
Get virtual borehole data Returns values at centroids - this might not match the borehole inputs

Parameters

- **bh_loc** – Location of boreholes
- **fin** – Filepath of properties input
- **fout** – Filepath to save borehole information

Returns X, Y values at centroids

Return type bhdf

hyvr.hyvr.utils.**load_gslib**(*fn*)
Load .gslib files. This has been appropriated from the HPGL library https://github.com/hpgl/hpgl/blob/master/src/geo_bsd/routines.py commit b980e15ad9b1f7107fd4fa56ab117f45553be3aa

Parameters **fn** (*str*) – .gslib file path and name

Returns **gslib_dict** – properties

Return type dict

hyvr.hyvr.utils.**load_pickle**(*pickfile*)
Pickle input file

Parameters **pickfile** – Input file

Returns **data** – Pickled data of input file

Return type dict

hyvr.hyvr.utils.**matlab_save**(*fn*, *data*)
Save numpy array to .mat file for use in matlab.

Parameters

- **fn** (*str*) – File name (ending with .mat)
- **data** (*numpy array*) – Data to save

Returns Save a dictionary of names and arrays into a MATLAB-style .mat file. This saves the array objects in the given dictionary to a MATLAB- style .mat file.

hyvr.hyvr.utils.**model_setup**(*pf*)
Set up model using grid.Grid() class and assign parameters

Parameters **pf** (*str*) – Parameter file path

Returns **run** – Model run parameters mod (dict): Model domain parameters sequences (dict): Sequence parameters hydraulics (dict): Hydraulic properties parameters flowtrans (dict): Flow & transport simulation parameters elements (dict): Architectural elements and parameters model_grid (object class): Grid object class

Return type dict

hyvr.hyvr.utils.**parameters**(*file_in*)
Get parameters for hierarchical facies modelling

Parameters **file_in** (*str*) – Parameter file path

Returns **run** – Model run parameters model (dict): Model domain parameters sequences (dict): Sequence parameters hydraulics (dict): Hydraulic properties parameters flowtrans (dict): Flow & transport simulation parameters elements (dict): Architectural elements and parameters

Return type dict

hyvr.hyvr.utils.**read_lu**(*sq_fp*)
Load user-defined sequences (architectural element lookup table), split the data based on a delimiter and return it as a new list

Parameters **sq_fp** – Load user-defined sequences (architectural element lookup table)

Returns **seq_lu** – Values of architectural element lookup table

Return type list

hyvr.hyvr.utils.**rotate_ktensor**(*count*, *aniso*, *azimuth*, *dip*, *k_in*)
Create a rotated K tensor

Parameters

- **count** (*int*) – Material number and/or identifier
- **aniso** – Anisotropy
- **azimuth** – Azimuth angles

- **dip** – Dipping angles
- **k_in** –

  –

Returns Rotated K tensor

Return type k_rotate

hyvr.hyvr.utils.**round_x**(*x*, *base=1*, *prec=2*)
Round to the nearest z-increment (Refer to http://stackoverflow.com/questions/2272149/round-to-5-or-other-number-in-python)

Parameters

- **x** (*float*) – Input parameter
- **base** (*int*) – Base parameter for avoiding floating-point values
- **prec** – Precision of rounding

Returns Rounded value of nearest z-increment

hyvr.hyvr.utils.**specsim**(*gr*, *var*, *corl*, *twod=False*, *covmod='gau'*)
Generate random variables stationary covariance function using spectral techniques of Dietrich & Newsam (1993)

Parameters

- **gr** – Grid class object
- **var** – Variance
- **corl** – Tuple of correlation length of random variable
- **twod** – Flag for two-dimensional simulation
- **covmod** – Which covariance model to use 'gau': Gaussian 'exp': Exponential

Returns

Random gaussian variable Real part of a complex array, created via inverse discrete Fourier Transform

Return type bigy

hyvr.hyvr.utils.**to_hgs**(*hgspath*, *mg*, *flowtrans*, *ktensors*, *poros*)
Convert HYVR outputs to HydroGeoSphere inputs

Parameters

- **hgspath** (*str*) – Path where to save HGS output file
- **ktensors** – Array with tensor values of K
- **poros** – Array with values of porosity

Returns **val_fmts** – Dictionary with values of K tensor and porosity val_filepath: file name of HGS output file

Return type dict

hyvr.hyvr.utils.**to_mf6**(*mfdir*, *runname*, *mg*, *flowtrans*, *k_iso*, *anirat*, *dip*, *azim*)
Convert HYVR outputs to MODFLOW6 inputs

Parameters

- **mfdir** – Directory of MODFLOW model object

- **runname** – Run name
- **mg** – Mesh grid object class
- **flowtrans** (*dict*) – Flow & transport simulation parameters
- **k_iso** – Hydraulic conductivity of HYVR
- **anirat** – Background anistropic ratio (K_h/K_v anisotropy ratio)

Returns MODFLOW model object dis: Discretization of modflow object bas: BAS package of modflow model lpf: LPF package of modflow model oc: OC package of modflow model pcg: pcg package of modflow model

Return type mf

hyvr.hyvr.utils.**to_modflow**(*mfdir*, *mg*, *flowtrans*, *k_iso*, *anirat*)
Convert HYVR outputs to MODFLOW inputs

Parameters

- **mfdir** – Directory of MODFLOW model object
- **mg** – Mesh grid object class
- **flowtrans** (*dict*) – Flow & transport simulation parameters
- **k_iso** – Hydraulic conductivity of HYVR
- **anirat** – Background anistropic ratio (K_h/K_v anisotropy ratio)

Returns MODFLOW model object dis: Discretization of modflow object bas: BAS package of modflow model lpf: LPF package of modflow model oc: OC package of modflow model pcg: pcg package of modflow model

Return type mf

hyvr.hyvr.utils.**to_vtk**(*data*, *file_name*, *grid=None*, *sc_name=None*)
Save a numpy array into a VTK STRUCTURED_POINTS file.

Parameters

- **data** (*numpy array*) – Numpy array containing the data, *int* or *float* or *uint*. The dimensions should be between 1 and 3
- **file_name** (*string*) – Name of the file for the output, optional (None)
- **grid** (*class Grid*) – Information about the grid can be also provided as a Grid object
- **sc_name** (*string*) – Name of the scalar quantities

Returns

A VTK 'STRUCTURED_POINTS' dataset file containing the input numpy data.

hyvr.hyvr.utils.**to_vtr**(*data_dict*, *file_name*, *grid*)
Save a numpy array into a `.vtr` rectilinear grid of voxels using pyevtk

Parameters

- **data_dict** (*numpy array*) – e.g. {'fac': fac, 'mat': mat}
- **file_name** (*string*) – Name of the file for the output.
- **grid** (*class Grid*) – The information about the grid can be also provided as a grid object.

Returns

A VTK STRUCTURED_POINTS dataset file containing the input numpy data.

hyvr.hyvr.utils.**try_makefolder**(*makedir*)
Create modflow output folder

hyvr.hyvr.utils.**vtk_mask**(*data*, *out_name*, *mask_val=-15*, *grid=None*)
Prepare a mask file from a `vtk` input. All values should be 0 or 1 after this operation

Parameters

- **data** (*numpy array*) – Data to replace
- **out_name** (*string*) – Name of the file for the output.
- **mask_val** (*int*) – Value of the facies which should be masked
- **grid** – Grid class

Returns A VTK 'STRUCTURED_POINTS' dataset file containing the mask data

hyvr.hyvr.utils.**vtk_read**(*file_in*)
Reads a `.vtk` file into a numpy array

Parameters **file_in** (*str*) – Name and filepath to read

Returns **gegrid** – Grid class props (numpy array): Grid properties

Return type hyvr.grid class

hyvr.hyvr.utils.**vtk_trim**(*file_in*, *dims*, *file_out=None*)
Trims a vtk file to the desired dimensions. Removes the effort of working out the indexing in a .grdecl file Saves as a .vtk file with everything the same except the dimensions

Parameters

- **file_in** – .vtk file to trim
- **dims** – 3-tuple of dimensions
- **file_out** – Output file

Returns

Grid class of the data props: The data as a nx x ny x nz array

...notes:

- Number of cells (nx, ny, nz) is changed
- Spacing (dx, dy, dz) is NOT changed

Return type gegrid

7.1.5 Module contents

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

[BHC17] Jeremy Bennett, Claus P. Haslauer, and Olaf A. Cirpka. The impact of sedimentary anisotropy on solute mixing in stacked scour-pool structures. *Water Resources Research*, 53(4):2813–2832, April 2017. doi:10.1002/2016WR019665.

[DN93] C. R. Dietrich and G. N. Newsam. A fast and exact method for multidimensional gaussian stochastic simulations. *Water Resources Research*, 29(8):2861–2869, August 1993. doi:10.1029/93WR01070.

[Fer76] R. I. Ferguson. Disturbed periodic model for river meanders. *Earth Surface Processes*, 1(4):337–347, October 1976. doi:10.1002/esp.3290010403.

[JSD94] Peter Jussel, Fritz Stauffer, and Themistocles Dracos. Transport modeling in heterogeneous aquifers: 1. Statistical description and numerical generation of gravel deposits. *Water Resources Research*, 30(6):1803–1817, June 1994. doi:10.1029/94WR00162.

h