

# Package ‘SoPhy’

January 2, 2012

**Version** 1.0.40

**Date** 2006-07-01

**Title** Soil Physics Tools

**Author** R by Martin Schlather

<martin.schlather@math.uni-goettingen.de>; FORTRAN by Rien van  
Genuchten, Carl A. Mendoza, Rene Therrien, Edward Sudicky

**Maintainer** Martin Schlather <martin.schlather@math.uni-goettingen.de>

**Depends** R (>= 2.2.0), RandomFields (>= 1.3.27)

**SystemRequirements** linux: libtiff (and libtiff-devel); windows: Tiff  
(libtiff3.dll), zlib1.dll and jpeg62.dll; see also the topic ‘SoPhy’ in the documentation

**Description** SWMS\_2D interface

**License** GPL

**URL** <http://www.stochastik.math.uni-goettingen.de/~schlather>

**Repository** CRAN

**Date/Publication** 2011-10-27 05:44:11

## R topics documented:

ADE . . . . .	2
analyse.profile . . . . .	3
calculate.horizons . . . . .	9
create.roots . . . . .	10
create.stones . . . . .	13
create.waterflow . . . . .	15
draw.horizons . . . . .	17
eval.parameters . . . . .	18
flowpattern . . . . .	22
getactions . . . . .	27

Locator . . . . .	27
modify.horizons . . . . .	28
my.legend . . . . .	30
Pareto . . . . .	31
plotFlow . . . . .	32
plotRF . . . . .	34
plotRGB . . . . .	36
plotWater . . . . .	37
Quader . . . . .	39
read.picture . . . . .	41
read.swms2d.table . . . . .	42
Readline . . . . .	45
risk.index . . . . .	46
sh . . . . .	49
simulateHorizons . . . . .	51
SoPhy . . . . .	53
swms2d . . . . .	55
tortuosity . . . . .	60
Tracer experiments . . . . .	62
useraction . . . . .	63
write.picture . . . . .	66
xswms2d . . . . .	67
<b>Index</b>	<b>88</b>

---

 ADE

*Advection-dispersion equation*


---

### Description

Solution of a simple advection-dispersion equation

### Usage

ADE(z, time, C0, dispersion, velocity)

### Arguments

z	vector of distances from surface
time	vector of time points
C0	constant concentration at the surface
dispersion	constant dispersion coefficient
velocity	constant velocity

**Details**

ADE solves the PDE

$$\frac{\partial C(z,t)}{\partial t} = D \frac{\partial^2 C(z,t)}{\partial z^2} - v \frac{\partial C(z,t)}{\partial z}$$

on the one-dimensional semi-finite column, i.e.  $z \in [0, \infty]$ . Here  $D$  is the dispersion coefficient and  $v$  the velocity. Further,  $C(\infty, t) = 0$  for all  $t$  and  $C(z, 0) = 0$  for all  $z$ . Further,  $C(0, t) = C_0$ .

**Value**

ADE calculates the concentration on a grid given by  $z$  and  $time$ , i.e., ADE returns a  $(\text{length}(z) \times \text{length}(time))$ -matrix.

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**References**

Schlather, M. and Huwe, B. (2004) The use of the language interface of R: two examples for modelling water flux and solute transport. *Computers & Geosciences* **30**, 197-201.

Tindall, J.A. and Kunkel, J.R (1999) *Unsaturated Zone Hydrology for Scientists and Engineers*. New Jersey: Prentice-Hall.

**Examples**

```
par(cex=1.5, mar=c(4.2,4,0.2,0.4))
z <- ADE(z=1:100, t=10^(0:3), C0=1, d=1, v=1)
matplot(z, outer(-1:-100, rep(1,4)), type='l',
        xlab='concentration', ylab='z',
        lwd=3, cex.lab=1.5, cex.axis=1.5, col=1)
legend(1.0, -100, legend=paste('t=', 10^(0:3), sep=''), xj=1, yj=0,
        lwd=3, lty=1:4, cex=1.5, col = 1)
```

**Description**

The function fits the Pareto distribution to a flow pattern obtained from a dye experiment

**Usage**

```
analyse.profile(picture, fct, param, lower, upper, loc,
  estimate.all=NULL, selected.dist=0.95, selected.rate=c(0.5, 0.8),
  measure=function(x) x^2, method=c("fix.m","optim.m", "ml"),
  endpoint.tolerance=-10,
  ppi.par = 5,
  ppi.xy = c(xlow=2, xup=2, ylow=2, yup=2),

  interactive=TRUE, PrintLevel=RFparameters()$Print,
  extensions=c("tif", "tiff", "gif", "jpeg"),

  X11.width=13, new=TRUE,
  col.thresh=c("white", "yellow", "black", "blue"),
  col.rect="red", col.bg="yellow", col.sep="gray",
  col.left="red", col.mid="darkgreen", col.right="darkgreen",
  col.line = "red", col.txt="black",
  reverse=TRUE, title
)
```

**Arguments**

**picture** list or array or character. If is a list, then the programme expects that the structure is of the output format of `analyse.profile`. Then the parameters `fct`, `param`, `upper`, `lower` and `loc` must be missing, and they are taken from the list. The list element `picture` might be missing and is reread from the original file.

If it is an array, a three dimensional array is expected where the third dimension has three or four components (the fourth component is ignored if any). The array is interpreted as RGB-A coded picture.

If `picture` is a character string, the function reads in a file with name or name base `picture`, see [read.picture](#).

**fct** threshold function that extracts the stained area from an RGB figure. `fct` has two input variables, the first is a three dimensional array, the second is an arbitrary list of *named* parameters. Further, the function must return a list of standard values of the parameters, if the second parameter is missing, independently of value of the first variable.

If `fct` is not given, then

```
fct <- function(i, p) {
  if (missing(p)) list(minR=160, maxGB=200)
  else {
    gb <- i[, ,2] + i[, ,3]
    (gb >= i[, ,1] * p$minR / 100 & 80 <= gb &
    p$maxGB >= gb)
  }
}
```

param	<p>list. Initial values for the parameter list of the function <code>fct</code>. If <code>param</code> is not named the names will be 'parameter1', 'parameter2', etc. Be sure that the parameter names match the names in <code>fct</code>.</p> <p>The values of <code>param</code> are changed in the interactive plot. If <code>param</code> is not given <code>param</code> takes the value returned by <code>fct()</code>.</p>
lower	<p>initial values for the lower bounds of the range of the parameters in <code>fct</code>. The values can be changed in the interactive plot. If not given <code>lower</code> is set to <code>param</code>.</p>
upper	<p>initial values for the upper bounds of the range of the parameters in <code>fct</code>. The values can be changed in the interactive plot. If not given <code>upper</code> is set to <code>param</code>.</p>
loc	<p>missing or list or list of lists. If it is a list then it must have the the components <code>x</code> and <code>y</code>. Each of the components are vectors of two components containing the <code>x</code> and <code>y</code> coordinates of two opposite corners of a rectangle, respectively. The rectangle gives the clipping area for which the number of stained pixels is calculated. The values can be changed interactively.</p> <p>If <code>loc</code> is missing <code>loc</code> is determined in the interactive plot.</p> <p>If <code>loc</code> is a list of lists then it has three components, each of which is a list as described above. The first component gives the clipping reactangle (that might be thought to be best), the other two the minimal and maximal tolerable cutting areas. If an edge of the last two rectangle agrees with an edge of the best rectangle up to 2 pixels, it is assumed that the user wishes that the two edges agree.</p> <p>Note that the ideal rectangle is first given by two points which are the opposite corners of the rectangle. This rectangle and the following ones are changed by giving a further point; the programme guesses suitably well which corner should be moved into which position.</p> <p>if <code>loc</code> is not given and <code>interactive=FALSE</code> then the whole picture is taken.</p>
estimate.all	<p>NULL or logical. If NULL then nothing is estimated. If FALSE only a single risk index according to <code>param</code> is estimated; if TRUE the risk index is calculated for a grid given by <code>lower</code>, <code>upper</code> and the second and third component of <code>loc</code>. The marginal number of grid points is <code>ppi.par</code> for the components of <code>lower</code> and <code>upper</code> and <code>ppi.xy</code> for the definition of the clipping rectangle; see also <a href="#">risk.index</a></p>
selected.dist	<p>number in <math>(0, 1)</math> or a vector of integers. Distances for which the form parameter of the pareto distribution is estimated; see <a href="#">risk.index</a>. If <code>selected.dist</code> is a number in <math>(0, 1)</math> the distances are <math>1:(1 + \text{round}(\max(\text{data}[,1]-1) * \text{selected.dist}))</math>.</p>
selected.rate	<p>additionally to the indices given by <code>selected.dist</code>, the form parameter is estimated also for those distances where the corresponding number of observed relative stained pixels (w.r.t. to the maximum number of observed pixels) is within the interval given by <code>selected.rate</code>. The risk index is calculated as the median of the estimated form parameters.</p>
measure	<p>instead of the default least squares another distance function can be given.</p>

method	the number of observed paths is a free parameter when fitting the Pareto distribution. It can either be set as the maximum number of stained pixels for the currently considered distances or depths ('fix.m') or fitted within the optimisation algorithm ('optim.m'). Usually, it is not worth using the slower 'optim.m' option. As third new option the maximum likelihood method "ml" is available.
endpoint.tolerance	<p>optimisation parameter. If the shape parameter is negative then the distribution has a finite upper endpoint. Hence, mathematically, the lowest upper end point of the Pareto distribution is given as the largest distance for which at least one stained pixel is observed. For stability reasons and because the observed data might be a scale mixture of Pareto distribution it is advantageous to allow for some tolerance of the minimal upper end point.</p> <p>If endpoint.tolerance is positive then the lower threshold for the upper end point is the largest distance for which the number of observed stained pixels is larger than endpoint.tolerance.</p> <p>If endpoint.tolerance is negative then the lower threshold equals the largest distance for which at least one stained pixel is observed minus the modulus of endpoint.tolerance.</p>
ppi.par	<p>list or scale. Number of points by which a given parameter interval is divided (if the interval is not trivial). The points of all intervals (and those for the clipping area) build a grid. For each point of the grid, the risk index is estimated, if estimate.all=TRUE. ppi.par must be given by name (or a single integer is given then the number of grid points are the same for each parameter); see also the variable param.</p> <p>If ppi.par is NULL and picture is a list, the value in picture is used.</p>
ppi.xy	<p>vector of 4 components; number of points by which the interval for the positions of the lower, upper, left and right edge of the clipping area is divided.</p> <p>If ppi.xy is NULL and picture is a list, the value in picture is used.</p>
interactive	logical. If TRUE the user can modify the parameters interactively.
PrintLevel	<p>numeric.</p> <ul style="list-style-type: none"> <li>• &lt;=0 no messages are printed</li> <li>• &gt;=1 the higher the number the more information is given</li> </ul>
extensions	accepted extensions for the graphics. Used if picture is a character
X11.width	width of the interactive plot. The height is determined automatically. Only used if new=TRUE
new	if TRUE a new window for the interactive plot is opened
col.thresh	<p>vector of 4 colour specifications. The components are the colours for</p> <ol style="list-style-type: none"> <li>1. the pixels that are not considered as stained by the current and the preceding parameter choice</li> <li>2. the pixels that are considered as stained by the preceding parameter choice, but not by the current one.</li> <li>3. the pixels that are considered as stained by the current parameter choice, but not by the preceding one</li> <li>4. the pixels that are considered as stained by the current and the preceding parameter choice</li> </ol>

	If the values of an interval for a parameter is changed than the parameters in param are considered as the the preceding parameters.
col.rect	colour for the interactive menu: colour of a button for free input.
col.bg	colour for the interactive menu: colour of an interactive bar
col.sep	colour for the interactive menu: colour of a separating line
col.left	colour for the interactive menu: colour of a preceding element
col.mid	colour for the interactive menu: colour for a message
col.right	colour for the interactive menu: colour of a subsequent element
col.line	colour for the interactive menu: colour of a marking line in interactive bars of direct choice.
col.txt	colour for the interactive menu: colour of headers
reverse	logical; see details
title	title for the interactive plot; if not given the picture name is used (usually the file name)

### Details

If the picture is read in from the file the following particular case may appear. In soil profiles often a metal grid is placed of size 1m x 1m. This metal grid must be first removed by using GIMP ([www.gimp.org](http://www.gimp.org)), for example, and, in general, also the figure has to be rotated. The pixels of the metal bars are removed by coding them 255, 255, 255 in RGB code. To get a reliable estimation of the risk index, the number of removed pixels must be the same for each horizontal line in the chosen clipping area, except the horizontal line consists of white pixels only that should be removed. Now the programme sets each pixel of a horizontal line to NA if at least 70% of the pixels of that line are coded 255, 255, 255. Otherwise only the white pixels are set to NA.

Plotting aspects:

reverse=TRUE is for the plot of soil profiles, where the numbering of the y-axis start at the top and is counted positively downwards.

reverse=FALSE The usual presentation of the vertical axis.

The functions checks whether y is in decreasing or increasing order, and plots the y axis accordingly.

### Value

returns a list of the following arguments. Since the arguments do not have a fixed order they should be called by name, only.

picture	a three dimensional array, containing the RGB-A coding
fct	see the variable fct above
param	see the variable param above
lower	see the variable lower above
upper	see the variable upper above
loc	see the variable loc above

ppi.par	see the variable ppi.par above
ppi.xy	see the variable ppi.xy above
r.i	output of <a href="#">risk.index</a> for the ideal parameters
risk.index	array of $2 + \text{length}(\text{param})$ dimensions containing the estimated risk indices obtained by the sensitivity analysis. The latter is based on intervals of values for the edges of the clipping area, see <code>loc</code> and <code>ppi.xy</code> and the intervals for the parameters, see <code>lower</code> , <code>upper</code> and <code>ppi.par</code> . The first two dimensions of <code>risk.index</code> are for the horizontal and vertical variation of the rectangle, the following dimensions are for the parameters, given in the order of <code>param</code> . The number of components in each dimension is given by <code>ppi.xy</code> and <code>ppi.par</code> except an interval is trivial. Then the number of components is 1.
raw.risk.index	same as <code>risk.index</code> except that the raw risk index is stored, see the function <a href="#">risk.index</a> . The raw risk index is usually not used.
absfreq	vector. Number of blue pixels per line (from top)

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**References**

Schlather, M. and Huwe, B. (2005) A risk index for characterising flow pattern in soils using dye tracer distributions *Submitted to J. Contam. Hydrol.*

**See Also**

[risk.index](#), [SoPhy](#)

**Examples**

```
data("K06")
path <- paste(system.file(package='SoPhy'),"tracer", sep="/")
K06$name <- paste(path, "K06.G.tif", sep="/")
rate <- if (interactive()) c(0.5, 0.8) else c(0.6, 0.67)
```

```

ri <- analyse.profile(K06, estimate.all=FALSE, method="fix.m",
                     selected.rate=rate, selected.dist=NULL,
                     Print=2)
cat("estimated risk index =", ri$r.i$risk.index, "\n")

```

---

calculate.horizons      *Update of the horizon structure*

---

## Description

The function updates the positions of the boundary points, the affiliation of the pixels to the horizons and the location and the size of the smallest rectangle that include a horizon.

## Usage

```
calculate.horizons(h)
```

## Arguments

`h`                      a list of the same format as the output of [xswms2d](#)

## Value

a list of the same format as the output of [xswms2d](#). The values of the following list elements are changed:

<code>h[[i]]\$points</code>	points defining the vertices of the boundary line for horizon <i>i</i> ; changed if x-coordinates of a boundary definition are all genuinely within the grid size
<code>idx.rf</code>	affiliation number of each grid point
<code>h[[i]]\$border</code>	grid points defining the boundary of horizon <i>i</i>
<code>h[[i]]\$cut.x</code> , <code>h[[i]]\$cut.y</code>	clipping rectangle that includes the horizon <i>i</i>
<code>h[[i]]\$idx</code>	indication of the points within the clipping area that belong to horizon <i>i</i>
<code>RF</code>	NULL
<code>Stone.R</code>	NULL
<code>Root.RF</code>	NULL

## Note

The function must be called after the definition of the horizons has been changed by the user. If a first or a last defining point for a boundary is truly within the range of the x-coordinates of the grid then a point with the respective x coordinate on the boundary of the grid is added.

## Author(s)

Martin Schlather, <[martin.schlather@math.uni-goettingen.de](mailto:martin.schlather@math.uni-goettingen.de)> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**See Also**

[draw.horizons](#), [modify.horizons](#), [plotRF](#), [plotWater](#), [simulateHorizons](#), [SoPhy](#), [swms2d](#) [xswms2d](#)

**Examples**

```
##### fetch the standard definition of a profile #####
h <- xswms2d(xlim=c(1, 100), ylim=c(1, 100), step=1, new=NULL)
##      new=NULL: xswms2d returns the standard definition for h
##                                     without entering the interactive surface

##### modify the profile definition #####
h$n <- 2      ## define a second horizon
h$H2 <- h$H1
h$H2$type <- "H" ## genuine horizon, not polygon
h$H2$model$model$scale <- 30 ## different
## coordinates of the boundary segments between the horizons:
h$H2$points <- list(x=seq(1, 100, 5))      ## x coordinates
h$H2$points$y <- 80 + 20 * cos(h$H2$points$x / 10) ## y coordinates
h <- calculate.horizons(h)      ## update the internal parameters
## after modifying the profile definition
draw.horizons(h)      ## plot the profile

##### simulate water flux for the specified profile #####
h <- simulateHorizons(h)      ## simulate stochastic components
plotRF(h)      ## plot the simulated random field
swms2d.out <- swms2d(h, iter.print=1) ## numerical simulation
if (is.character(swms2d.out)) {
  cat(swms2d.out, "\n")      ## an error has occurred
} else {
  plotWater(swms2d.out, what="theta") ## plot the water contents
}
```

---

create.roots

*Simulation of roots*

---

**Description**

This function generates root systems within a profile according to a given specification of the soil and the root model.

**Usage**

```
create.roots(h, trials = 10, PrintLevel=RFparameters()$PrintLevel,
             message=NULL)
```

**Arguments**

h	a list of the same format as the output <code>xswms2d</code> , see there for a description of the components.
trials	number of trials to find a realisation of the plant positions (by a trial and error algorithm) where the distances have a distance of at least <code>\$plants.mindist</code> .
PrintLevel	The higher the value the more tracing information is given; up to value 2, no information is given.  Note that if <code>PrintLevel&gt;2</code> a running counter is shown that includes the printing of backspaces (^H). The backspaces may have undesirable interactions with some few other R functions, e.g. <code>Sweave</code> . See package <code>RandomFields</code> for the default option <code>RFparameters()\$Print</code> .
message	function with one parameter (string). The function is used to plot messages.

**Details**

The simulation algorithm works currently as follows.

1. For each plant type and according to `$plants.lambda` a Poisson random variable is drawn which gives the number of plants of this type. The starting points of the plants are created successively where the horizontal positions are uniformly distributed over the profile. The vertical position of the starting point is 1 plus the vertical position of the first pixel that is not air at the drawn horizontal pixel. If the simulated horizontal distance has distance less than `$plants.mindist` to any other plant of this type, the horizontal distance is redrawn up to `trials - 1` times. If still not successful the last drawn position is taken.
2. In the second step the root growth is simulated, for each plant separately and independently. For each plant the following steps are iterates until or root segments have stopped growing or the aimed total root length has been reached. In each step a list of active root segments is considered that is ordered with respect to some priority value  $v$ 
  - (a) The root with the highest priority stops growing with probability `$stop.probab`. If this happens this root segment is dropped from the list. This step is repeated until the list is void (then the simulation of the root system is terminated) or the growth does not stop.
  - (b) The list  $p$  of neighbours of the selected root is determined. Out of the 8 potential neighbours only those are selected which do not leave the profile and that do not entre free air or a stone. In case `no.own.root` is TRUE the neighbouring pixel may also not be a already simulated segment of the current root system.
  - (c) The priority number of the remaining potential new root segments is a sum of the following values
    - i. the direction change from previous segment to current to next is calculated (separately for x and y coordinate); this value is the mean and the standard deviation (up to `$dir.ch.s`) of a Gaussian random variable.
    - ii. `$depth.bonus` or `$side.bonus` if the neighbouring pixel will be in one of the three pixels in vertical direction or in one of the two horizontal directions.
    - iii. `$rf.link` function value for `rf` that can be interpreted as water availability of soil quality.

- (d) if the distance to the previous knot is greater than `knot.mindist` then the current root segment is a knot with probability `knot.probab`.  
The number of children is determined both by the number of available neighbouring pixels and the values of `shoots.4` and `shoots.3`.
- (e) The lengths of the root segments to be added are calculated. If then the aimed total length is exceeded, (biased) coins are thrown which of the neighbouring pixels are kept
- (f) The remaining children/neighbouring pixels are added to the list of active root segments; The current, selected root segment is deleted from the list.
- (g) The priority of root segments that have already been created in former steps are multiplied by `age.bonus`.

### Value

The function returns the input argument `h` except that `h$plants`, `h$plant.idx` and `h$rf` have been updated.

<code>\$plants</code>	is a list of simulated plants where each component contains the positions of the roots. The latter is given by a matrix of 8 columns: <ol style="list-style-type: none"> <li>1. horizontal coordinate of a root segment. The coordinate is given in pixels.</li> <li>2. vertical coordinate of a root segment, For the first pixel it is 1 plus the vertical position of the first pixel that is not air at the drawn horizontal pixel.</li> <li>3. index for the parent root pixel</li> <li>4. index where the subsequent root pixel is found; NA if it is a terminating root segment. If the pixel is a knot with <math>k</math> children then the value gives the position of the first child, the others follow immediately.</li> <li>5. number of children in case it is a knot, 1 otherwise</li> <li>6. the number of preceding knots until and including this position</li> <li>7. aimed random total length of the roots if it is the first pixel, and the current distance to the surface along the roots, otherwise.</li> <li>8. distance to the preceding knot</li> </ol>
<code>\$plant.idx</code>	vector or NULL. <code>h\$plant.idx[i]</code> gives the plant type (1, . . .) of the $i$ th simulated plant.
<code>\$Root.RF</code>	<code>\$Stone.RF</code> modified as follows: if <code>h\$root[[i]]\$rf.Kf</code> is TRUE then the locations of the roots of plant type $i$ get the value of the function <code>h\$Kf</code> .

### Author(s)

Martin Schlather, <[martin.schlather@math.uni-goettingen.de](mailto:martin.schlather@math.uni-goettingen.de)> <http://www.stochastik.math.uni-goettingen.de/~schlather>

### References

Schlather, M. and Huwe, B. (2006) Modelling inhomogeneous soils: Theory. *In preparation*.

**See Also**[SoPhy](#), [xswms2d](#)**Examples**

```
## get a standard definition first
h <- xswms2d(xlim=c(1, 100), ylim=c(1, 100), step=1, new=NULL)
h$root[[1]]$plants.lambda <- 0.02
h <- simulateHorizons(h)
for (i in 1:5) {
  plotRF(create.roots(h), cex.pch=0.5, pch=16, root.col=1)
  readline("Press return")
}
```

---

`create.stones`*Simulation of stones*

---

**Description**

This function simulates stones within a profile according to a given soil definition. The algorithm returns an intermediate stage of the random sequential adsorption (RSA) model.

**Usage**

```
create.stones(h, trials = 10)
```

**Arguments**

<code>h</code>	a list of the same format as the output of <a href="#">xswms2d</a>
<code>trials</code>	parameter for the number of trials after which the simulation for a horizon terminates and a warning is given. The number of trials equals, for each horizon, <code>trials</code> times the Poisson random number of stones to simulate, see also <a href="#">Details</a> .

**Details**

In the simulation, each horizon is considered separately in the ordering they are defined. The ordering is important in case the stones are allowed to reach into neighbouring horizons. For each horizon:

1. a Poisson number  $p$  of stones is drawn with mean  $h$stone$lambda$  times modulus of the area of the horizon [in the given units].
2. while not all  $p$  points are simulated and the total number of trials for the horizon is less than `trials` times  $p$ :

- (a) within the rectangle that is given by `cut.x` and `cut.y` and that encloses the area of the horizon a uniformly distributed point  $P$  is draw. If the point is outside the horizon, the drawing is repeated, up to `trials` times. If still not successful, the simulator for a horizon terminates, a warning is given, the stones simulated up to now are kept and the next horizon is simulated.
  - (b) According to `$stone$phi.distr`, `$stone$phi.mean`, `$stone$phi.s`, `$stone$main.distr`, `$stone$main.mean`, `$stone$main.s`, `$stone$sec.distr`, `$stone$sec.mean`, and `$stone$sec.s`, an ellipse is simulated with centre  $P$ . The length of the main axis is given by a random variable with distribution `stone$main.distr`, mean parameter `$stone$main.mean` and standard deviation `$stone$main.s`. The length of the secondary axis and the angle between the main axis and the horizon are given in a similar way.
  - (c) The pixels that are covered by the ellipse are determined.
  - (d) The ellipse including  $P$  is rejected in the following cases
    - i. `x $stone$no.overlap` is TRUE and any of the pixels belong already to a stone
    - ii. `$stone$no.lower` is TRUE and any pixel belongs to a horizon or polygon that precedes the current one.
    - iii. `$stone$no.upper` is TRUE and any pixel belongs to a horizon or polygon that follows the current one.
3. if the total number of trials is reached without having  $p$  points, the algorithm fails.

### Value

The function returns the input argument `h` except that `h$idx.rf` and `h$Stone.RF` have been updated.

`$idx.rf` gives, at first instance, the association of a pixel to a horizon (0,...,h\$N-1). If the pixel belongs to one or more stones the value of the maximum number of horizons is added.

`$Stone.RF` RF, modified at the locations with stones according to the `h[[...]]$stone$values`.

### Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

### References

Schlather, M. and Huwe, B. (2006) Modelling inhomogeneous soils: Theory. *In preparation*.

Stoyan, D., Kendall, W.S., Mecke, J. (1995) *Stochastic Geometry and its Applications* Chichester: Wiley, 2nd edition.

RSA model (also called SSI process in stochastic geometry)

- Herczynski, R. (1975) Distribution function for random distribution of spheres. *Nature*, **255**, 540-541.

**See Also**[SoPhy](#), [xswms2d](#)**Examples**

```
## get a standard definition first
h <- xswms2d(xlim=c(1, 100), ylim=c(1, 100), step=1, new=NULL)
h <- simulateHorizons(h)
h$H1$stone$lambda <- 0.002      ## in average 20 stones
h$H1$stone$no.overlap <- FALSE ## allow for overlapping stones
for (i in 1:5) {
  plotRF(create.stones(h), what="Stone")
  readline("Press return")
}
```

---

create.waterflow	<i>Preparation for swms2d</i>
------------------	-------------------------------

---

**Description**

The function creates the input list for [swms2d](#)

**Usage**

```
create.waterflow(h, Kat = 2, MaxIt = 20, hTab = c(0.001, 200),
                dtMinMax = h$water$dtMinMax, DMu1 = c(1.1, 0.33),
                atmadd=FALSE
                )
```

**Arguments**

h	a list of the same format as the output <a href="#">xswms2d</a>
Kat	type of flow system: <ul style="list-style-type: none"> <li>• 0 : horizontal</li> <li>• 1 : axisymmetric (well or single drain)</li> <li>• 2 : vertical (profile)</li> </ul>
MaxIt	maximum number of iteration during any time step.
hTab	c(hTab1, hTabN), interval of pressure heads within which a table of hydraulic properties is generated.
dtMinMax	(dtmin, dtmax), minimum and maximum permitted time increment.

DMu1	$c(dMu1, dMu12)$ , $dMu1 \geq 1$ , $dMu12 \leq 1$ ; if number of required iterations is less than 4 or greater than 6 then the next time step is multiplied by $dMu1$ and $dMu12$ , respectively.
atmadd	logical. If a segment contains several roots, the potential water uptake for all the roots is added if <code>atmadd=TRUE</code> . Otherwise the maximum potential water uptake is assigned to the segment.

## Details

The function is usually called internally; see Examples for an explicit call. The function performs several tasks

- According to `h$water$red` the grid is thinned.
- Several checks are performed to insure that the users input parameters are sound.
- the boundary points and the numerical conditions are determined.
- the root uptake is determined. This functionality is still in an experimental stage.

If several root segments belong to a single pixel then the boundary condition is taken that appears most. If equality, then in the order of atmospheric condition, Neumann condition, and Dirichlet condition, the first one is taken as condition for the pixel.

In case of the atmospheric condition, the root uptake function is the minimal function of the standard form (given by  $P0$ ,  $P2H$ ,  $P2L$ , and  $P3$ ), such that the values equal the maximal root uptake function at the considered pixel.

For the Neumann condition, the sum of the Neumann values is taken over all root segment at the pixel with Neumann condition.

For the Dirichlet condition, the minimum over all Dirichlet values is taken.

- the finite element mesh is created, where points that belong to the atmosphere or to a non-penetrable stone (`$stone$value=NA`) are excluded.

## Value

The output is a list suitable as input list for `swms2d`. Additionally, the list has the following elements:

<code>flux</code>	<code>a length(water.x)·length(water.y)</code> vector of logical values indicating which grid elements of the simulation grid are part of the finite element mesh.
<code>water.x</code>	the x coordinates of the thinned grid
<code>water.y</code>	the y coordinates of the thinned grid

## Author(s)

Martin Schlather, <[martin.schlather@math.uni-goettingen.de](mailto:martin.schlather@math.uni-goettingen.de)> <http://www.stochastik.math.uni-goettingen.de/~schlather>

## See Also

[modify.horizons](#), [plotRF](#), [plotWater](#), [simulateHorizons](#), [SoPhy](#), [swms2d](#), [xswms2d](#)

## Examples

```

h <- xswms2d(xlim=c(1, 50), ylim=c(1, 30), step=1, new=NULL)
h$water$red <- 1
h$water$top.value <- -50 ## pressure head at surface
h$water$TPrint <- 10
h <- simulateHorizons(h) ## simulation of the stochastic components
plotRF(h)
swms2d.in <- create.waterflow(h) ## rewrite h as a list readable by
##      swms2d; the following function can also directly be called
##      with the argument h; then create.waterflow is called
##      internally
str(swms2d.in)
swms2d.out <- swms2d(swms2d.in, iter.print=1) ## numerical simulation
if (is.character(swms2d.out)) cat(swms2d.out, "\n") else
plotWater(swms2d.out, what="H")

```

---

draw.horizons	<i>Draw of the profile of horizons</i>
---------------	--

---

## Description

The function draws a profile, i.e. a sequence of horizons, by coloured areas or by boundary lines.

## Usage

```

draw.horizons(h, areas=TRUE,
              col.hor=c('#000000', '#996600', '#660000', '#CC9933',
                       '#666600', '#CCCC99', '#CCCCCC', '#990000',
                       '#FFCC00', '#FFFFCC', rep('white', h$max.horizons)),
              border.col=NULL, picture=NULL,
              lwd = 2, quadratic=TRUE, all=TRUE, cex = 1, cex.leg = 1
              )

```

## Arguments

h	a list of the same format as the output of <code>xswms2d</code>
areas	logical. If TRUE the horizons are plotted in different colours. If FALSE only the boundary lines are plotted.
col.hor	the colours of the horizons or the boundary lines; the vector must have 20 entries. The first ten give the colours of the horizons, the second ten give the colour of the stones that belong to a certain horizon (here the position of the pixel not the center of the stone counts).
border.col	NULL or any vector of colours. If not NULL the boundary pixels are plotted in the colour of the respective horizon.

picture	array of three dimensions where the third dimension has 3 or 4 components (RGB or RGBA coding). The pictures is given as a background figure if areas=FALSE and all=TRUE.
lwd	postive number, only used if area=FALSE. Then it gives the width of the boundary lines.
quadratic	logical. If TRUE the figure matrix is enlarged and filled with NA symmetrically in x-direction or on the bottom in y-direction such the matrix (and the figure) become quadratic.
all	If FALSE no axis and no background picture are plotted.
cex	type size for the axes and the labels, see <a href="#">par</a>
cex.leg	type size for the legend

**Value**

invisible list of xlim and ylim of the figure.

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**See Also**

[modify.horizons](#), [SoPhy](#), [xswms2d](#)

**Examples**

```
## see \link{modify.horizons} for an example
```

---

eval.parameters

*Interactive menu*

---

**Description**

eval.parameters provides an interactive menu on a X11 graphical device of R

**Usage**

```
eval.parameters(variable, entry, update, simulate, dev, create = TRUE,
  col.rect = "red", col.bg = "blue", col.sep = "grey",
  col.left = "red", col.mid = "white", col.right = "white",
  col.line = "red", col.txt = "black",
  cex=0.8, cex.i=cex, sep="-----", ...)
```

**Arguments**

variable	string. The name of the variable to be changed. The name consist of \$ and [[]] expression pointing to sublists of a list. The complete list must be given by name in ...
entry	A list of lists. See Details.
update	logical. If TRUE then simulate is called after each interactive input.
simulate	function that is called if simulations are to updated. The parameters must equal the variables given by ...; the function must return the complete list indicated by variable.
dev	Before calling eval.parameters() split.screen must have been called. dev gives the screen on which the interactive menu should be plotted.
create	logical. If TRUE missing list elements of variable are created.
col.rect	colour of the button for free input.
col.bg	colour of a interactive bar
col.sep	colour of the separating line
col.left	colour of preceding element
col.mid	colour for the message
col.right	colour of subsequent element
col.line	colour of the marking line in interactive bars of absolute choice.
col.txt	colour of headers
cex	font height of headers
cex.i	font height of text for elements
sep	style of added characters in separating line.
...	The input variables given by name; the names may not start with a dot; There the complete list to which variable refers must be given. Further additional parameters for the function simulate.

**Details**

eval.parameters shows a menu list on X11. Depending on the mode of the variables the menu bars have a different appearance and behave differently if the user clicks on the bar. Most of the menu bars have a small rectangle on the right hand side. If this rectangle is pressed the input of a variable is expected in the xterm where R is run.

entry is a list of lists. Each list may contain the following elements:

- name : header for menu button if var is not NULL; otherwise printed instead of a menu button
- var :
  - NULL : if val=NULL then it is a separating line in colour col.sep; name is surrounded by sep; all other elements of the list are ignored. If val is a string then val is interpreted as a function; a special string is "simulate", which entails the call of the function simulate with the appropri.
  - string : selected element of the list that is given by variable. A special string "undo" will be installed to undo things.

- `val` :
  - `function(d, v)` gives the update for `var`. If `v` is missing, a starting value (for  $d=1/2$ ) is expected. Otherwise, `v` is the current value of `var` and `d` is the choice of the user, a value in  $[0, 1]$
  - `TRUE` : logical bar.
  - `FALSE` : logical bar. The value for `var` is negated before shown. In the menu, the negative value for `var` is shown.
  - `NULL` : a string is read from the terminal inot `var`.
  - `character (vector)`: if `var` is given then this vector of strings is interpreted as belonging to a categorical variable  $1, \dots, \text{length}(\text{val})$  and `var` gives the number of the selected elements. If `var=NULL` then `val` is interpreted as a function; a special string is "simulate", which entails the call of the function `simulate` with the appropriate parameters.
- `delta` : logical. In the menu bar absolute values are plotted if `delta=FALSE` and increments otherwise. Only considered if `val` is a function.
- `cond` : The menu points is shown only if the given condition is satisfied. The condition must be expressed by named elements of the list variable, see example.
- `col` : colour that overwrites the standard colour for the rectangle or the separating text.
- `update` : logical. If not missing, its value overwrites locally the value of the global parameter `update`.
- `...` : additional parameters for `simulate` that overwrite the values given in `...` in the call of `eval.parameters`.

## Value

The first variable given in “...”, which is a list. To this list the entry `.history` is added.

If the users enters ‘exit immediately’ at any point, the program stops with an error message.

## Note

To the list given by `variable` the element `.history` is added. `.history` is a list that contains the history of the user input. Each element is a list where the first entry is the number of the menu, the second and the third entries are the former and the new value. Exception: for entries with character `val`, the value of `val` is returned as second entry. Consequently, the name `.history` should not be used for other purposes in `variable`.

Further, any variable name given in `...` must start with a letter.

## Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

## See Also

[useraction](#)

**Examples**

```

## Not run:
## the following lines define a menu that does not make
## too much sense, but shows the various kinds of buttons

quadratic <- function(d, v, a, mini=0, maxi=Inf) {
  d <- pmin(1, pmax(0, d)) - 0.5
  d <- ((d>0) * 2 - 1) * d^2 * a * 4
  if (missing(v)) d else pmax(mini, pmin(maxi, v + d))
}

simulate <- function(H, par) { ## not a serious example
  Print(c(H$x$var, par, runif(1)))
  return(H) ## the function must return the first parameter
}

entry <- list(
  list(name="Nonsense Menu"),
  list(name="Simulate!", val="simulate", col="blue"),
  list(name="show H", val="str(H)", col="blue"),
  list(name="colx", var="colour",
        val=c("red", "green", "blue", "brown")),
  list(name="open", var="closed", val=FALSE, par=4.5),
  list(name="modifying", var="modify", val=TRUE, par=5),
  list(name="probability", var="probab", delta=FALSE,
        val=function(d, v) pmin(1, pmax(0, d))),
  list(name="variance", var="var", delta=TRUE,
        val=function(d, v) quadratic(d, v, 10)),
  list(name="name", var="name", par=3, cond="modify")
)

scr <- split.screen(rbind(c(0, 0.45, 0, 1), c(0.5, 1, 0, 1)))
## before proceeding make sure that both the screen and the xterm
## are completely visible

H <- list(modify=5, x=list()) # note that in this example eval.parameters
## will be called by by H$x, hence modify=5 will be left
## unchanged.
options(locatorBell=FALSE)

useraction("start.register") ## registering the user's input
Print(eval.parameters("H$x", entry, simulate, update=TRUE, dev=scr[2],
  H=H, par=17)) # do not forget to call by name
getactions()

## replay the user's input
useraction("replay")
Print(eval.parameters("H$x", entry, simulate, update=TRUE, dev=scr[2],
  H=H, par=17))

## End(Not run)

```

flowpattern

*Simulation of flow pattern***Description**

Purely stochastic simulation of a water flow pattern as they appear in dye tracer experiments

**Usage**

```
flowpattern(type=c('identical', 'unif', 'independent',
                  'dependent', 'all'),
            length.profile=200, depth=100, width.slice=1,
            delta.x = depth * sqrt(x.var), delta.y = depth * sqrt(y.var),
            lambda.path=0.1, len.x, len.y, grid=FALSE,

            x.name='whittle', x.var=1, x.v.scale=10, x.kappa=2, x.h.scale=1,
            y.name=x.name, y.var=x.var, y.v.scale=x.v.scale, y.kappa=x.kappa,
            y.h.scale=x.h.scale, unif.b=2,

            drop.distr=function(x) x * 80, drop.name='whittle',
            drop.scale=1, drop.kappa=2, drops=1,

            selected.dist = 2/3, front.factor=2, method = NULL,
            endpoint.tolerance=0, measure=function(x) x^2,

            simu.method=if (type %in% c("dependent", "all")) "TBM3",
            drop.simu.method = NULL, register=c(1,2),
            old.paths=NULL, PrintLevel=RFparameters()$Print, wait=FALSE,
            raw=FALSE, compress=TRUE, max.points = 5500000)
```

**Arguments**

type	character, one of 'identical', 'unif', 'independent', 'dependent', 'all'. Type of model. See details.
length.profile	positive integer. Length of the profile.
depth	positive integer. Depth of the profile.
width.slice	positive integer. Thickness of the slice. Usually a small number.
delta.x	enlargement of the simulation block to the left and right of the profile to avoid edge effects.
delta.y	enlargement of the thickness of slice in both directions to avoid edge effects.
lambda.path	intensity of the Poisson point process at the surface; if grid=TRUE and len.x or len.y are missing they are approximated by the number the number of points of the process. If grid=TRUE and len.x and len.y are not missing, the value of lamda.path is ignored.

len.x	integer. Number of grid points in x-direction. Only used if grid=TRUE. Then it overwrites n.path
len.y	integer. Number of grid points in y-direction. Only used if grid=TRUE. Then it overwrites n.path
grid	logical. If FALSE the locations are uniformly and independently distributed.
x.name	character. Name of the covariance function used to simulate the path increments. Only used if type is different from 'unif', see <a href="#">CovarianceFct</a> .
x.var	numeric. Variance of the covariance function.
x.v.scale	numeric. Scale in vertical direction.
x.kappa	Vector of additional parameters for the covariance function, if there are any.
x.h.scale	numeric. Scale for the two horizontal directions. Only used if type='dependent' or type='all' - otherwise, the path are independent.
y.name	analogously to x.name for the y-direction
y.var	analogously to x.var for the y-direction
y.v.scale	analogously to x.v.scale for the y-direction
y.kappa	analogously to x.kappa for the y-direction
y.h.scale	analogously to x.h.scale for the y-direction
unif.b	numeric greater than; upper endpoint of the uniform distributon; relevant only if type='unif'; see Details.
drop.distr	The stained path length reached by a single drop is calculated as follows. First a Gaussian random field is simulated with the parameters given below. Then the Gaussian random field is marginally transformed to uniform distribution by the Gaussian distribution function. Then the inverse of the required distribution function is applied. drop.distr gives this inverse. It should allow for vectors as input variables.
drop.name	Name of the covariance function for the horizontal dependence of the Gaussian random field that is used to model the stained path length. Only used if type='all' - otherwise a Gaussian random field with pure nugget effect is used. See <a href="#">CovarianceFct</a> for possible values.
drop.scale	numeric. Scale in (horizontal) direction.
drop.kappa	Vector of additional parameters for the covariance function, if there are any.
drops	Number of drops running through each path. Then the stained part equals the longest stained part for all drops.
selected.dist	number in (0, 1) or a vector of integers. selected.dist used for estimating the risk index, see <a href="#">risk.index</a> . selected.dist gives the distances for which the form parameter of the pareto distribution is estimated. If selected.dist is a number in (0, 1) the distances are 1:(1 + round(max(data[,1]-1) * selected.dist)). Otherwise the integers are interpreted as indices for data.

front.factor	<p>front.factor is used for estimating the risk index, see <a href="#">risk.index</a>.</p> <p>The upper bound for the upper endpoint equals the front.factor times the largest distance for which at least stained pixel is observed.</p> <p>The value should not be changed if the algorithm is not understood.</p>
method	<p>character, 'fix.m' or 'optim.m'. method is used for estimating the risk index, see <a href="#">risk.index</a>.</p> <p>If 'fix.m', the estimated number of path equals the maximum number of ob stained pixels below a given threshold. If 'optim.m' the number of paths is estimated by optimisation.</p> <p>If method is a vector containing both values, the risk index is calculated for both methods.</p> <p>If method is NULL nothing is estimated.</p>
endpoint.tolerance	<p>numeric. endpoint.tolerance is used for estimating the risk index, see <a href="#">risk.index</a></p> <p>If the shape parameter is negative then the distribution has a finite upper endpoint. Hence, mathematically, the lowest upper end point of the Pareto distribution is given as the largest distance for which at least one stained pixel is observed. For stability reasons and because the observed data might be a scale mixture of Pareto distribution it is advantageous to allow for some tolerance of the minimal upper end point.</p> <p>If endpoint.tolerance is positive then the lower threshold for the upper end point is the largest distance for which the number of observed stained pixels is larger than endpoint.tolerance.</p> <p>If endpoint.tolerance is negative then the lower threshold equals largest distance for which at least one stained pixel is observed minus the modulus of endpoint.tolerance.</p>
measure	<p>function that measures the distance between points; the function gets and returns a vector; used for estimating the risk index, see <a href="#">risk.index</a></p>
simu.method	<p>NULL or string; Method used for simulating the Gaussian random fields for the paths, see <a href="#">RFMethods</a></p> <p>to get all options. If model is given as list then method may not be set if model[[i]]\$Method, <math>i = 1, 3, ..</math> is given, and vice versa</p> <p>"TBM3" allows for some special effect: grid points and arbitrary points can taken from the same realisation by two subsequent calls of the function.</p>
drop.simu.method	<p>NULL or string; Method used for simulating the Gaussian random fields for the drops, see <a href="#">simu.method</a></p>
register	<p>vector of 2 components. The first component gives the register used in <a href="#">GaussRF</a> to simulate the paths, the second component gives the register for the drops.</p>
old.paths	<p>NULL or list. If list it must have the same format as the return list of flowpattern. Then the flow pattern is recalculated where only the drop.distr is changed.</p>
PrintLevel	<ul style="list-style-type: none"> <li>• <math>\leq 0</math> : no messages are printed</li> <li>• <math>\geq 1</math> : the higher the number the more information is given</li> </ul>

wait	logical. If PrintLevel>1 than the simulated profile is shown. If wait=TRUE the system waits for return after plotting.
raw	logical. If grid=TRUE then the simulated random fields for the x- and y-coordinates of the paths are also returned. Ignored otherwise.
compress	logical. If grid=FALSE, then paths where no part is within the considered slice are deleted at an early stage. Ignored otherwise.
max.points	logical. If grid=FALSE, type %in% c('dependent', 'all') and compress=TRUE, then the simulation of the the coordinates of the paths is done in several steps and “compression” is performed between the steps.

## Details

type allows for the following values

'identical' All paths are identical, the increments of the path curve in x-direction are given by a Gaussian random field with parameters x.name x.var, x.v.scale and x.kappa. The increments in y-direction by an independent Gaussian random field with parameters y.name y.var, y.v.scale and y.kappa. The drops are i.i.d. according to drop.distr.

'unif' For each path the angles of the segments to the horizontal plane are constant, such that the length of the paths are uniformly distributed  $\sim U([1, \text{unif.b}]) * [\text{units of the vertical axis}]$ . The paths are varied only in the x-direction, but not in the y-direction.

The drops are i.i.d.

'independent' The paths and the drops are independently distributed.

'dependent' The paths increments in x-direction are spatially dependent given by a Gaussian random field with parameters x.name x.var, x.h.scale and x.kappa. Indeed, a three dimensional random field of geometrical anisotropy is generated, which includes the vertical direction with parameter x.v.scale. Analogously for the increments in y-direction. The x-y-coordinates of the sampling points of the random field are the starting points of the paths at the surface.

The drops are i.i.d. according to drop.distr.

'all' The paths are simulated as for type='dependent'. Here the drop distributions are spatially dependent, given by a two-dimensional Gaussian random field with parameters drop.name, drop.scale and drop.kappa.

See the references for some background.

## Value

it returns the following list

c.r	empty list if length(method)==0. Otherwise a list with length(method) components. Each of them is the output of <a href="#">risk.index</a> .
intermediate	list containing xx, a matrix of x-coordinates of the vertices of the paths, yy, a matrix of y-coordinates of the vertices of the paths, and LEN, the field of all the stained path lengths, after marginal transformation to uniform $U([0, 1])$ distribution.
dist	1:depth



---

getactions	<i>Get input behaviour</i>
------------	----------------------------

---

### Description

The functions return values stored by [useraction](#), [Locator](#) and [Readline](#)

### Usage

```
getactions()  
getactionlist()
```

### Value

`getactionlist` returns a list of stored values created by [Locator](#) and [Readline](#)

`getactions` returns a list of all parameters that influence the behaviour of [Locator](#) and [Readline](#).

### Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

### See Also

[Locator](#), [Readline](#), [useraction](#),

### Examples

```
## see useraction
```

---

Locator	<i>Graphical Input</i>
---------	------------------------

---

### Description

Reads the position of the graphics cursor when the (first) mouse button is pressed, or replays from storage.

### Usage

```
Locator(n, type="n", info=NULL, ...)
```

**Arguments**

n	the maximum number of points to locate; the value of n is ignored if
type	One of "n", "p", "l" or "o". If "p" or "o" the points are plotted; if "l" or "o" they are joined by lines.
info	arbitrary object; tracing information that is useful in case the user likes to edit the stored sequence of inputs; the value of info is considered only if <code>useraction</code> has been called by <code>action="start.register"</code> or <code>action="continue.register"</code>
...	additional graphics parameters

**Details**

The behaviour of Locator depends on the the value of action set by `useraction`, see there for more information.

See also `locator` for further information on the parameters.

**Value**

A list containing 'x' and 'y' components which are the coordinates of the identified points in the user coordinate system, i.e., the one specified by `'par("usr")'`.

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/institute>

**See Also**

`getactions`, `Readline`, `useraction`, `userinput`

**Examples**

```
## see useraction
```

---

modify.horizons

*Stochastic variation of the definitions of the horizons*

---

**Description**

The function modifies the geometric definition of the horizons stochastically

**Usage**

```
modify.horizons(h, percent=5, level.percent=5, rdistr=rnorm)
```

**Arguments**

h	list of the same format as the output of <a href="#">xswms2d</a>
percent	magnitude of variation of a single point; see Details.
level.percent	magnitude of the length of the translation vector for the whole object; see Details.
rdistr	distribution for the translation. The function must have two parameters: the first parameter gives the number of random variables to generate; the other parameter is called by name, 's', and should determine (or be related to) the standard deviation of the distribution.

**Details**

The function `modify.horizons` changes stochastically the location of the vertices of the defining segments for the boundary of a horizon. To this end, the standard deviations  $s_x$  and  $s_y$  for the x coordinates and the y coordinates of the vertices are calculated. Then the `s` parameter for the distribution of the translation distance equals  $s_x \cdot \text{percent}/100$  for the x coordinates (analogously for the y coordinates).

- For genuine horizons there is also a shift of all the points in vertical direction by the distribution `rdistr(,s)` where `s` is the product of the total length of the grid in y direction and `level.percent` divided by 100 times the number of genuine horizons. (Without the division by the number of genuine horizons the user would have to pay much more attention that the horizons do not awefully overlap.) Further, if the first point is right to the second or the last point is left to the last but one point, the x coordinate of the first and/or the second point is replaced by the value of the neighbouring point minus or plus a random value that is uniformly distributed on  $[0, s_x]$ . Finally, if the x coordinate of the first or the last point is within the grid than points on the border of the grid are added.
- If it is a polygon, then the whole figure is additionally shifted to the right and top according to the given distribution where `s` equals (the range of the respective coordinate) \* `level.percent` / (100 \* (the number of polygons)).

`modify.horizons` calls [calculate.horizons](#) before returning the list.

**Value**

List of the same format as the output of [xswms2d](#).

`modify.horizons` calls [calculate.horizons](#) before returning the list.

**Author(s)**

Martin Schlather, <[martin.schlather@math.uni-goettingen.de](mailto:martin.schlather@math.uni-goettingen.de)> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**See Also**

[SoPhy](#), [xswms2d](#)

**Examples**

```

h <- xswms2d(xlim=c(1, 100), ylim=c(1, 100), step=1, new=NULL)
##          new=NULL: xswms2d returns the standard definition for h
##          without entering the interactive surface
h$n <- 2          ## define a second horizon
h$H2 <- h$H1
h$H2$type <- "H" ## genuine horizon, not polygon

## coordinates of the boundary segments between the horizons
h$H2$points <- list(x=seq(1, 100, 5))          ## x coordinates
h$H2$points$y <- 40 + 20 * cos(h$H2$points$x / 5) ## y coordinates

h <- calculate.horizons(h) ## update the internal parameters
draw.horizons(h)          ## plot the horizons
for (i in 1:20) {
  readline("Press return")
  h <- modify.horizons(h) ## stochastic modification of the horizons
  draw.horizons(h)
}

```

---

my.legend

*legend for a spectrum of colours*


---

**Description**

the function plots a legend for a spectrum of colours

**Usage**

```
my.legend(lb.x=0, lb.y=0, zlim, col, cex=0.8, y.intersp=0.02, bg='white')
```

**Arguments**

lb.x	x-coordinate of the left bottom corner of the legend box
lb.y	y-coordinate of the left bottom corner of the legend box
zlim	value range for the colours
col	spectrum of colours, obtained by the various <a href="#">palettes</a> .
cex	letter size increase
y.intersp	the space between the small colour lines; this value might be changed if the size of the legend box is not appropriate.
bg	background colour

**Value**

NULL

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**Examples**

```
## see \link{read.swms2d.table}, for example
```

---

Pareto *The Pareto Distribution*

---

**Description**

Density, distribution function, quantile function and random generation for the Pareto distribution with form parameter  $\xi$  and scale parameter  $s$

**Usage**

```
ppareto(q, xi, s=1, lower.tail=TRUE)
```

```
dpareto(x, xi, s=1)
```

```
qpareto(p, xi, s=1, lower.tail=TRUE)
```

```
rpareto(n, xi, s=1)
```

**Arguments**

<code>xi</code>	form parameter of the Pareto distribution
<code>s</code>	scale parameter of the Pareto distribution
<code>lower.tail</code>	logical; if TRUE (default), probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$
<code>q</code>	vector of quantiles
<code>x</code>	vector of quantiles
<code>p</code>	vector of probabilities
<code>n</code>	number of observations. If <code>length(n) &gt; 1</code> , the length is taken to be the number required.

**Value**

`dpareto` gives the density, `ppareto` gives the distribution function, `qpareto` gives the quantile function, and `rpareto` generates random deviates.

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

## References

Embrechts, P., Klueppelberg, C. and Mikosch, T. (1997) *Modelling Extremal Events*. Springer: Berlin.

---

plotFlow	<i>plotting of the simulated infiltration pattern</i>
----------	---

---

## Description

plotFlow2d plots two dimensional profiles of simulated stochastic flow patterns; plotFlow3d plots a 3d, perspective figure of simulated stochastic flow patterns.

## Usage

```
plotFlow2d(coord, pointradius=1, slice=2 * pointradius,
           full.size=TRUE, Profiles=1, dev=1, ps="",
           height=4, unit="cm", cex=2, correction=1.2, col=1,
           rl = function(x) readline(paste(x, ": press return")))
```

```
plotFlow3d(paths, horizons=c("no", "absorbing", "breakthrough"),
           drop.distr, n.balls=1, pointradius=1,
           dev=1, ps="3d.dye.pattern", ps.background=FALSE,
           profileheight=4, unit="cm", unit.scale=1, inf, sun,
           rl = function(x) readline(paste(x, ": press return")),
           low.resolution=TRUE,
           col=grey(pmin(1, pmax(0, seq(0.95, 0, -0.001)))))
```

## Arguments

coord	output of plotFlow3d: $n \times 3$ matrix; 3d coordinates for the centre of the balls that approximate the tube around each path
pointradius	radius of the dyed tube around each path; the tube is approximated by a sequence of balls
slice	thickness of the slice for which the paths are visible
full.size	logical. If FALSE then the projected radii of the sliced balls are plotted. If TRUE then instead of the projected radius the pointradius is used.
Profiles	number of the in the y-direction equally spaced profiles of the 3d flow pattern
dev	plotting device, see <a href="#">Dev</a>
ps	name of the postscript file, see <a href="#">Dev</a>
height	height of the plotted figure, see <a href="#">Dev</a>
unit	string; if empty then labels are not given. Otherwise unit denotes the units of the plot
cex	used for cex.axis and cex.lab, see <a href="#">par</a>

correction	positive real number. Do not change its value. This constant takes into account that the plotting procedure used in the function does not plot a circle of exactly the given radius.
col	colour of the points; plotFlow3d needs a vector of colours starting with the colour of the nearest points
paths	list as returned by <a href="#">flowpattern</a>
horizons	<p>Temptative parameter. If "absorbing" the path will be 'infinitely' long in a layer at depth 50 about, so that fluxes will stop in this layer. If "breakthrough" the path will be very long in a layer at depth 50 about, so that only a few paths are percolated.</p> <p>The values "absorbing" and "breakthrough" are used to create the figures in the Discussion of Schlather and Huwe (2004a).</p>
drop.distr	function or missing. The stained path length reached by a single drop is calculated as follows. First a Gaussian random field is simulated with the parameters given below. Then the Gaussian random field is marginally transformed to uniform distribution by the Gaussian distribution function. Then the inverse of the required distribution function is applied. drop.distr gives this inverse. It should allow for vectors as input variables. The drop.distr used to create paths is overwritten if drop.distr is not missing, but the realisation of the paths is always kept.
n.balls	the tube for each path segment is approximated by n.balls balls
ps.background	logical. If dev=TRUE then a postscript file is created, which is, in general, huge. Therefore, any postscript file created on a unix system is transformed into a tiff-file by 'convert' and back again to a postscript file. These transformation are time consuming. If ps.background=TRUE then these transformations are started in the background.
profileheight	essentially the height of the figure; the value is used to calculate the width of the plot. The final height of the plot is higher than profileheight because additional space needed for the perspective plot
unit.scale	scaling factor for the numerical values of all three-dimensional coordinates
inf	Vector of three components. The first two give the position of the infinitely far point; the third component gives the extension factor for the second dimension that is orthogonal to the screen; inf is of order 1/100
sun	the three dimensional coordinates of the fictive sun
r1	function. Called, after a figure, but the last, is plot on the screen.
low.resolution	logical. If TRUE and dev=FALSE, i.e. postscript file will be produced, then on unix systems the postscript file will be rewritten to have much smaller size, but also lower resolution.

### Value

plotFlow2d returns NULL. plotFlow3d returns invisibly a matrix for the centers of the approximating balls

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**References**

- Schlather, M. and Huwe, B. (2005a) A stochastic model for 3-dimensional flow patterns in dye tracer experiments. *J. Hydrol.* **In press**, .
- Schlather, M. and Huwe, B. (2005b) A risk index for characterising flow pattern in soils using dye tracer distributions *Submitted to J. Contam. Hydrol.*

**See Also**

[flowpattern](#)

**Examples**

```
## see \link{flowpattern}
```

---

plotRF

*Plotting random fields*

---

**Description**

the function plots random fields including stones and roots using the definition of the horizons in [xswms2d](#).

**Usage**

```
plotRF(h, col.txt='black', col.stones='white',
       col.rf=if (is.null(h$col.rf)) rainbow(100) else h$col.rf,
       titl=TRUE, line = 0.7, lim=1, quadratic=TRUE,
       cex=1, cex.leg=0.8, pch=5,
       cex.pch = max(0.05, 3/min(length(h$grid.x), length(h$grid.y))),
       root.col=grey(1 - (i-1) / (1.6 * length(h$plants))),
       what=c('Root.RF', 'Stone.RF', 'RF'),
       transf=c("K", "H", "theta", "none"), legend = TRUE, ylim, zlim)
```

**Arguments**

h	a list of the same format as the output of <a href="#">xswms2d</a>
col.txt	colour of the title
col.stones	colour of the stones and the air
col.rf	colour spectrum of the random field; plants are always grey
titl	logical. If TRUE a title is plotted using the colour col.txt

line	place where the title is plotted, see <a href="#">title</a> .
lim	value in $[0, 1]$ that is used only if <code>zlim</code> is missing; quantile for the upper bound <code>zlim[2]</code> ; the value of <code>lim</code> should be less than 1 if the random field contains some extreme values, since the colour scale is linear
quadratic	logical. If TRUE the figure matrix is enlarged and filled with NA symmetrically in x-direction or on the bottom in y-direction such the matrix (and the figure) become quadratic
cex	A numerical value giving the amount by which the title text, the axes and the labels of the axes should be scaled relative to the default
cex.leg	A numerical value giving the amount by which the legend text should be scaled relative to the default
pch	the symbol for a root segment
cex.pch	size of the symbol
root.col	the colours for a root segments
what	chooses between the random field that includes modifications by roots and stones or by stones alone, or the random field without modifications
transf	Kind of random field: saturated hydraulic conductivity "K", pressure head "H", the water content "theta", or the original Gaussian random field ("none")
legend	logical. If (TRUE) a legend is added.
ylim	missing or vector of two components. If missing the range of the simulation is used.
zlim	missing or vector of two components. Values below <code>zlim[1]</code> are plotted in white, values above <code>zlim[2]</code> are plotted also in white. If missing then <code>zlim[1]</code> is the minimal value and <code>zlim[2]</code> is calculated by means of <code>lim</code> .

### Details

If `transf="K"` then the function plots the random field of hydraulic conductivity, i.e., the Gaussian random field after `h$millier.link` and `h$millierK` have been applied.

### Value

the matrix of the random field values, always excluding the plants, if no error has occurred and an error message otherwise.

### Author(s)

Martin Schlather, <[martin.schlather@math.uni-goettingen.de](mailto:martin.schlather@math.uni-goettingen.de)> <http://www.stochastik.math.uni-goettingen.de/~schlather>

### See Also

[plotWater](#), [xswms2d](#)

### Examples

```
## see \link{simulateHorizons}
```

---

`plotRGB`*Plotting RGB figures*

---

**Description**

The function plots an RGB array

**Usage**

```
plotRGB(picture, x=1:dp[1], y=if (reverse) dp[2]:1 else 1:dp[2],
        reverse=TRUE, cex.axis=1, ...)
```

**Arguments**

<code>picture</code>	three dimensional array where the third dimension must have length 3 or 4; the fourth component is ignored if there is any
<code>x</code>	x-coordinates; the length must match the length of the first dimension of <code>picture</code>
<code>y</code>	y-coordinates; the length must match the length of the first dimension of <code>picture</code> ; see details
<code>reverse</code>	logical; see details
<code>cex.axis</code>	graphical parameter, see <a href="#">par</a>
<code>...</code>	additional graphical parameters for <a href="#">image</a>

**Details**

`reverse=TRUE` is for the plot of soil profiles, where the numbering of the y-axis start at the top and is counted positively downwards.

`reverse=FALSE` The usual presentation of the vertical axis.

The functions checks whether y is in decreasing or increasing order, and plots the y axis accordingly.

**Value**

NULL

**Author(s)**

Martin Schlather, <[martin.schlather@math.uni-goettingen.de](mailto:martin.schlather@math.uni-goettingen.de)> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**See Also**

[read.picture](#), [write.picture](#)

**Examples**

```
fig <- read.picture(paste(system.file(package='SoPhy'),
                          'tracer', 'K06', sep="/"))
plotRGB(fig)
```

---

plotWater

*Plotting the result of SWMS2D*


---

**Description**

the function plots the H, Q,  $\theta$ , vx, vz, or conc, stored in a list as in the output of SWMS2D

**Usage**

```
plotWater(h, instance,
          what=c("H", "Q", "theta", "vx", "vz", "Conc", "logH"),
          col.txt="black",
          col.simu=if (is.null(h$col.simu)) rainbow(100) else
            if (what.nr == 7) rev(h$col.simu) else h$col.simu,
          col.exception = c("yellow", "darkred"), lim=1,
          titl=TRUE, line=0.2, quadratic=TRUE, cex=1, cex.leg=0.8,
          legend = TRUE, ylim, zlim)
```

**Arguments**

h	a list as returned by <a href="#">swms2d</a> or a list as returned by <a href="#">xswms2d</a> ; usually <a href="#">create.waterflow</a> has been called beforehand.
instance	the number of the time point for which the data should be plotted. If missing or NULL the last one is taken.
what	character. what gives the parameter to be plotted: H water potential $H$ , Q discharge/recharge rates $Q$ for internal sink/sources, theta water contents $\theta$ , vx x-components of the Darcian flux vector vx, vz z-components of the Darcian flux vector vz, Conc solute concentration logH log of $-H$ , The parameter what allows also the numbers 1, . . . , 6 corresponding to "H", . . . , "logH".
col.txt	colour of the title
col.simu	colour spectrum of the simulated field of H, Q, $\theta$ , vx, vz, or conc; areas of stones or air are white.

<code>col.exception</code>	vector of 2 components; colour plotted values that are below or above the given range, see <code>zlim</code> ; for <code>what="logH"</code> and positive water potential, <code>col.exception[1]</code> is plotted.
<code>lim</code>	value in $[0, 1]$ that is used only if <code>zlim</code> is missing or NULL; the range <code>zlim</code> of the plotted values depends on what. Let $q(p)$ be the $p$ -quantile of the values. Then <ul style="list-style-type: none"> <li>• <code>H</code>: <code>zlim</code> = <math>[q(1 - \text{lim}); 0]</math></li> <li>• <code>theta</code>: <code>zlim</code> = <math>\text{range}(\{0, \theta_{s,i}\}, \text{max}(h\\$hQThF1C))</math>, where the <math>i</math> runs over all horizons and polygons. That is, the range is independent of the value of <code>lim</code>. See <code>xswms2d</code> for <math>\theta_s</math>.</li> <li>• <code>Conc</code>: <code>zlim</code> = <math>[0, q(\text{lim})]</math></li> <li>• <code>logH</code>: <code>zlim</code> = <math>[q(1 - \text{lim}), 1]</math></li> <li>• otherwise: <code>zlim</code> = <math>[q(1 - \text{lim}), q(\text{lim})]</math></li> </ul> <p>quantile for the upper bound of the values of the random field; this variable should be less than 1 if the random field contains some extreme values, since the colour scale is linear</p>
<code>titl</code>	logical or character. If TRUE a title is plotted using the colour <code>col.txt</code>
<code>line</code>	parameter of function <code>title</code>
<code>quadratic</code>	logical. If TRUE the figure matrix is enlarged and filled with NA symmetrically in x-direction or on the bottom in y-direction such the matrix (and the figure) become quadratic
<code>cex</code>	A numerical value giving the amount by which the title text, the points indicating the root segments, the axes and labels of the axes should be scaled relative to the default
<code>cex.leg</code>	A numerical value giving the amount by which the legend text should be scaled relative to the default
<code>legend</code>	logical. If TRUE a legend is added.
<code>yylim</code>	missing or vector of two components. If missing the range of the simulation is used.
<code>zlim</code>	missing or vector of two components. Values below <code>zlim[1]</code> are plotted in <code>col.exception[1]</code> , values above <code>zlim[2]</code> are plotted in <code>col.exception[2]</code> . If missing or NULL the vector <code>zlim</code> is calculated by means of <code>lim</code> .

### Details

For all variables the plot is over the whole range except for the following cases. If `theta` is plotted, the minimum is 0 and the maximum the maximal field capacity, If `H` is plotted, the maximum is at most 0.

### Value

matrix of the plotted values if no error has occurred and an error message otherwise.

### Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**See Also**

[plotRF](#), [xswms2d](#)

**Examples**

```
## see \link{create.waterflow}
```

---

Quader

*Perspective plot*

---

**Description**

Simple functions that plot points in a perspective view

**Usage**

```
belichtet(x, radius=1, fix.scale=TRUE, correction=1.2)
```

```
pos3D(x, inf)
```

```
quader(size, inf, sun, bottomleftfront= c(0,0,0),
        col=grey(seq(1,0,-0.001)), col.frame=c("grey", "black"), lty=c(2,1),
        cex.axis=1.5, reverse=TRUE, unit = "cm", add=FALSE, plot=TRUE,
        font = c("sans serif", "bold"), srt=NULL)
```

**Arguments**

<code>x</code>	matrix of three rows; the three dimensional coordinates
<code>radius</code>	radius of the dyed tube around each path; the tube is approximated by a sequence of balls
<code>fix.scale</code>	logical. If FALSE the plotted radius is modified according to the perspective distance
<code>correction</code>	positive real number. Do not change its value. This constant takes into account that the plotting procedure used in the function does not plot a circle of exactly the given radius.
<code>inf</code>	Vector of three components. The first two give the position of the infinitely far point; the third component gives the extension factor for the second dimension that is orthogonal to the screen; <code>inf</code> is of order 1/100
<code>size</code>	dimension of the quader
<code>sun</code>	the three dimensional coordinates of the fictive sun
<code>bottomleftfront</code>	location of the quader; the standard is that the quader is in the first quadrant with one vertice in the origin

<code>col</code>	vector of colours; the point that is nearest to the sun gets the colour <code>col[1]</code> , the one that is the furthest away <code>rev(col)[1]</code> .
<code>col.frame</code>	vector of two components. The edges that are visible get colour <code>col.frame[2]</code> the others <code>col.frame[1]</code>
<code>lty</code>	vector of two components. the edges that are visible have line type <code>lty[2]</code> the others <code>lty[1]</code>
<code>cex.axis</code>	The magnification to be used for axis annotation relative to the current, see <a href="#">par</a>
<code>reverse</code>	logical. If TRUE the z-axis is numbered upside down
<code>unit</code>	unit
<code>add</code>	logical. If TRUE, a quader is added to current figure
<code>plot</code>	logical. If FALSE only the values for the coordinate system are calculated and stored in the background, which will be used by <code>belichtet</code> , for example.
<code>font</code>	font used for the labelling of the axes, see also <code>par("font")</code>
<code>srt</code>	angle for the character rotation when labelling the y axis; if NULL the angle is calculated, but might be not always the best choice.

### Details

The function `quader` must be called first with `add=FALSE`. `quader` plots a perspective quader and provides global variables whose name starts with `.p3d`. If, for the second time, `quader` is called solely by the argument `add=TRUE`, the original quader is redrawn.

The auxiliary function `pos3D` calculates the two-dimensional projected positions of the three dimensional points. If `inf` is not given, the one given in `quader` is used.

`belichtet` plots the three dimensional points in perspective view.

### Value

The functions `belichtet` and `quader` return NULL, the function `pos3D` returns a matrix of 2 rows and `ncol(x)` columns.

### Author(s)

Martin Schlather, <[martin.schlather@math.uni-goettingen.de](mailto:martin.schlather@math.uni-goettingen.de)> <http://www.stochastik.math.uni-goettingen.de/~schlather>

### Examples

```
f <- function(x)
  cos((x[2,] + x[1, ]) / 5) + cos((x[2, ]-x[3,]) / pi + 3) +
  cos(x[3, ] / 2 + 2)
x <- t(expand.grid(1:50, 1:100, 1:50))
y <- x[, f(x)>0.5]
quader(size=c(50, 100, 50), inf=c(150, -400, 1/500),
       sun=c(200, -30000, 50))
belichtet(y)
quader(add=TRUE, col.frame=c("transparent", "black"))
```

---

read.picture	<i>Reading TIF pictures and conversion to RGB matrix</i>
--------------	--

---

### Description

The function reads in TIF formatted pictures and returns an RGB-A matrix, where A is the associated alpha matting information, if there is any (and identically 255 otherwise). If the format is different from TIF, it is first convert-ed to TIF.

### Usage

```
read.picture(picture, extensions=c("tif", "tiff",
    if (.Platform$OS.type=="unix") c("gif", "jpeg")),
    PrintLevel=RFparameters()$Print, tmp.dir=".")
```

### Arguments

picture	name of the graphic file, see details
extensions	standard extensions for the graphic file
PrintLevel	<ul style="list-style-type: none"> <li>• <math>\leq 0</math> : no messages are printed</li> <li>• <math>&gt; 0</math> : the higher the number the more information is given.</li> </ul>
tmp.dir	NULL or character. The variable is used if the picture file does not end with 'tif' or 'tiff'. Then the picture file is <i>converted</i> into a tif file before being read in. The storage path is given by tmp.dir. An existing tif file in tmp.dir is not overwritten, but this file used instead and a warning is given.

### Details

The procedure of finding the graphic is the following.

- if the file is given with one of the standard extension, it is assumed that this file is wanted
- else it is assumed that only the base name was given. Then, as above, the standard extensions are added sequentially.
- else the algorithm fails

If the file is neither TIF nor TIFF it is first convert-ed.

### Value

three-dimensional array, where the first two dimensions are the dimensions of the graphic, and the third dimension has length 4, for red, green, blue and alpha.

### Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**See Also**

[plotRGB](#), [write.picture](#)

**Examples**

```
fig <- read.picture(paste(system.file(package='SoPhy'),
                          'tracer', 'K06', sep="/"), Pr=3)
plotRGB(fig)
```

---

read.swms2d.table	<i>Standard SWMS2D input files</i>
-------------------	------------------------------------

---

**Description**

This function reads in the standard SWMS2D input files

**Usage**

```
read.swms2d.table(path, selct.in = "SELECTOR.IN",
                  grid.in = "GRID.IN", atm.in = "ATMOSPH.IN")
```

**Arguments**

path	directory of the input files
selct.in	file name for general input data, see the manuscript of SWMS2D
grid.in	file name for information about the finite elements
atm.in	file name for atmospheric data

**Value**

the function returns a list of the following components; see the SWMS2D manual for details

Units	c(LUnits, TUnits, MUnits, BUnits), units of length, time, mass.
Kat	type of flow system; 0:horizontal, 1:axisymmetric; 2:vertical.
MaxIt	maximum number of iteration during any time step.
TolTh	maximum desired change of water content.
TolH	maximum desired change of pressure head.
lWat	logical. If TRUE transient water flow else steady state.
FreeD	logical. TRUE if free drainage at the bottom.
NLay	Number of subregions for water balances.
hTab	c(hTab1, hTabN), interval of pressure heads within which a table of hydraulic properties is generated.
Par	vector of 9 elements or matrix of 9 columns, each row representing a different kind of material.

dt	initial time increment.
dtMinMax	c(dtmin, dtmax), minimum and maximum permitted time increment.
DMu1	c(dMu1, dMu12), $dMu1 \geq 1$ , $dMu12 \leq 1$ ; if the number of required iterations is less than 4 or greater than 6 then the next time step is multiplied by dMu1 and dMu12, respectively.
TPrint	vector of increasing print-times.
NP	matrix of one or two rows and arbitrary columns; seepage information: each row defines a seepage face, the columns give the node numbers (or are NA).
DrCorr	reduction factor for drainage; see the SWMS2d manual
ND	vector of up to two elements given the global node number of the drain.
EfDim	(length{ND} x 2)-matrix drain information: first column gives the effective diameter; second column gives the dimension of the square in the finite element mesh.
KE1Dr	matrix of length(ND) rows and arbitrary columns; drainage information: each row defines the surrounding finite elements of the drain; values are either the global finite element numbers or NA.
Epsi	temporal weighing coefficient: 0: explicit scheme, 0.5:Crank-Nicholson, 1:fully implicit.
lUpW	logical. Upstream weighing formulation used if TRUE else Galerkin formulation.
lArtD	logical. TRUE if artificial dispersion is to be added to fulfill the stability criterion PeCr.
PeCr	Stability criterion, see the SWMS2d manual; zero if lUpW=TRUE.
ChPar	vector of 9 element if Par is a vector; other wise (nrow(Par) x 9)-matrix; chemical material properties: bulk density, ionic diffusion coefficient in free water, longitudinal dispersivity, transverse dispersivity, Freundlich isotherm coefficient, first-order rate constant for dissolved phase, first-order rate constant for solid phase, zero-order rate constant for dissolved phase, zero-order rate constant for solid phase.
KodCB	vector of sum(nCodeM[, 2]!=0) elements, defining the chemical boundary conditions, see the SWMS2d manual.
cBound	vector of six elements: jth element gives the concentration for boundary nodes i with chemical boundary condition abs(KodCB[i])==j.
tPuls	time duration of the concentration pulse.
nCodeM	matrix of 12 columns and arbitrary number of rows; nCodeM gives the nodal information; the 12 columns contain: (1) nodal number, (2) code giving the boundary condition (see the SWMS2D manual), (3) x-coordinate, (4) z-coordinate, (5) initial pressure head, (6) initial concentration, (7) prescribed recharge or discharge rate, (8) material number, (9) water uptake distribution value, (10) pressure head scaling factor, (11) conductivity scaling factor, (12) water content scaling factor.
KXR	matrix of 8 columns and arbitrary rows; gives the finite element information; (1)-(4) give the nodes in counter-clockwise order (last node is repeated if element is a triangle); (5)-(7) give the anisotropy parameters of the conductivity tensor (angle, first and second principal component), (8) subregion number.

Width	vector of $\text{sum}(\text{nCodeM}[,2] \neq 0)$ elements; width of the boundary associated with the boundary nodes (in the order given by nCodeM).
rLen	width of soil surface associated with transpiration.
Node	vector of observation nodes for which information is collected at each time level.
SinkF	logical; water extraction from the root zone if TRUE.
qGWLf	logical; If TRUE then the discharge-groundwater level relationship is used, see the SWMS2d manual.
GWL0L	reference position of groundwater table (usually the z-coordinate of the soil surface).
Aqh	parameter in the discharge-groundwater level relationship.
Bqh	second parameter in the discharge-groundwater level relationship.
tInit	starting time of the simulation.
hCritS	maximum allowed pressure head at soil surface.
atmosphere	matrix of 10 columns and arbitrary rows; each row presents the atmospherical conditions at a certain instance; the columns are (1) the instance at which the time period ends (2) precipitation, (3) solute concentration of rainfall water, (4) potential evaporation rate, (5) potential transpiration rate, (6) absolute value of minimum allowed pressure head at the soil surface, (7) drainage flux across the bottom boundary (for $\text{abs}(\text{nCodeM}[,2]) = 3$ ), (8) ground water level, (9) concentration of drainage flux (for $(\text{abs}(\text{nCodeM}[,2]) = 3) \ \& \ (\text{KodCB} < 0)$ ) (10) concentration of drainage flux (for $(\text{abs}(\text{nCodeM}[,2]) = 3) \ \& \ (\text{KodCB} > 0)$ ).
root	c(P0, P2H, P2L, P3, r2H, r2L) information about water uptake by roots, see the SWMS2D manual.
POptm	scalar if Par is a vector, a vector of $\text{nrow}\{\text{Par}\}$ elements otherwise; pressure heads, below which roots start to extract water at maximum possible rate.
path	the input parameter.
selct.in	the input parameter.
grid.in	the input parameter.
atm.in	the input parameter.

### Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

### References

- Schlather, M. and Huwe, B. (2004) The use of the language interface of R: two examples for modelling water flux and solute transport. *Computers \& Geosciences* **30**, 197-201.
- Simunek, J., Vogel, T., and van Genuchten, M.Th. (1994) *The SWMS2D code for simulating water flow and solute transport in two-dimensional variably saturated media, Version 1.21*. Research Report No. 132, 197 p., U.S. Salinity Laboratory, USDA, ARS, Riverside, California.

**See Also**[swms2d](#)**Examples**

```
#####
##  SWMS2D, Example 1
#####

path <- paste(system.file(package='SoPhy'), 'swms2d', sep="/")
x <- read.swms2d.table(path)
x$TPrint <- seq(10, 5400, 10)
par(cex=1, mar=c(4.2,4,0.2,0.2))

z <- swms2d(x)$hQ[3, , ]
i <- ((length(x$nCodeM$z) / 2):1) * 2 - 1
image(x=c(0, x$TPrint) / 60, y=x$nCodeM$z[i] - max(x$nCodeM$z),
      z=t(z[i, ]), xlab='time [min]', ylab='z [cm]',
      col=grey(seq(1, 0.15, -0.01))), cex.lab=1.5, cex.axis=1.5)

my.legend(0, -max(x$nC$z[i]), zlim=range(z), y.i=0.02,
          col=grey(seq(1,0.15,-0.01))), cex=1.5)
```

---

**Readline***Read a Line*

---

**Description**

Readline reads a line from the terminal or from storage

**Usage**

```
Readline(prompt="", info=NULL)
```

**Arguments**

prompt	the string printed when prompting the user for input. Should usually end with a space " "
info	arbitrary object; tracing information that is useful in case the user likes to edit the stored sequence of inputs; the value of info is considered only if <code>useraction</code> has been called by <code>action="start.register"</code> or <code>action="continue.register"</code>

**Details**

The behaviour of Readline depends on the the value of action set by [useraction](#), see there for more information.

The prompt string will be truncated to a maximum allowed length, normally 256 chars (but can be changed in the source code), since the function is based on [readline](#).

**Value**

A character vector of length one.

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**See Also**

[getactions](#), [Locator](#), [readline](#) [useraction](#), [userinput](#),

**Examples**

```
## see useraction
```

---

risk.index

*Risk index for soil leaching*

---

**Description**

Estimation of the index of potential risk to groundwater

**Usage**

```
risk.index(data, selected.dist=0.95,
           selected.rate=cbind(c(0.5, 0.8), c(0.4, 0.9), c(0.3, 1.0)),
           weights=1, measure=function(x) x^2,
           method=c('fix.m', 'optim.m', 'ml'),
           min.neg.xi = -10, max.neg.xi = -0.1, max.pos.xi = 10,
           endpoint.tolerance = 0, front.factor = 2,
           # min.no.paths=max(data[, 2]),
           max.no.paths=10 * max(data[, 2]),
           PrintLevel=RFparameters()$Print, max.rate=TRUE)
```

**Arguments**

<code>data</code>	matrix of two columns. First column gives the distances (depths in the profile measured from the surface) and the second column the number of observed blue pixels.
<code>selected.dist</code>	scale or vector with values in $(0, 1)$ or a vector of integers. Distances for which the form parameter of the pareto distribution is estimated; see Details. If <code>selected.dist</code> is a number in $(0, 1)$ the distances are $1 : (1 + \text{round}(\max(\text{data}[, 1] - 1) * \text{selected.dist}))$ . If it is a vector with values in $(0, 1)$ , then the vector must have an even number of elements and pairs of elements are interpreted as intervals. Otherwise the integers are interpreted as indices for data.
<code>selected.rate</code>	vector of matrix of nrow. additionally to the indices given by <code>selected.dist</code> , the form parameter is estimated also for those distances where the corresponding number of observed relative stained pixels (w.r.t. to the maximum number of observed pixels) is within the interval given by the first column of <code>selected.rate</code> . The risk index is calculated as the median of the estimated form parameters. In case no values are in the given interval of the first column the second column is considered etc, i.e. the first row should contain decreasing values and the second row increasing values.
<code>weights</code>	the estimation algorithm is based on a weighted least square algorithm; <code>weights</code> is usually either 1 or a vector of length <code>nrow(data)</code> .
<code>measure</code>	instead of the default least squares another distance function can be given.
<code>method</code>	the number of observed paths is a free parameter when fitting the Pareto distribution. It can either be set as the maximum number of stained pixels for the currently considered distances or depths ( <code>'fix.m'</code> ) or fitted within the optimisation algorithm ( <code>'optim.m'</code> ). Usually, it is not worth using the slower <code>'optim.m'</code> option. See also the Details.
<code>max.neg.xi</code>	optimisation parameter : largest negative value that is allowed as shape parameter of the Pareto distribution, i.e. a negative value close to 0.
<code>min.neg.xi</code>	optimisation parameter : smallest negative value that is allowed as shape parameter of the Pareto distribution
<code>max.pos.xi</code>	optimisation parameter : largest allowed shape parameter of the Pareto distribution
<code>endpoint.tolerance</code>	optimisation parameter. If the shape parameter is negative then the distribution has a finite upper endpoint. Hence, mathematically, the lowest upper end point of the Pareto distribution is given as the largest distance for which at least one stained pixel is observed. For stability reasons and because the observed data might be a scale mixture of Pareto distribution it is advantageous to allow for some tolerance of the minimal upper end point. If <code>endpoint.tolerance</code> is positive then the lower threshold for the upper end point is the largest distance for which the number of observed stained pixels is larger than <code>endpoint.tolerance</code> . If <code>endpoint.tolerance</code> is negative then the lower threshold equals largest distance for which at least one stained pixel is observed minus the modulus of <code>endpoint.tolerance</code> .

front.factor	optimisation parameter . The upper bound for the upper endpoint equals the front.factor times the largest distance for which at least one stained pixel is observed. The value should best not be changed.
max.no.paths	the number of paths is estimated as nuisance parameter when estimating the risk index; max.no.paths give the upper bound for the nuisance parameter in the optimisation.
PrintLevel	The higher the value of PrintLevel the more tracing information is given. Up to value 1, no information is given. Note that if PrintLevel $\geq$ 2 a running counter is shown that includes the printing of backspaces (^H). The backspaces may have undesirable interactions with some few other R functions, e.g. Sweave. See package RandomFields for the default option RFparameters() $\$$ Print.
max.rate	logical. If TRUE then the lines for which $m(D)/m(0)$ is in selected.rate are used to calculate the final risk index. Here $m(D)$ gives the maximum of $p(d)$ , $d = D, D + 1, \dots$ where $p(d)$ the number of stained pixels in depth $d$ . If FALSE then the criterion $m(D)/m(0)$ is replaced by $p(D)/p(0)$ .

### Details

Denote by  $f(d)$  the number of blue pixels registered at depth  $d$  (or distances from the soil surface). Then, the risk index is by definition a shape parameter of  $f(d)$  for large distances  $d$ . Since the term *large* cannot be defined precisely, the shape parameter is calculated for the function values  $f(d)$  for distances  $d \geq d_i$  and several fixed starting distances  $d_i$ . The distances  $d_i$  are given by selected.rate. (The approach is similar to that for analyzing extremal events.)

The selection criterion  $m(D)/m(0)$  is **always** based on method='fix.m', whatever method is chosen to estimate  $\xi$ .

### Value

list of the following components

par	matrix of estimated parameters; first row: risk index; second row: scale parameter; third row: estimated maximum number of paths $m(D)$ except $m(0)$ that is given by max.freq and is always set to the maximum number of pixels. forth row: $D$ (sel.dist).
data	the input data except for some reordering
weights	the input weights except for some reordering
selected.dist	the selected distances in form of indices (in clear text, in case they were given in form of a real value in $(0, 1)$ ).
selected.rate	range of the selected number of stained pixels
sel.rate	index set for the data where the observed number of stained pixels are within selected.rate
sel.dist	the index set containing selected.dist and sel.rate
max.freq	maximum number of observed stained pixels

values	the minimal least squares values
method	the input parameter method
measure	the input parameter measure
raw.risk.index	risk index calculated as median of the estimated form parameters for selected.rate
risk.index	the median is calculated only for values greater than 0.999min.neg.xi and less than 0.999max.pos.xi

### Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

### References

Embrechts, P., Klueppelberg, C. and Mikosch, T. (1997) *Modelling Extremal Events*. Berlin: Springer.

Schlather, M. and Huwe, B. (2005) A risk index for characterising flow pattern in soils using dye tracer distributions. *J. Contam. Hydrol.* **79**, to appear.

### See Also

[SoPhy](#), [analyse.profile](#), [xswms2d](#)

### Examples

```
sample.depth <- 1 : 100
d <- rexp(1000, 1/25)
freq <- numeric(length(sample.depth))
for (i in 1:length(sample.depth)) freq[i] <- sum(d>=sample.depth[i])
cr <- risk.index(cbind(sample.depth, freq),
                selected.rate=c(0.95, 0.9),
                endpoint.tolerance=20, method="fix.m")
cr$risk.index ## the true value is 0
```

---

sh

*Sample code*


---

### Description

The functions provide the code for the figures and simulations in the publications of Schlather and Huwe

**Usage**

```
sh.jch(input=NULL, dev=2, pspath=".", txt.result.dir = "txt/",
       simu.path="simu/", standard.method = "fix.m",
       final=TRUE, PrintLevel=0, bw=TRUE, readlines=TRUE)
sh.jh(input=NULL, dev=2, pspath=".", final=TRUE, PrintLevel=0,
      readlines=TRUE, low.resolution=TRUE, bw=TRUE)
```

**Arguments**

input	vector of values for automatic choice of the submenus
dev	device, see <a href="#">Dev</a>
pspath	path used for all postscript files in sh.jh and for all postscript files in sh.jch except those based on the extended simulation study
txt.result.dir	path used for non-graphic results
simu.path	path used for the postscript files based on the extended simulation study
standard.method	standard method for estimating the <a href="#">risk.index</a>
final	logical. If TRUE all the parameters are identical to those used in the papers; then at least 1GB of memory on a Linux machine is needed. If FALSE some few parameters are changed so that the simulations run faster and less memory is needed.
PrintLevel	If <code>PrintLevel &gt; 1</code> some information on the course of the algorithm is printed
bw	logical. if TRUE then all graphics will be printed in black and white, only.
readlines	logical. Only used if <code>is.numeric(dev)</code> . If <code>readlines=FALSE</code> then the system waits a second before the next plot is calculated or shown. Otherwise the system waits for return.
low.resolution	logical. Figures of low resolution are created. This takes time, but reduces size by about factor 10. This feature only works on unix/linux systems.

**Details**

Additionally to the figures in Schlather and Huwe (2005a,b), sh.jh allows for the simulation of profiles given in Crestana and Posadas (1998) and Schwartz et al. (1999).

**Value**

NULL

Depending on the value of dev, see [Dev](#), postscript or pdf files are created.

**Author(s)**

Martin Schlather, <[martin.schlather@math.uni-goettingen.de](mailto:martin.schlather@math.uni-goettingen.de)> <http://www.stochastik.math.uni-goettingen.de/~schlather>

## References

- Schlather, M. and Huwe, B. (2005a) A stochastic model for 3-dimensional flow patterns in dye tracer experiments. *J. Hydrol.* **310**, 17-27.
- Schlather, M. and Huwe, B. (2005b) A risk index for characterising flow pattern in soils using dye tracer distributions. *J. Contam. Hydrol.* **79**, to appear.
- Crestana, S. and Posadas, A.N.D. (1998) 2-d and 3-d fingering in unsaturated soils investigated by fractal analysis, invasion percolation modeling and non-destructive image processing. In Baveye, P., Parlange, J.Y., and Stewart, B.A., editors, *Fractals in Soil Science*. Boca Raton: CRC Press.
- Schwartz, R.C., McInnes, K.J., Juo, A.S.R., and Cervantes, C.E. (1999) The vertical distribution of a dye tracer in a layered soil. *Soil Sci.*, **164**, 561–573.

## See Also

[SoPhy](#), [plotFlow2d](#), [plotFlow3d](#)

## Examples

```
final <- FALSE # final <- TRUE
printlevel <- 1 + !interactive()

sh.jh(input=if (!interactive()) c(1, 2, 5, 0), final=final,
      Print=printlevel)

## Not run:
sh.jch(input=if (!interactive()) c(1:14, 0), final=final,
      Print=printlevel)

## End(Not run)
```

---

simulateHorizons

*Simulation of the stochastic part of the definition of the horizons*

---

## Description

simulateHorizons simulates the Gaussian random fields that define the Miller-similar medium, the stones and the roots according to the definition of the horizons.

## Usage

```
simulateHorizons(h, first = 1, what = 'all',
                PrintLevel = RFparameters()$Print,
                message = function(s) {print(s)}, stone.trials=50)
```

**Arguments**

h	a list of the same format as the output of <a href="#">xswms2d</a>
first	first gives the first horizon for which a Gaussian random field should be generated. It is assumed that the preceding horizons have already been simulated. first greater than 1 is suitable in the following situation. Assume the random fields for all horizons of the profile have been generated. Then the random field parameter of the <i>n</i> th horizon is changed. For updating the simulated random fields it suffices now to call simulateHorizons by first= <i>n</i>
what	what is a second parameter after first that is used to determine the starting point of the updating procedure: <b>'all'</b> the simulations of the Gaussian random fields (according to first), the simulation of the stones and the simulation of the roots are performed <b>'stone'</b> only the simulations for the stones and the roots are performed <b>'root'</b> only the simulations for the roots are performed <b>'randomfield'</b> only the simulations for the Gaussian random fields are performed
PrintLevel	If non-positive nothing is printed. The higher the number the more information is given on during the calculations.
message	function that has a string as argument. message is called for printing error messages and for the current stage of the calculations. message is called independently of the value of PrintLevel. message is intended to be used for messages on graphical devices.
stone.trials	number of trials to find a realisation for the spatial distributions of the stones before simulate.stones gives up.

**Value**

List of the same format as the output of [xswms2d](#). The following elements are changed in general

random.seed	The random seed for the simulation of the Gaussian random fields. It is changed when what='all' or random.seed has been NULL
stone.random.seed	The random seed for the simulation of the stones. It is changed when what=%in%c('all', 'stone') or stone.random.seed has been NULL
root.random.seed	The random seed for the simulation of the roots. It is changed when what=%in%c('all', 'stone', 'root') or root.random.seed has been NULL
RF	the simulations of the Gaussian random fields
Stone.RF	the simulations of the profile including the values for the stones, see also <a href="#">create.stones</a>
Root.RF	the simulations of the profile including the root values if \$rf.Kf=TRUE, see also <a href="#">create.roots</a>
rf.complete	logical. rf.complete is FALSE if the simulation has not been completed (because of an error).

**Note**

As a side effect the global variable `.Random.seed` is reset.

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**References**

Schlather, M. (2001) Simulation and analysis of random fields *R News* **1** (2), 18-20.

**See Also**

[SoPhy](#), [xswms2d](#)

**Examples**

```
h <- xswms2d(xlim=c(1, 50), ylim=c(1, 50), step=1, new=NULL)
h$n <- 2          ## define a second
h$H2 <- h$H1
h$H2$type <- "H"  ## genuine horizon, not polygon

## coordinates of the boundary segments between the horizons
h$H2$points <- list(x=seq(1, 100, 5))      ## x coordinates
h$H2$points$y <- 40 + 20 * cos(h$H2$points$x / 5) ## y coordinates

## Gaussian random field of the second horizon is pure nugget effect:
h$H2$model <- list(model=list("$", var=0.5, scale=1, list("nugget")),
                  trend=0.5)
h$H2$stone$lambda <- 0.005 ## intensity of the stones in the 2nd horizon
h <- calculate.horizons(h)
for (i in 1:2) {
  plotRF(simulateHorizons(h)) ##
  readline("Press return")
}
```

**Description**

This package provides a collection of tools that might be useful in soil physics. It is a first version of work in progress.

## Details

Currently, the following functionalities are provided:

- **Physical modelling**
  - `swms2d` : R interface to SWMS<sub>2d</sub>, an algorithm by Simunek et. al based on finite elements to simulate water flux and solute transport in soil
  - `xswms2d` : interactive plot to define soil structure and to simulate water flux and solute transport
  - various functions for the modified simulation of the water flux or the solute transport in a profile; see `calculate.horizons` for an example
- **Stochastic modelling**
  - `flowpattern` stochastic simulation of dye tracer profiles
- **Estimation of the risk index**
  - `analyse.profile` : analysis of Brilliant Blue tracer experiments w.r.t. the contamination risk index
  - `BrilliantBlue` : examples of profile pictures
  - `Pareto` : the Pareto distribution
  - `risk.index` : estimation of the risk index
- **Estimation of the variability of the tortuosity**
  - `tortuosity` : estimation of tortuosity from Brilliant Blue tracer experiments
- **Graphics**
  - `my.legend` : legend giving a colour spectrum
  - `quader` : tools for 3d perspective plot
  - `read.picture`, `plotRGB` : Conversion of tiff pictures to RGBA array and plotting RGB figures
- **Code used in publications by Schlather and Huwe**
  - `sh.jh` : Schlather, M. and Huwe, B. (2005a)
  - `sh.jch` : Schlather, M. and Huwe, B. (2005b)

## Acknowledgement

The author thanks Rien van Genuchten, Carl A. Mendoza, Rene Therrien, and Edward Sudicky for kindly changing the former copyright to the GNU copyleft licence, hence making their code directly available for this package.

The work has been financially supported by the German Federal Ministry of Research and Technology (BMFT) grant PT BEO 51-0339476C during 2000-03.

## Installation

LINUX: The tiff-library is available from <http://www.remotesensing.org/libtiff>. Users of SuSe install libtiff and libtiff-devel.

WINDOWS: The binaries (libtiff3.dll, zlib1.dll, jpeg62.dll) are available from [gnuwin32.sourceforge.net/packages/tiff.htm](http://gnuwin32.sourceforge.net/packages/tiff.htm). Before starting the R session, add in the system variable %PATH the path to the libtiff3.dll file; the standard path to be added is C:\Programme\GnuWin32\bin. Further, make sure that zlib1.dll and jpeg62.dll are present.

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

The Fortran code has been published as 'swms\\_2d' by Jirka Simunek, T. Vogel and Martinus Th. van Genuchten, <http://www.ussl.ars.usda.gov/MODELS/MODELS.HTM>; the file ORTHOFEM.f was written by Edward A. Sudicky and Carl A. Mendoza, based on code by Frank W. Letniowski and contributions by Rene Therrien, and modified by Jirka Simunek.

**References**

Schlather, M. (2006) SOPHY - a package for the simulation and the analysis of water flux in soil *In preparation*

Schlather, M. and Huwe, B. (2005a) A stochastic model for 3-dimensional flow patterns in dye tracer experiments. *J. Hydrol.* **310**, 17-27.

Schlather, M. and Huwe, B. (2005b) A risk index for characterising flow pattern in soils using dye tracer distributions *J. Contam. Hydrol.* **79**, to appear.

Schlather, M. and Huwe, B. (2006) Modelling inhomogeneous soils: Part I, Theory *In preparation*

Schlather, M. and Huwe, B. (2006) SOPHY - an interactive programme for water flow modelling *In preparation*

Schlather, M., Zeilinger, J. and Huwe, B. (2006) Modelling inhomogeneous soils: Part II, Applications *In preparation*

Simunek, J., Vogel, T., and van Genuchten, M.Th. (1994) *The SWMS\\_2D code for simulating water flow and solute transport in two-dimensional variably saturated media, Version 1.21*. Research Report No. 132, 197 p., U.S. Salinity Laboratory, USDA, ARS, Riverside, California.

---

 swms2d

---

*SWMS2D function call*


---

**Description**

Modelling of water flow and solute transport by SWMS2D

**Usage**

```
swms2d(d, max.iteration = 1e+05, iter.print = max.iteration, ShortF = TRUE,
      message = NULL, breakpoint = 1e+10, intermediate.result = NULL)
```

**Arguments**

d	list; see details
max.iteration	maximum number of incremental time steps
iter.print	number of incremental time steps after which a short message of status is given
ShortF	logical. If TRUE information is provided only at selected time points given by d\$TPrint

message	NULL or a function that takes a scalar time argument and returns a logical value. The function is called in case <code>max.iteration</code> is exceeded. The function may allow the user to decide on the basis of the current time whether the simulation should be continued.
breakpoint	number of incremental time steps after which <code>intermediate.result</code> is called
<code>intermediate.result</code>	NULL or a function with two parameters: the first one gives the time, the second is a $(6 \times \text{NumNP}) \times 1$ -array where the first dimension gives the (1) current pressure $h$ , (2) discharge rate $Q$ , (3) water content $\theta$ , (4) x-component $v_x$ of Darcian flux, (5) z-component $v_z$ of Darcian flux, and (6) the solute concentration; <code>NumNP</code> gives the number of nodes; <code>intermediate.result</code> can be used to create movies of water flow at calculation time.

## Details

`d` is a profile definition of a type as the return list of `link{xswws2d}`, or it is a list that should contain the following elements

- `Kattype` of flow system; 0:horizontal, 1:axisymmetric; 2:vertical.
- `MaxIt` maximum number of iteration during any time step.
- `TolT` maximum desired change of water content within one step.
- `TolH` maximum desired change of pressure head within one step.
- `lWat` logical. If TRUE transient water flow else steady state.
- `FreeD` logical. TRUE if free drainage at the bottom.
- `NLayer` number of subregions for water balances.
- `hTabc` (`hTab1`, `hTabN`), interval of pressure heads within which a table of hydraulic properties is generated.
- `Par` vector of 9 elements or matrix of 9 columns, each row representing a different kind of material.
- `dtinitial` time increment.
- `dtMinMax` (`dtmin`, `dtmax`), minimum and maximum permitted time increment.
- `DMulc` (`dMul1`, `dMul2`),  $dMul1 \geq 1$ ,  $dMul2 \leq 1$ ; if number of required iterations is less than 4 or greater than 6 then the next time step is multiplied by `dMul1` and `dMul2`, respectively.
- `TPrint` vector of increasing print-times.
- `NPseepage` information: matrix or list. matrix of one or two rows and arbitrary columns or each row defines a seepage face, the columns give the node numbers (or are NA) or a list of vectors;
- `DrCorr` reduction factor for drainage; see the SWMS2D manual.
- `ND` vector of up to two elements given the global node number of the drain.
- `EfDim` (`length{ND} \times 2`)-matrix drain information: first column gives the effective diameter; second column gives the dimension of the square in the finite element mesh.
- `KE1Dr` drainage information: matrix or list; matrix of `length(ND)` rows and arbitrary columns; each row defines the surrounding finite elements of the drain; values are either the global finite element numbers or NA or a list of vectors.

- Epsitemporal weighing coefficient: 0: explicit scheme, 0.5:Crank-Nicholson, 1:fully implicit.
- lUpWlogical. Upstream weighing formalation used if TRUE else Galkerin formulation.
- lArtDlogical. TRUE if artificial dispersion is to be added to fulfill the stability criterion PeCr.
- PeCrStability criterion, see the SWMS2D manual; zero if lUpW.
- ChParvector of 9 element if Par is a vector; other wise (nrow(Par) x 9)-matrix; chemical material properties: bulk density, ionic diffusion coefficient in free water, longitudinal dispersivity, transverse dispersivity, Freundlich isotherm coefficient, first-order rate constant for dissolved phase, first-order rate constant for solid phase, zero-order rate constant for dissolved phase, zero-order rate constant for solid phase.
- KodCBvector of  $\sum(nCodeM[, 2] \neq 0)$  elements, defining the chemical boundary conditions, see the SWMS2D manual.
- cBoundvector of six elements: jth element gives the concentration for boundary nodes i with chemical boundary condition  $abs(KodCB[i]) = j$ .
- tPulse duration of the concentration pulse.
- nCodeMmatrix of 11 or 12 columns and arbitrary number of rows; nCodeM gives the nodal information; the 12 columns contain: (1) nodal number (might be omitted), (2) code giving the boundary condition (see the SWMS2D manual), (3) x-coordinate, (4) z-coordinate, (5) initial pressure head, (6) initial concentration, (7) prescribed recharge or discharge rate, (8) material number, (9) water uptake distribution value, (10) pressure head scaling factor, (11) conductivity scaling factor, (12) water content scaling factor.
- KXRmatrix of 8 columns and arbitrary rows; gives the finite element information; first 4 colums give the nodes in counter-clockwise order (last node is repeated if element is a triangle), 5th to 7th column give anisotropy parameters of the conductivity tensor (angle, first and second principal component), 8th column: subregion number.
- Widthscalar or vector of  $\sum(nCodeM[, 2] \neq 0)$  elements; width of the boundary associated with the boundary nodes (in the order given by nCodeM).
- rLenwidth of soil surface associated with transpiration
- Nodevector of observation nodes for which information is collected at each time level.
- SinkFlogical; water extraction from the root zone if TRUE.
- qGWLflogical; If TRUE then the discharge-groundwater level relationship is used, see the SWMS2D manual.
- GWL0Lreference position of groundwater table (usually the z-coordinate of the soil surface).
- Aqhparameter in the discharge-groundwater level relationship.
- Bqhsecond parameter in the discharge-groundwater level relationship.
- tInit starting time of the simulation.
- hCritSmaximum allowed pressure head at soil surface.
- atmospherematrix of 10 columns and arbitrary rows; each row presents the atmospherical conditions at a certain instance; the columns are (1) the instance at which the time period ends (2) precipitation, (3) solute concentration of rainfall water, (4) potential evaporation rate, (5) potential transpiration rate, (6) absolute value of minimum allowed pressure head at the soil surface, (7) drainage flux across the bottom boundary (for  $abs(nCodeM[, 2]) = 3$ ), (8) ground water level, (9) concentration of drainage flux (for  $(abs(nCodeM[, 2]) = 3) \ \& \ (KodCB < 0)$ ) (10) concentration of drainage flux (for  $(abs(nCodeM[, 2]) = 3) \ \& \ (KodCB > 0)$ ).

- `rootvector` of 6 elements or  $(nrow(nCodeM) \times 6)$ -matrix. `c(P0, P2H, P2L, P3, r2H, r2L)` information about water uptake by roots, see the SWMS2D manual.
- `P0ptm` pressure heads, below which roots start to extract water at maximum possible rate; if `is.vector(root)` then `P0ptm` is a scalar if `Par` is a vector, and a vector of  $nrow\{Par\}$  elements otherwise; if `!is.vector(root)` then `P0ptm` has  $nrow(nCodeM)$  elements.

## Value

If an error occurred a string containing the error message is returned. Otherwise the result is a list of the following elements

<code>hQThFlc</code>	array( $dim=c(6, NumNP, n)$ ) where $n \leq MPL + 2$ (simulation output); first dimension gives (i) water pressure H, (2) discharge rate Q, (3) water content theta, (4) x-component $v_x$ of Darcian flux, (5) z-component $v_z$ of Darcian flux, and (6) concentration; the third dimension gives the time points given by <code>TPrint</code> except the first that gives the initial conditions.
<code>T1SolObs</code>	matrix with $46 + 3 * Nobs$ columns (simulation output); the columns are (1) time point, (2) <code>rAtm</code> , (3) <code>rRoot</code> , (4-10) <code>vK[Atm, Root, 3, 1, Seep, 5, 6]</code> , (11-17) <code>hK[Atm, Root, 3, 1, Seep, 5, 6]</code> , (18) <code>CumQAP</code> , (19) <code>CumQRP</code> , (20-26) <code>CumQ[Atm, Root, 3, 1, Seep, 5, 6]</code> , (27) <code>dt</code> , (28) <code>Iter</code> , (29) <code>ItCum</code> , (30) <code>Peclet</code> , (31) <code>Courant</code> , (32) <code>CumCh0</code> , (33) <code>CumCh1</code> , (34) <code>ChumR</code> , (35-40) <code>ChemS[1:6]</code> , (41-46) <code>SMean[1:6]</code> , <code>hNew[1:Nobs]</code> if <code>Nobs &gt; 0</code> , <code>ThNew[1:Nobs]</code> if <code>Nobs &gt; 0</code> , <code>ConcNew[1:Nobs]</code> if <code>Nobs &gt; 0</code> ; see the SWMS2D manual for details.
<code>atmOut</code>	$(MaxA1 \times 9)$ -matrix (simulation output); (1) <code>AtmTime</code> , (2) <code>CumQAP</code> , (3) <code>CumQRP</code> , (4) <code>CumQA</code> , (5) <code>CumQR</code> , (6) <code>CumQ3</code> , (7) <code>hAtm</code> , (8) <code>hRoot</code> , (9) <code>hKode3</code> ; see the SWMS2D manual for details.
<code>balance</code>	matrix of <code>balance.ncol</code> columns and at most <code>boundary.n</code> rows (simulation output); first 5 columns are (1) time, (2) <code>WatBalT</code> , (3) <code>WatBalR</code> , (4) <code>CncBalT</code> , (5) <code>CncBalR</code> ; for each of the subjects (i) Area, (ii) Volume, (iii) <code>InFlow</code> , (iv) <code>hMean</code> , and (v) <code>ConcVol</code> and (vi) <code>cMean</code> if solute transport, <code>NLay + 1</code> subsequent columns are given for the total value and the value of each subregion; see the SWMS2D manual for details.
<code>boundary</code>	array( $dim=c(NumBP, 11, n)$ ) where $n \leq boundary.n$ (simulation output); the second dimension gives (1) i, (2) n, (3) x, (4) z, (5) Code, (6) Q, (7) v, (8) h, (9) th, (10) Conc, (11) time; see the SWMS2D manual for details.
<code>flux</code>	<code>d\$flux</code> ; different from NULL if the function is called with the output of <code>create.waterflow</code> or with a profile definition, see the return value of <code>xswms2d</code> .
<code>water.x</code>	<code>d\$water.x</code> ; different from NULL if the function is called with the output of <code>create.waterflow</code> or with a profile definition, see the return value of <code>xswms2d</code> .
<code>water.y</code>	<code>d\$water.y</code> ; different from NULL if the function is called with the output of <code>create.waterflow</code> or with a profile definition, see the return value of <code>xswms2d</code> .

## Acknowledgement

The author is grateful to Rien van Genuchten, Carl A. Mendoza, Rene Therrien, and Edward Sudicky for kindly changing the copyright of the modified SWMS2D code, as used in this package, to the GNU copyleft licence.

Note that, however, HYDRUS2D, an advanced development of SWMS2D, and its code is proprietary.

The work has been financially supported by the German Federal Ministry of Research and Technology (BMFT) grant PT BEO 51-0339476C during 2000-2002.

### Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

The Fortran code was published as 'swms\\_2d' by Jirka Simunek, T. Vogel and Martinus Th. van Genuchten, <http://www.ussl.ars.usda.gov/MODELS/MODELS.HTM>; the file ORTHOFEM.f was written by Edward A. Sudicky and Carl A. Mendoza, based on code by Frank W. Letniowshi and contributions by Rene Therrien, and modified by Jirka Simunek.

### References

- Schlather, M. and Huwe, B. (2004) The use of the language interface of R: two examples for modelling water flux and solute transport. *Computers & Geosciences* **30**, 197-201.
- Simunek, J., Vogel, T., and van Genuchten, M.Th. (1994) *The SWMS2D code for simulating water flow and solute transport in two-dimensional variably saturated media, Version 1.21*. Research Report No. 132, 197 p., U.S. Salinity Laboratory, USDA, ARS, Riverside, California.

### See Also

[xswms2d](#), [read.swms2d.table](#), [SoPhy](#)

### Examples

```
#####
## SWMS2D, Example 1, stochastically modified material properties
#####
# E = exp(mu + 1/2 * sd^2)
# Var = exp(2*mu + sd^2)*(exp(sd^2) - 1)

path <- paste(system.file(package='SoPhy'), 'swms2d', sep="/")
x <- read.swms2d.table(path)
x$TPrint <- 20 * 60
sd <- 0.3 ## then sd of log.gauss is 0.306878
mu <- log(x$nCodeM[, 10]) - 0.5 * sd^2
n <- if (interactive()) 100 else 10 # better 10000
front <- integer(n)

for (j in 1:n) {
  cat(j, "\n")
  x$nCodeM[, 10] <- rlnorm(length(x$nCodeM$z), m = mu, sd = sd)
  x$nCodeM[, 11] <- x$nCodeM[, 10]^(-2)
  z <- swms2d(x)$hQ[3, , ]
  front[j] <- (min(which(z[-1:-2, 2] / z[-1:-2, 1] < 1.005)) %/% 2) + 1
```

```

}

idx <- 24
cat('The probability that the front advances at most',
    max(x$nCodeM$z) - x$nCodeM$z[2 * idx], 'cm is about',
    format(mean(front <= idx), dig=2), '\n')

par(cex=1, mar=c(4.2,4.2,1.2,0.4))
distance <- max(x$nCodeM$z) - x$nCodeM$z[2 * front]
h <- sort(unique(distance))
dh <- min(diff(h) / 2)
hist(distance, freq=FALSE,
      breaks=seq(h[1]-dh, h[length(h)]+dh, 2 * dh),
      main="Histogramm for the depth of the water front",
      xlab="depth", cex.axis=1.5, cex.main=1.2, cex.lab=1.5)

```

---

tortuosity

*Variability of the tortuosity*


---

### Description

The function estimates the ratio between the mean tortuosity and the smallest tortuosity from profiles of Brilliant Blue tracer experiments

### Usage

```
tortuosity(depth, freq, range.depth, range.xi, len.xi=100,
           lower.bound.m = 1.005, tolerance.m=0.01, PrintLevel=0)
```

### Arguments

depth	depth in pixels
freq	number of stained pixels found in depth depth
range.depth	range of depths for which $1 - H^*$ is fitted
range.xi	range of values of $\xi$ for which $1 - H^*$ is fitted
len.xi	range.xi and len.xi define the grid $\text{seq}(\min(\text{range.xi}), \max(\text{range.xi}), \text{len}=\text{len.xi})$ for which $1 - H^*$ is fitted.
lower.bound.m	lower bound for $m$ used in the numerical optimisation
tolerance.m	non-negative value. If tolerance.m=0 only the optimal $\xi$ is used to estimate $m$ . If tolerance.m>0 then all estimated $m(D, \xi)$ are considered for which the span of $m$ is less than or equal to $(1 + \text{tolerance.m})$ times the minimal span value.
PrintLevel	integer. If the value is greater than 1 then tracing information is given

**Value**

tortuosity returns a list of the following elements:

<code>xi</code>	vector of length <code>len.xi</code> . It contains the values of $\xi$ used in the algorithm
<code>raw.m</code>	matrix with <code>len.xi</code> rows. The number of columns equals the number of depth values that are between the given bounds <code>range.depth</code> . It contains the fitted values of $m$
<code>span.xi</code>	vector of length <code>len.xi</code> contains span of all <code>raw.m</code> values for the respective value of $\xi$
<code>span.m</code>	vector of length <code>len.xi</code> . It contains the difference between the largest and the smallest value of <code>raw.m</code> for given $\xi$
<code>opt</code>	list of the optimal values: <code>xi</code> , <code>m</code> , <code>s</code> , <code>mspan</code> (the <code>span.xi</code> found at <code>xi</code> ) and <code>D</code> (threshold depth)
<code>input</code>	list of two vectors: <code>freq</code> and <code>depth</code>
<code>fitted</code>	list of two vectors that give the best fitted curve: <code>depth</code> and <code>p</code> ; the latter cannot be given if <code>tolerance.m &gt; 0</code>

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**References**

Schlather, M. and Huwe, B. (2005) A characteristic for the variability of tortuosity. *Submitted*.

**Examples**

```
data(F04)
path <- paste(system.file(package='SoPhy'),'tracer", sep="/")
F04$name <- paste(path, "F04.G.tif", sep="/")
m <- analyse.profile(F04, estimate.all=FALSE, method="fix.m",
                    selected.rate=c(0, 0), selected.dist=NULL,
                    interactive=FALSE, Print=0)
stained <- m$fct(m$picture, m$param)
freq <- m$absfr
truedepth <- m$r.i$data[, 1] + ncol(stained) - m$loc[[1]]$y[2]
tol = 0

mt <- tortuosity(truedepth, freq, range.depth=c(85, 97),
                range.xi=c(-2.5, -0.6), Print=2, tol=tol)
str(mt$opt)

# fig. 3A in Schlather and Huwe (2005)
matplot(mt$xi, mt$raw.m, xlab=expression(xi), ylab="m",
        type="l", col=1, lty=1)

# fig. 3B in Schlather and Huwe (2005)
```

```

plot(mt$xi, mt$span.m, xlab=expression(xi), ylab="span m",
     type="l", col=1, lty=1)
for (cex in c(9, 9.5, 10))
  points(mt$opt$x, mt$opt$mspan, cex=cex, col="darkgrey")

# fig. 2 in Schlather and Huwe (2005)
plot(mt$input$depth, mt$input$freq, xaxs="i", yaxs="i",
     pch=16, ylab="stained pixels", xlab="depth [pixels]",
     col="blue", cex=2)
lines(mt$fitted$depth, mt$fitted$p, lwd=3, col=2)

```

---

Tracer experiments      *Brilliant Blue tracer profile*

---

### Description

Digitised pictures of profiles that have been excavated after Brilliant Blue tracer experiments. The picture were taken by the soil physics group at ETH Zurich, see Source and References below.

### Format

3-dimensional array, where the last dimension codes red, green, blue, and alpha.

### Source

The data were collected by the soil physics group at ETH Zurich, <http://www.ito.umnw.ethz.ch/SoilPhys/Fliessmuster/>

### References

#### Data

- Flury, M., Leuenberger, J., Studer, B. and Jury, W.A. and Fluehler, H. (1992-94) *Präferentielle Fließwege in Ackerböden - ein screening-Test unter Feldbedingungen*. [http://www.ito.umnw.ethz.ch/SoilPhys/Fliessmuster/projekt\\_flury.html](http://www.ito.umnw.ethz.ch/SoilPhys/Fliessmuster/projekt_flury.html)
- von Albertini, N., Laeser, H.P., Leuenberger, J. and H. Fluehler (1994-97) *Strukturerholung eines verdichteten Ackerbodens unter einer mehrjaehrigen Kunstwiese*. [http://www.ito.umnw.ethz.ch/SoilPhys/Fliessmuster/projekt\\_vonalbertini.html](http://www.ito.umnw.ethz.ch/SoilPhys/Fliessmuster/projekt_vonalbertini.html)

#### Papers

- von Albertini, N., Leuenberger, J., Laeser, H.P. and H. Fluehler (1995) Regeneration der Bodenstruktur eines verdichteten Ackerbodens unter Kunstwiese *Bodenkundliche Gesellschaft der Schweiz, Dokument 7*, 10-16
- Flury, M. and Fluehler, H. (1994) Brilliant Blue FCF as a Dye Tracer for Solute Transport Studies - A Toxicological Overview *J. Environm. Quality* **23**, 1108-1112
- Flury, M. and Fluehler, H. (1995) Tracer characteristics of Brilliant Blue FCF. *Soil Sci. Soc. Am. J.* **59**, 22-27.

- Flury, M., Fluehler, H., Jury, W.A. and Leuenberger, J. (1994) Susceptibility of Soils to Preferential Flow of Water: A Field Study. *Water Resour. Res.* **30**, 1945-1954
- German-Heins, J. and Flury, M. (2000) Sorption of Brilliant Blue FCF in soils as affected by pH and ionic strength. *Geoderma* **97**, 87-101.
- Schlather, M. and Huwe, B. (2005) A risk index for characterising flow pattern in soils using dye tracer distributions. *J. Contam. Hydrol.* **79**, to appear.

## Examples

```

path <- paste(system.file(package='SoPhy'), 'tracer', sep="/")
for (nm in c("F04", "F06", "K06")) {
  cat("\n", nm, ":\nOriginal picture. ", sep="")
  p <- read.picture(paste(path, "/", nm, sep=""))
  plotRGB(p)
  readline("Press return.")
  p <- read.picture(paste(path, "/", nm, ".G", sep=""))
  plotRGB(p)
  readline("Picture after first preparation. Press return.")
}

## Not run:
path <- paste(system.file(package='SoPhy'), 'tracer', sep="/")
F04 <- read.picture(paste(path, "F04", sep="/"))
F04g <- read.picture(paste(path, "F04.G", sep="/"))
F06 <- read.picture(paste(path, "F06", sep="/"))
F06g <- read.picture(paste(path, "F06.G", sep="/"))
K06 <- read.picture(paste(path, "K06", sep="/"))
K06g <- read.picture(paste(path, "K06.G", sep="/"))

## End(Not run)

```

---

useraction

*Set input behaviour*

---

## Description

The functions store values that influence the behaviour of [Locator](#) and [Readline](#)

**Usage**

```

useraction(action=c("none", "start.register", "continue.register",
                  "replay", "endless", "delete"),
          sleep, wait, PrintLevel, actionlist)
putactions(l)

```

**Arguments**

action	<p>string of the following values</p> <p>“<b>none</b>” the behaviour of <a href="#">Readline</a> and <a href="#">Locator</a> is that of <a href="#">readline</a> and <a href="#">locator</a>, respectively. The values are <b>not</b> stored.</p> <p>“<b>start.register</b>” the behaviour of <a href="#">Readline</a> and <a href="#">Locator</a> is that of <a href="#">readline</a> and <a href="#">locator</a>, respectively. The current list of stored values is deleted and new inputs by <a href="#">Readline</a> and <a href="#">Locator</a> are <b>stored</b>.</p> <p>“<b>continue.register</b>” the behaviour of <a href="#">Readline</a> and <a href="#">Locator</a> is that of <a href="#">readline</a> and <a href="#">locator</a>, respectively. The current list of stored values is continued with inputs from <a href="#">Readline</a> and <a href="#">Locator</a>.</p> <p>“<b>replay</b>” <a href="#">Locator</a> and <a href="#">Readline</a> do not get the data from the mouse or terminal, but read them from the stored list. If the end of storage list is reached, the user is informed and the action is switched to “none”.</p> <p>“<b>endless</b>” similar to “replay”, but storage list is looped.</p> <p>“<b>delete</b>” The current list of stored values is deleted and action is set to “none”</p>
actionlist	list of stored values by <a href="#">Readline</a> and <a href="#">Locator</a>
sleep	sleeping time before the blinking starts (replay mode of <a href="#">Locator</a> ); value is kept until explicit changing; starting value: 2.5
wait	regulated the blinking speed of the characters ( <a href="#">Locator</a> ) and of the speed of appearance of characters ( <a href="#">Readline</a> ); value is kept until explicit changing; starting value: 0.1
PrintLevel	if <code>Printlevel&gt;1</code> and <a href="#">Readline</a> or <a href="#">Locator</a> is in replay mode then the current pointer position within the storage list and the tracing information (if any) is shown. The value is kept until explicit changing; starting value: 0
l	list of all parameters relevant to the behaviour of <a href="#">Readline</a> and <a href="#">Locator</a> .

**Details**

The function `putactions` restores the state taken by `getactions`.

Concerning the function `useraction`, the parameter `action` must be given. If any of the other parameters is not given, its current value is kept. If `action="start.register"` or `action="delete"`, the value of `actionlist` is ignored.

If replaying, [Readline](#) shows the input values on the screen and [Locator](#) shows blinking characters as side effects.

**Value**

useraction returns NULL.

putactions returns invisibly the state before the new state is set.

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**See Also**

[eval.parameters](#), [getactions](#), [Locator](#), [Readline](#),

**Examples**

```
sample.interaction <- function(read.r2=TRUE, type="p", pch=16) {
  r1 <- Readline(prompt="first input: ")
  r2 <- if (read.r2) Readline(prompt="second input: ", info="2nd input")
  else " -- no input -- "
  cat("Mark some points with the right mouse key, then leave with the left mouse key\n")
  do.call(getOption("device"), list())
  plot(Inf, Inf, xlim=c(-1,1), ylim=c(-1,1), xlab="", ylab="")
  l <- Locator(100, info="locator input", type=type, pch=pch)
  r3 <- Readline(prompt="third input: ", info="last input")
  return(list(r1=r1, r2=r2, l=l, r3=r3))
}
```

```
## Not run:
```

```
#####
## user terminal input ##
#####
```

```
{
  useraction("start.register")
  str(sample.interaction())
  str(l <- getactionlist())
}
```

```
#####
## just replay ##
#####
```

```
useraction("replay", sleep=2, wait=interactive() * 0.2 * 5,
           PrintLevel=2, actionlist=l)
str(sample.interaction(type="l"))
```

```
#####
```

```

## modify first the input list, then replay ##
#####
l2 <- l[-2]
l2[[1]] <- list("some other words", info="changed input")
str(l2)
useraction("replay", sleep=1, wait=interactive() * 0.05,
           PrintLevel=0, l2)
str(sample.interaction(read.r2=FALSE, type="o")) # input now from l2
Readline(prompt="new input: ", info="?!") # switch to terminal
##                                     since end of stored list
# str(getactionlist()) # new input has not been not stored ...

#####
## use of the two lists, l and l2, in a mixed way ##
#####
useraction("replay", sleep=2, wait=interactive() * 0.05,
           PrintLevel=0, actionlist=l)
Readline(prompt="first input of 1: ")
dump <- getactions()
useraction("replay", sleep=0.5, wait=interactive() * 0.05,
           PrintLevel=0, actionlist=l2)
Readline(prompt="first input of 2: ")
dump2 <- putactions(dump)
Readline(prompt="second input of 1: ")
## locator call reading from list 1:
plot(Inf, Inf, xlim=c(-1,1), ylim=c(-1,1), xlab="", ylab="")
Locator(100, info="locator input", type="p")
dump <- putactions(dump2)
##locator call reading from list 2:
Locator(100, info="locator input", type="p", pch=20, col="blue")
Readline(prompt="last input of 2: ")
putactions(dump)
Readline(prompt="last input of 1: ")

#####
## see help("eval.parameters") for another example ##
#####

## End(Not run)

```

---

write.picture

*RGB to TIF*


---

## Description

The function writes an RGB-array to a TIF file.

**Usage**

```
write.picture(picture, tif)
```

**Arguments**

```
picture      three dimensional array where the third dimension must have length 3 or 4
tif          character. Name of the tif file.
```

**Value**

None.

**Author(s)**

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

**See Also**

[plotRGB](#), [read.picture](#)

---

xswms2d

*Modelling of water flux and solute transport*

---

**Description**

Interactive surface that is used to define a soil profile and to give the physical and chemical properties for each horizon. It simulates the water flux and solute transport using swms2d by J. Simunek, T. Vogel, and M.Th. van Genuchten.

**Usage**

```
sophy(...)

xswms2d(h, xlim, ylim, step, picture=NULL,

water=list(

  TPrint=500, red=4, print=7, mesh=TRUE, lim.rf = 1,
  lim.swms2d = 1,

  TolTh=0.0001, TolH=0.01, lWat=TRUE, dt=1, dtMinMax = c(0.01, 60),
  max.iteration=1000, max.iter.prec=50000,
  iter.print=100, breakpoint=100,

  top.bound=2, top.value=0, bottom.bound=1, bottom.value=0
),
```

```

materials=list(
  thr=.02, ths=0.35, tha=0.02, thm=0.35, Alfa=0.041, n=1.964,
  Ks=0.000722, Kk=0.000695, thk=0.2875,
  first=1, second=1, angle=0,
  Hslope=0, Hseg=-100, POptm=-25, sharpness=1,
  Bulk.d=1500, Diffus=0, longiDisper=1, transvDisper=0.5,
  Adsorp=0.0004, SinkL1=-0.01, SinkS1=-0.01, SinkL0=0, SinkS0=0
),
Hinit = function(Hseg, depth) Hseg,
model=list(model="exp", param=c(0, 0.25, 0, diff(ylim) * 0.1)),
anisotropy=NULL,
miller.link=function(rf) exp(sign(rf) * sqrt(abs(rf))),
millerH = function(lambda) lambda,
millerK = function(lambda) lambda^-2,
millerT = function(lambda) 1,

chemical=list(
  lChem=FALSE, Epsi=2, lUpw=1, lArtD=FALSE, PeCr=10, tPulse=500,
  top.bound=2, top.value=1, bottom.bound=1, bottom.value=0,
  root.uptake=0, intern.source=0
),

atmosphere=list(
  AtmInf=TRUE, tInit=0, Aqh=-.1687, Bqh=-.02674,
  hCritS=1.e30, GWL0L=yylim[2], rLen=diff(xlim)
),
atm.periods=1,
atm.data=c(
  end=100000000, prec=0, cPrec=0, rSoil=0, rRoot=0,
  hCritA= 1000000, rGWL=0, GWL=0, crt=0, cht=0
),

stone=list(

  value=NA, lambda=0,

  no.upper=FALSE, no.lower=TRUE, no.overlap=FALSE,

  main.distr=rnorm, main.mean=diff(ylim)/20, main.s=diff(ylim)/400,
  sec.distr=rnorm, sec.mean=diff(ylim)/50, sec.s=diff(ylim)/400,
  phi.distr=rnorm, phi.mean=1, phi.s=0
),

plant.types=1,

```

```

Kf=function(plant, distances, depth, rf,
             param=list(field.value=1, field.factor=0))
  rep(param$field.value, length(rf)) + param$field.factor * rf,
beta=function(plant, distances, depth, rf,
              param=list(beta.value=1, beta.factor=0))
  rep(param$beta.value, length(rf)) + param$beta.factor * rf,
root=list(
  plants.lambda=0,
  plants.mindist=diff(xlim) / 20,
  mean=sqrt(diff(ylim) * diff(xlim)),
  sd=sqrt(diff(ylim) * diff(xlim)) / 5,

  knot.probab=0.1,
  knot.mindist = 5 * step,
  shoots.3=0.4, shoots.4=0,
  stop.probab=function(knot, dist, m, s) 1 - exp(-m + s * dist),
  stop.m=0, stop.s=0,
  rf.link=function(v, m, s){
    v <- s * v
    m * (exp(sign(v) * sqrt(abs(v))))^(-2)
  },
  rf.m=0, rf.s=0,

  no.own.root=TRUE, age.bonus=1.2, depth.bonus=5.2,
  side.bonus=5.1, diagonal=TRUE, dir.ch=3.71, dir.ch.s=0.5,

  rf.Kf=FALSE, P0=-10, P2H=-200, P2L=-800, P3=-8000, r2H=0.5,
  r2L=0.1, root.condition=3, root.uptake=-150
),

root.zone = NULL,

col.rf = NULL, col.simu = NULL,

MaxIt=20, hTab=c(0.001,200), DMul=c(1.1, 0.33),

col.rect="red", col.bg="yellow", col.sep="gray", col.left="red",
col.mid="darkgreen", col.right="darkgreen", col.line="red",
col.txt="black", col.submenu="darkgreen", col.subord="steelblue",
col.forbid="gray88", col.bg.forbid="lightyellow", col.flash="blue",
## drawing configure CFLAGS="-O2 -Wall -pedantic"

col.hor=c("#000000", "#996600", "#660000", "#CC9933", "#666600",
          "#CCCC99", "#CCCCCC", "#990000", "#FFCC00", "#FFFFCC"),
col.draw="green", col.mess="red",

col.mesh="blue", col.mesh.pt="green",

```

```

col.exception = c("brown", "lightgray"),
cex.eval = 0.8,
areas=is.null(h$picture), PrintLevel=RFparameters()$Print,
new=TRUE, X11.width=9.5, X11.height=X11.width * 0.789,
frequent.reset = TRUE, update=FALSE, waterfall=FALSE,
zlim = NULL, Zlim = NULL,

print.par=list(ps="sophy", height=3, titl=TRUE, legend = TRUE)
)

```

## Arguments

h	a list of the output format, see the Details. If h is given the parameters xlim, ..., col.simu are ignored.
xlim	vector of two elements. The horizontal extension of the profile, see also picture. If diff(xlim) is not an integer multiple of step then xlim[2] is decreased suitably.
ylim	vector of two elements. The vertical extension of the profile, see also picture. If diff(ylim) is not an integer multiple of step then ylim[2] is decreased suitably.
step	numeric. the grid spacing for both directions. The finite element method is essentially based on a quadratic mesh.
picture	array or file name for a digitised profile in tif or jpeg format; if picture is an array it must have three dimensions, where the third dimension has 3 or 4 components (RGB or RGBA format). If picture is given then either xlim or ylim might be missing and is calculated from the other lim parameter and the dimension of the picture.
water	{swms2d, water} list of the following components <b>TPrint</b> { <b>TPrint</b> } scale; modelling time after which the result is printed. Note that for directly calls of <a href="#">create.waterflow</a> and subsequent call of <a href="#">swms2d</a> , TPrint might also be a vector, see <a href="#">read.swms2d.table</a> for further information. <b>red</b> { <b>size reduction</b> } positive integer. to increase the simulation speed the size of the finite element grid can be coarsened <b>print</b> { <b>printed variable</b> } integer 1,...,6. One of the following spatial data sets can be plotted: the water potential $H$ , discharge/recharge rates $Q$ for internal sink/sources, the water contents $\theta$ , the x-components of the Darcian flux vector $v_x$ , the z-components of the Darcian flux vector $v_z$ , or the solute concentration Conc. These variables are numbered consecutively from 1 to 6. <b>mesh</b> { <b>show FE mesh</b> } logical. If TRUE the currently used finite element mesh in a swms2d simulation is shown in the upper left drawing area. <b>lim.rf</b> { <b>upper quantile for random field plot</b> } numeric in $[0, 1]$ . The value of zlim[2] in image equals the lim.rf quantile of the random field values when the transformed random field is plotted. <b>lim.swms2d</b> { <b>quantile for swms2s plot</b> } numeric in $[0, 1]$ . See <a href="#">plotWater</a> for details.

- TolTh **{TolTh}** maximum desired change of water content within one step of the FE method.
- TolH **{TolH}** maximum desired change of pressure head within one step of the FE method.
- lWat **{LWat}** logical. If TRUE transient water flow else steady state.
- dt **{dt - initial time step}** initial time increment of the FE method.
- dtMinMax **{minimal time step} and {maximal time step}** c(dtmin, dtmax), minimum and maximum permitted time increment of the FE method.
- max.iteration **{max. iterations}** maximum number of incremental time steps for coarsed finite element meshes.  
Once max.iteration is reached, the current time and the aimed time is shown. The user is asked whether it should be continued.
- max.iter.prec **{max. iter. (precise)}** same as max.iteration except that this value is used if the main menu bottom “precise waterflow” is pressed.
- iter.print **{swms message, loop period}** number of incremental time steps after which a short message of status is given.
- breakpoint **{swms, image, loop period}** number of incremental time steps after which the current value of the spatial variable is shown in a two dimensional plot. If breakpoint is chosen appropriately the user gets the impression of a movie. Note that pictures are presented after a certain amount of iteration loops, so the length of the simulated time intervals between the pictures differ.
- top.bound **{top}** surface boundary condition: 1 (impermeable), 2 (Dirichlet), 3 (Neumann), 4 (variable H), 5 (variable Q), or 6 (atmospheric). See the SWMS2D manual for details.
- top.value **{top value}** the value of the top.bound condition, if appropriate.
- bottom.bound **{bottom}** boundary condition at the bottom: 1 (free drainage), 2 (deep drainage), 3 (impermeable), 4 (Dirichlet), or 5 (Neumann). See the SWMS2D manual for details.
- bottom.value **{bottom value}** the value of the bottom.bound condition, if appropriate.
- materials {material (phys) / material (chem)}; thr, ..., sharpness are found in the menu for the physical properties, Bulk.d, ..., SinkS0 in the menu for the chemical ones. List has the following components
- thr **{ $\theta_r$ }** residual water contents  $\theta_r$
- ths **{ $\theta_s$ }** saturated water contents  $\theta_s$
- tha **{ $\theta_a$ }** fictitious parameter  $\theta_a \leq \theta_r$ , introduced into the van Genuchten model for more flexibility (to allow for a non-zero air-entry value), see page 10 of the SWMS2d manual for details.
- thm **{ $\theta_m$ }**  $\theta_m \geq \theta_s$ , introduced into the van Genuchten model for more flexibility (to allow for a non-zero air-entry value)
- Alfa **{ $\alpha$ }** factor  $\alpha$  in the van Genuchten model
- n **{ $n$ }** exponent  $n$  in the van Genuchten model
- Ks **{ $K_s$ }** saturated hydraulic conductivity  $K_s$
- Kk **{ $K_k$ }** non-saturated hydraulic conductivity  $K_k$  measured for some  $\theta_k$

	thk $\{\theta_k\}$ see $K_k$
	first <b>{1st principal comp}</b> anisotropy parameter for the conductivity tensor
	second <b>{2nd principal comp}</b> anisotropy parameter for the conductivity tensor
	angle <b>{angle (degrees)}</b> anisotropy parameter for the conductivity tensor
	Hslope <b>{initial H, slope}</b> The initial matrix potential is calculated from the <code>miller.linked</code> and then <code>millerH</code> -transformed Gaussian random field by linear transformation. Hslope and <code>Hseg</code> , see <code>Hinit</code> , give the slope and the segment of that linear transformation.
	Hseg <b>{initial H, segment}</b> see Hslope and Hinit
	POptm <b>{POptm (root)}</b> highest (negative) matrix potential (so close to saturation) for which the water uptake is still optimal; see <code>root\$P2H</code> for further information
	sharpness <b>{sharpness}</b> The values for a random field (of the hydrolic conductivity) of a simulated horizon are obtained by a linear combination of a conditional simulation (towards random fields of the preceding horizons) and an independent simulation. sharpness is the factor for the independent simulation and $\sqrt{1 - \text{sharpness}^2}$ the factor for the conditional simulation. The mean of the random field to be simulated has been subtracted beforehand from the boundary conditions. sharpness is not used for the initial horizon, thus not given as a menu point in this case.
	Bulk.d <b>{bulk density}</b> bulk density
	Diffus <b>{diffusion}</b> ionic diffusion coefficient in free water
	longiDisper <b>{longitud. Dispers.}</b> longitudinal dispersivity
	transvDisper <b>{transvers. Dispers.}</b> transverse dispersivity
	Adsorp <b>{Freundlich isoth coeff.}</b> Freundlich isotherm coefficient
	SinkL1 <b>{1st-order const./dissolved}</b> first-order rate constant $\mu_w$ for dissolved phase, see page 15 of the SWMS2d manuscript
	SinkS1 <b>{1st-order const./solid}</b> first-order rate constant $\mu_s$ for solid phase
	SinkL0 <b>{0-order const./dissolved}</b> zero-order rate constant $\gamma_w$ for dissolved phase
	SinkS0 <b>{0-order const./solid}</b> zero-order rate constant $\gamma_s$ for solid phase
Hinit	function of two variables; the first variable is a vector of Hseg, the second the depths $d$ . the initial $h$ values are calculated as $\text{Hinit}(\text{Hseg}, d) + \text{Hslope} * \text{millerH}(Z)$ where $Z$ is the simulated random field. The initial $h$ value may afterwards be modified for certain segments due to the given boundary conditions and to a given Dirichlet condition for the roots.
model	<code>{structure}</code> the covariance model for the Gaussian random fields used for the definition of the Miller similar media. See <code>CovarianceFct</code> for details on the definition of the covariance model. If the mean equals NA and if it is the last horizon, the area is interpreted as air.

anisotropy	logical or NULL. If logical it overrides the anisotropy condition given by model. If model was anisotropic and anisotropic=FALSE, the first anisotropic scale parameter is used as scale for the isotropic model. If model is isotropic, only the representation is changed, and the user has the opportunity to change the values to an truly anisotropic model.
miller.link	function that transforms the Gaussian random field to a field of non-negative values. The argument and the value are matrices (of the same size). If NULL the miller.link is the identity.
millerH	function that transforms the miller.linked field into a field of water potential $H$ . This field is used for areas of constant potential, and for the potential at starting time. For the latter the millerH transformed field is further transformed linearly by a function of \$materials\$Hslope, see Hinit, and \$materials\$Hseg. The argument and the value are matrices (of the same size).
millerK	function that transforms the miller.linked field into a field of scaling factors that are associated with the saturated hydraulic conductivity. The argument and the value are matrices (of the same size).
millerT	function that transforms the miller.linked field into a field of scaling factors that are associated with the water content. The argument and the value are matrices (of the same size).
chemical	{swms2d (chem)} list of the following components 1Chem <b>{Solute transport}</b> logical that indicates whether solute transport should be modelled Epsi <b>{scheme}</b> integer 1,2,3. Temporal weighing coefficient: 1 (explicit: weight=0), 2 (Crank-Nicholson: weight=0.5) or 3 (implicit: weight=1) 1UpW <b>{formulation}</b> 1 (upstream weighing formulation) or 2 (Galerkin formulation) 1ArtD <b>{added artific. disp}</b> logical. If TRUE artificial dispersion is added to satisfy the stability criterion PeCr PeCr <b>{PeCr}</b> Stability criterion (Peclet number * Courant number); usually 2, but can be increased to 5 or 10; see page 44 of the SWMS2D manuscript. tPulse <b>{pulse duration}</b> time duration of the concentration pulse top.bound <b>{top}</b> boundary condition for soil surface: 1 (impermeable), 2 (Dirichlet), 3 (Neumann), 4 (variable Concentration), 5 (variable Q), 6 (atmospheric) top.value <b>{top value}</b> value of the boundary condition if appropriate bottom.bound <b>{bottom}</b> boundary condition for the bottom: 1 (free drainage), 2 (deep drainage), 3 (impermeable) bottom.value <b>{bottom value}</b> value of the boundary condition if appropriate root.uptake <b>{root uptake}</b> - not implemented yet intern.source <b>{internal source}</b> - not implemented yet
atmosphere	{atmosphere, control} list of the following components AtmInf <b>{use atmospheric data}</b> logical. Determines whether or not atmospheric data should be used.

	<p>tInit <b>{start time of simu}</b> starting time of the simulation. This information is necessary since the atm.data (see below) give the end of an atmospheric period only. So for the first period the starting time is missing.</p> <p>Aqh <b>{<math>A_{qh}</math>}</b> only used if water\$bottom.bound equals 2 (deep drainage). Then the modulus of the discharge rate equals <math>\text{step} * A_{qh} \exp(B_{qh}  h - \text{GWL0L} )</math> where <math>h</math> is the local pressure head.</p> <p>Bqh <b>{<math>B_{qh}</math>}</b> see Aqh</p> <p>hCritS <b>{max pressure at surface}</b> maximum allowed pressure head at the soil surface</p> <p>GWL0L <b>{ref. pos of groundwater}</b> Reference position of the groundwater table (usually the z-coordinate of the soil surface); only used if water\$bottom.bound equals 2 (deep drainage)</p> <p>rLen <b>{width of soil (transpiration)}</b> width of soil surface associated with transpiration. Only used in case of atmospheric root uptake.</p>
atm.periods	(maximum) number of atmospherical periods. Only used if atm.data is a vector. Otherwise atm.periods is set to the number of the rows of atm.data.
atm.data	<p>{atmosphere, data} vector of 10 components or matrix of 10 columns. The 10 components are</p> <p>end <b>{end point}</b> end point of the atmospherical period; the value of end for the last atmospherical period must be greater than the value of TPrint</p> <p>prec <b>{precipitation}</b> precipitation</p> <p>cPrec <b>{solute, precipitation}</b> solute concentration of rainfall water</p> <p>rSoil <b>{potential evaporation}</b> potential evaporation rate</p> <p>rRoot <b>{potential transpiration}</b> potential transpiration rate</p> <p>hCritA <b>{abs. min. pressure at surface}</b> absolute value of the minimum allowed pressure head at the soil surface</p> <p>rGWL <b>{drainage flux (drain or var. Q)}</b> drainage flux or other time-dependent prescribed flux boundary condition. For nodes with atmospheric flux condition.</p> <p>GWL <b>{ground water level}</b> groundwater level or other time-dependent prescribed head boundary condition. For nodes with atmospheric head condition. Pressure is then <math>\text{GWL} + \text{GWL0L}</math>.</p> <p>crt <b>{conc. flux (drainage)}</b> time-dependent concentration flux</p> <p>cht <b>{conc. 'pressure' (drain/var. H)}</b> time-dependent concentration of the first-type boundary condition. See the SWMS2D manual for details.</p>
stone	<p>{stones} list of the following components</p> <p>value <b>{value}</b> numeric or NA. NA is used to model stones, i.e. water cannot penetrate. If not NA, the (Gaussian) random field gets, at the place of the 'stone', the value value. This might be of interest if small lenses are modelled by means of 'stones'.</p> <p>lambda <b>{intensity}</b> intensity of the stones</p> <p>no.upper <b>{!overlap into upper horizons}</b> if TRUE overlap into any upper horizon is forbidden</p> <p>no.lower <b>{!overlap into lower horizons}</b> if TRUE overlap into any lower horizon is forbidden</p>

`no.overlap` **{!overlap of stones}** if TRUE the overlap of stones is forbidden. This point is clear from a practical point of view. However the currently implemented algorithm (SSI/RSA) for non-overlapping stones is slow and it is furthermore not guaranteed that it will not fail (due to 'bad' random numbers). If overlapping is allowed a fast and simple algorithm is used (Boolean model). See for both, SSI and Boolean model, the references in Stoyan et al., for example.

`main.distr` distribution for the length of the main axis of the ellipse

`main.mean` **{main axis, mean}** the parameter for the mean of the distribution

`main.s` **{main axis, sd}** the parameter for the standard deviation of the distribution

`sec.distr` distribution for the length of the second axis of the ellipse

`sec.mean` **{second axis, mean}** the parameter for the mean of the distribution

`sec.s` **{second axis, sd}** the parameter for the standard deviation of the distribution

`phi.distr` distribution for the angle between the horizontal line and the main axis of the ellipse

`phi.mean` **{second axis, mean}** the parameter for the mean of the distribution

`phi.s` **{second axis, sd}** the parameter for the standard deviation of the distribution

`plant.types` positive integer. Number of different types of root systems that will be generated

`Kf` `function(plant, distances, depths, rf, param=list(field.value=1, field.factor=0))`. According to the parameters the (Gaussian) random field get a new value if there is a root pixel. Here, `plant` is the type of plant (scalar integer value `1..plant.types`), `distances` a vector of distances from the considered locations to the surface along the root, `depth` is a vector of depths of the considered locations, and `rf` is the value of the Gaussian random field (including the stones). `param` is a list of named elements (possibly empty) whose values can be modified interactively; all parameters are real valued and the usual magnitude of the range is 1; the names may not match any name in the list for root, since the parameters are included in the root list, which is passed to `Kf` as a whole.

The function is only used if `root$rf.Kf` is TRUE for the specific plant type. This function is still in an experimental stage.

`beta` `function(plant, distances, depths, rf, param= list(beta.value=1, beta.factor=0))`. `beta` gives the raw potential root water uptake according to a single plant type, the distances to the beginning of the root, the depths, and `rf` (see also `Kf`). The raw potential values are subsequently normed such that they sum up to one. This gives the potential root water uptake. `param` is a list of named elements (possibly empty) whose values can be can be modified interactively; all parameters are real valued and the usual magnitude of the range is 1; the names may not match any name in the list for root or any parameter name within the parameter list in `Kf`, since the parameters are included into the root list, which is passed to `beta` as a whole.

root {root growth} for plants.lambd to dir.ch.s and {root, water uptake} otherwise.  
 Any plant type is set to the values of root at starting time. root is a list of the following components

plants.lambd {**mean \# of plants**} the number of plants is Poisson distributed with mean plants.lambd

plants.mindist {**min. plant. dist.**} plants of the same species are at least plants.mindist away. Actually, the algorithm tries to find random positions so that this condition is satisfied, but will give up if not successful after a few attempts. Then the value of plants.mindist is ignored.

mean {**aimed mean total length**} mean total length of all the roots of a single plant. For a long time run, the average will be about the mean, but the algorithm is not too sophisticated to guarantee this.

sd {**aimed sd of total length**} standard deviation for the total length of all the roots of a single plant. There is a check by the algorithm to be close to sd

knot.probab {**knot probability**} probability of any root pixel to be a knot, except if the distance from the position to the previous knot is less than knot.mindist. Then the probability for a knot is 0.

knot.mindist {**min. knot distance**} see knot.probab

shoots.3 {**3 shoots probab.**} positive number, see shoots.4

shoots.4 {**4 shoots probab.**} positive number.  
 shoots.4  $\geq 1$  knots have always 4 shoots.  
 shoots.4  $< 1$  and shoots.4+shoots.3  $\geq 1$  knots have with probability shoots.4 4 shots and 3 shoots otherwise.  
 shoots.4  $< 1$  and shoots.4+shoots.3  $\geq 1$  knots have 4, 3, 2 shoots with probability shoots.4, shoots.3 and 1-shoots.4-shoots.3, respectively

stop.probab function(knot, dist, stop.m, stop.s) that returns a values in [0, 1], which is the probability that a given root pixel does not continue to grow. The function should be able to take matrices for the first 2 components. knot is the number of knots preceding and including the current pixel, dist is the distance to the surface along the root, stop.m and stop.s are arbitrary additional parameters.

stop.m {**stop probability, m**} see stop.probab

stop.s {**stop probability, s**} see stop.probab

rf.link function(v, rf.m, rf.s). Summand in calculating the priority for all the neighbouring pixels of all active root pixels (end points of the roots). The pixels with the highest priority will be the next new root parts. rf.link transforms the value of the simulated Gaussian random field where rf.m and rf.s are additional parameters.

rf.m {**random field link, m**} see rf.link

rf.s {**random field link, s**} see rf.link

no.own.root {**own root penetration**} if TRUE a pixel may contain at most one root part of the same plant.

age.bonus {**age bonus**} factor by which the priority number of a pixel is multiplied after each step. The age.bonus is usually greater than or equal to 1.

- depth.bonus **{depth bonus}** summand added to the priority number if the subsequent pixel is below the current root pixel
- side.bonus **{side bonus}** summand added to the priority if the subsequent pixel has the same depth than the current root pixel; see depth.bonus for deeper points.
- diagonal **{diagonal directions}** if TRUE then roots may also grow into diagonal directions.
- dir.ch **{no direction change}** The number of pixels in x direction and the number of pixels in y direction by which the new direction of the root growth deviates from the previous is added and gives a value  $d$  between 0 and 4. E.g., if the sequence of root pixels is (1,3), (2,4) then the previous direction is (1,1). If the pixel (2,3) is considered the new direction is (0, -1), so the modulus of deviation is 1 and 2 in x and y direction, respectively. So,  $d = 1 + 2 = 3$ .  
Let  $v = d \text{dir.ch}$ . Then the summand for the priority of a pixel is given by normal random variable with mean  $v$  and  $sd = |v| \text{dir.ch.s}$ .
- dir.ch.s **{no dir. change, rel. sd}** see dir.ch
- rf.Kf **{root, water uptake}{root changes field value}** logical. If TRUE the value of the (Gaussian) random field at a root segment is changed according to the function Kf.
- field.value **{root, water uptake}{field.value}** The list entries field.value and field.factor are present only if the function Kf has the default definition. In general, the named list elements of the parameter param of Kf are given. See also Kf.
- field.factor **{root, water uptake}{field.factor}** see field.value
- beta.value **{root, water uptake}{beta.value}** The list entries beta.value and beta.factor are present only if the function beta has the default definition. In general, the named list elements of the parameter param of beta are given. See also beta.
- beta.factor **{root, water uptake}{beta.factor}** see beta.value
- P0 **{root, water uptake}{P0}** (negative) pressure above which too less oxygene to allow water uptake by plants
- P2H **{root, water uptake}{P2H}** P2H, P2L, r2H and r2L determine, in dependence of the potential transpiration, the lower bound for optimal water uptake (piecewise linear curves); the upper bound of optimal water uptake is given by materials\$P0<sub>optm</sub>. See the SWMS2D manual for details.
- P2L **{root, water uptake}{P2L}** see P2H
- P3 **{root, water uptake}{P3}** (negative) pressure below which no water uptake is possible
- r2H **{root, water uptake}{r2H}** see P2H
- r2L **{root, water uptake}{r2L}** see P2H
- root.condition **{root, water uptake}{root water uptake}** boundary condition for the root. Orginally, only 'atmospheric' was allowed.
- root.uptake **{root, water uptake}{water uptake value}** value at the root in case of 'dirichlet' or 'neumann' boundary condition

root.zone	NULL or function of one variable. If not NULL then rootzone defines for any given $x$ coordinate the interval for the height of the rootzone. A plant of species 1 is defined so that all segments within the so defined root zone contains a root of this plant.
col.rf	sequence of colours that are used to show the random field; if NULL then some internally defined color sequence is used
col.simu	sequence of colours that are used to present any simulation result by swms2d; if NULL then some internally defined color sequence is used
MaxIt	maximum number of iteration during any time step.
hTab	$c(hTab1, hTabN)$ , interval of pressure heads within which a table of hydraulic properties is generated.
DMu1	$c(dMu1, dMu2)$ , $dMu1 \geq 1$ , $dMu2 \leq 1$ ; if the number of required iterations is less than 4 or greater than 6 then the next time step is multiplied by $dMu1$ and $dMu2$ , respectively.
col.rect	colour of the button for free input; see <a href="#">eval.parameters</a>
col.bg	colour of a interactive bar; see <a href="#">eval.parameters</a>
col.sep	colour of the separating line; see <a href="#">eval.parameters</a>
col.left	colour of preceding element; see <a href="#">eval.parameters</a>
col.mid	colour for the message; see <a href="#">eval.parameters</a>
col.right	colour of subsequent element; see <a href="#">eval.parameters</a>
col.line	colour of the marking line in interactive bars of absolute choice; see <a href="#">eval.parameters</a>
col.txt	colour of headers; see <a href="#">eval.parameters</a>
col.submenu	colour for the text in the main menue
col.subord	colour for the text in the title of a secondary menue
col.forbid	colour for text that indicate forbidden main menues, e.g. 'polygon' after having defined already the maximum number of allowed horizons
col.bg.forbid	background colour for forbidden menue points
col.flash	colour for hints or the titles of active windows
col.hor	vector of 10 components. Colours that are used to draw the horizons
col.draw	colour for drawing or showing selected horizons
col.mess	colour for warning or error messages
col.mesh	colour for the edges of the finite element mesh
col.mesh.pt	colour for the vertices of the finite element mesh
col.exception	vector of 2 components; colour plotted values that are below or above the given range of $zlim$ , see <code>lim.swms2s</code> and <code>zlim</code> .
cex.eval	value for the parameters <code>cex</code> and <code>cex.i</code> of <a href="#">eval.parameters</a> and for the parameter <code>cex.names</code> in <a href="#">ShowModels</a> .
areas	logical. If TRUE the horizons are coloured. If FALSE only the border lines between the horizons are given.
PrintLevel	If $\leq 0$ nothing is printed. The higher the value the more information is given.

<code>new</code>	logical or NULL. If TRUE the interactive plot is opened in a new window. If NULL then the interactive window is not opened and the standard definitions are returned.
<code>X11.width</code>	numeric. Width of the window. Only used if <code>new=TRUE</code> .
<code>X11.height</code>	numeric. Height of the window. Only used if <code>new=TRUE</code> .
<code>frequent.reset</code>	logical. Background: the current implementation of <code>xswms2d</code> uses <code>split.screen</code> to build the interactive plot; <code>split.screen</code> memorises all changings in the plot and replays them all if the X11 window has to be rebuilt. If <code>frequent.reset=TRUE</code> then the memory effect is reduced at the cost of some flickering of the X11 window in the main menu.
<code>update</code>	{updating} logical. If TRUE the simulations are updated after any changing of the parameters. Independently of the value of <code>update</code> an update of the simulation is performed if the user clicks on any of the given titles or subtitles of the submenu.
<code>waterflow</code>	{water flow} logical. If TRUE <code>swms2d</code> is run
<code>zlim</code>	vector of 2 components or NULL which gives the limits for the plotted values in the <code>images</code> . If NULL then <code>zlim</code> is calculated internally, cf. <code>plotWater</code> .
<code>Zlim</code>	vector of 2 components or NULL which gives the limits for the plotted values in <code>ShowModels</code> .
<code>print.par</code>	{postscript} list of the following components <code>ps</code> {ps base name} character. Base name of the postscript file. <code>height</code> {picture height} numeric. Height of the postscript figure in inches. <code>title</code> {plot title} logical. If TRUE a title is plotted. <code>legend</code> {legend} logical. If TRUE a legend is added to the plot.
<code>...</code>	<code>sophy</code> is alias for <code>xswms2d</code> and takes the same arguments as <code>xswms2d</code> .

## Details

The **interactive plot** is organised as follows:

**left top** drawing area: here, a soil profile is shown (if available, see parameter `picture`) and the user draws the boundaries of the horizons by a sequence of left mouse button clicks.

**left bottom** Random field of hydraulic conductivity. That is, the Gaussian random field including stones and roots, to which `miller.link` and then `millerK` has been applied. (In the standard setting, the modulus of the Gaussian random field is anti-proportional to the root of the hydraulic conductivity  $K$  and proportional to the initial water potential  $H$ .)

**centre top** Main menu; this area is also used for defining the parameters of the Gaussian random fields

**centre bottom** plotting area for the water pressure  $H$ , the charge rate  $Q$ , water contents  $\theta$ , the flux vector components or the solute concentration as calculated by `swms2d` for coarsened grids.

**right** area used by submenus

**right bottom** plotting area for the water pressure etc for the original, fine grid, see main menu point 'precise waterflow' below

**The menu points** of the interactive plot are

**postscript** plotting the water potential  $H$ , discharge/recharge rates  $Q$  for internal sink/sources, the water contents  $\theta$ , the x-components of the Darcian flux vector  $v_x$ , the z-components of the Darcian flux vector  $v_z$ , or the concentration; in grey it is noted whether the ‘precise water flow’ or the result of the approximate ‘water flow’ is given (according to the main menu point ‘precise waterflow’). Further the random field might be plotted. For the additional parameters, see the input variable `print.par`.

**polygon** A sequence of left mouse button clicks in the upper left drawing area defines the vertices of the polygon. Up to 512 vertices are allowed. The right mouse button terminates the drawing. A polygon must consist of at least three points. The points may be placed such that the connecting lines cross.

**horizon** A sequence of left mouse button clicks in the upper left drawing area defines the border of the horizon. Up to 512 vertices are allowed. The right mouse button terminates the drawing. A sequence of chosen points is only accepted to define a horizon if the first point has a smaller x-coordinate than all the other points and the last point has a greater x-coordinate than all the other points; except for this restriction and the fact that at least 2 points must be given, any sequence of points is allowed, points may even leave the plotting area. If the first or the last point is within the area then the horizon boundary is linearly extrapolated by a line through the first and second or the two last points, respectively. The new horizon gets as default parameters for the stones the input stone parameters, and for the material the material parameters of the preceding horizon.

**structure** Definition of the random fields that underly the Miller similar medium. If more than one horizons or at least one polygon is defined the user selects first the part the user likes to define by a click into the appropriate part of the drawing area. Afterwards a new menu is opened in the upper right part of the window. Within this new menu we have, see [ShowModels](#) for details,

**bottom left** a simulation of the Gaussian random field with the specified parameters.

**top left** the covariance function or the variogram for the random field and the transformed random field according to `miller.link`. If the model is anisotrope or `anisotrop=TRUE` is given explicetely, the models are shown for the two main axis.

**right** Interactive menu. From top to bottom:

- ‘simulate’ is given if ‘updating’ of the main menu is FALSE
- name of the model
- the formula. Note the formula is not given for all the coded variogram models.
- the specific parameters if there are any
- variance
- nugget effect
- the scale parameter or the anisotropy paramters
- mean.
- If ‘practical range’=TRUE then the (isotropic) covariance model is rescaled such that at distance 1 the value of the model is approximately 0.05.
- ‘variogram’ switches between the presentation of the variogram and the corresponding covariance function.
- ‘variogram angle’ gives the angle for the direction of the main variogram. If ‘variogram angle’=NA the angle equals the angle of the anisotropy specification.

- If 'show transformed field'=TRUE then the same transformation as for the hydraulic conductivity parameter  $\alpha_k$  is applied, i.e., `millerK(miller.link( ))`

This menu is left by the right mouse button. A menu for choosing another variogram model is reached. Again, the right mouse button leaves the menu.

The first horizon gets as default the model and the parameter values passed to xswms2d by model and anisotropy. For the following horizons the values of the previously considered horizon are taken over, when considered first.

**stones** definition of the stones

**root growth** definition of the root growth; if `plant.types` is greater than 1, the user first chooses among the different types, then enters the submenu. 'plant type' is used to name a species only, and is used nowhere else. The description of all the other variables ('mean \# of plants', ..., 'no dir. change, rel. sd') is given above; see the input variable `root`.

**material[phys ]** Definition of the physical material constants, see variable `materials`.

**material[chem ]** Definition of the chemical material constants, see variable `materials`.

**root water uptake** Definition of the water uptake by the roots. See the variable `root`.

**atmosphere, data** See the variable `atm.data` for a description. If `nrow(atm.data)` or `atm.periods` is greater than 1 then the user chooses the atmospherical period first.

**swms2d [water ]** see the variable `water` for a description

**swms2d [chem ]** see the variable `chemical` for a description

**atmosphere control** see the variable `atmosphere` for a description

**new simulation** If parameters are changed, simulations are redone reusing previous simulations as much as possible. That is, if a parameter of SWMS2D is changed the simulation of the Gaussian random field is reused. Further the redone simulations are based on the old random seeds.

Here, a simulation is redone from scratch, based on a new random seed.

**precise waterflow** By 'swms2d water', submenu 'size reduction' (variable `water$red`) the finite element mesh can be coarsened for faster simulations. If the coarsening is genuine, i.e. `water$red > 1`, then this menu point is active. If pressed, a simulation is performed on the original finite element mesh.

**water flow** yes/no. If 'no', only the Gaussian random field, the stones and the roots are simulated.

**updating** yes/no. If yes the simulation is redone after any changing of any parameter, except for 'structure' definition, where the new simulation is performed when the submenu 'structure' is left.

**end** terminates the interactive surface

### General information

- Currently, the number of horizons is limited to `h$max.horizon=10`.
- The horizons are build up from the bottom, i.e. first, the boundary for the C horizon should be drawn then those for the B horizons, etc.
- If the mean of the last genuine(!) horizon (polygons excluded) is defined NA within the menu 'structure', then this part of the profile is interpreted as air and the previously defined horizons are genuine ones. Any other horizon but the last one should not have mean NA.

- Polygons are used to define large lenses or to circumvent the predefined ordering of horizons.
- The ordering (as the sequence in which the horizons and polygons are defined, not their position in the profile) of the horizons and polygons is used in the conditional simulation of the Gaussian random fields, where conditioning happens with respect to preceding horizons.
- First the Gaussian random fields are simulated, starting with the first horizon. After simulation of the complete profile, the stones are simulated, then the roots. If 'water flow'='yes' then the SWMS2D simulation is performed. See [simulateHorizons](#) and the references therein for further details on the stochastic simulation. See the SWMS2D manuscript for details on SWMS2D.
- If a parameter was changed, usually only a part of the simulations is redone, namely that level of simulation that corresponds to the parameter (random field for a specific horizon, stones, roots, SWMS2D) and all subsequent levels. For example, if a stone parameter has been changed, the simulation of the stones and the roots is done, and SWMS2D is performed.

## Value

sophy is a synonym for xswms2d. A list is returned, where the first 10 components contain the definitions of the horizons; these 10 components are called by their position in the list. Any other list component does not have a predefined location and may be called only by name.

Note that if sophy or xswms2d is called with parameter new=NULL the interactive window is not opened, but the initial list (of h) is returned immediately.

- 1:10            Each of the first 10 components is a list of the following elements:
- type    character. Either 'Start', 'H', or 'P'. Exactly the first element has the value 'Start', all the others may be horizons ('H') or polygons ('P').
  - cut.x, cut.y    cut.x and cut.y define the horizontal and vertical position of a rectangle, respective, which encloses the whole horizon or polygon. cut.x and cut.y should be as tight as possible, to accelerate the simulation of the random fields for the horizons.
  - stone    list of stone parameters, see the input variable stone. The input parameters are the default parameters for all stone specifications
  - materials    list of material constants, see the input variable materials. The default parameters for the first horizon are the input parameters; the default parameters for any other horizon (or polygon) are the parameters of the preceding horizon (or polygon)
  - model    list or NULL. Covariance model in list format, see [CovarianceFct](#). The model is NULL if 'structure' of the main menu has not been called yet for the respective horizon. The 'Start' horizon gets the value of the input parameter at starting time. If 'undo' is called when only the 'Start' horizon exists then model is set to NULL even for the 'Start' horizon.
  - border    matrix of two columns or NULL. boundary points between two horizons (or polygons); used for the conditional simulation of the Gaussian random fields. Further, if for the selection of a horizon the user clicks on such a boundary points, the user is warned. border is NULL if only the 'Start' horizon exists.

	<p>idx matrix of logical values with <math>\text{diff}(\text{cut.x})+1</math> rows and <math>\text{diff}(\text{cut.y})+1</math> columns. idx indicates which pixels of the <math>\text{cut.x} \times \text{cut.y}</math> area belongs to the present horizon.</p>
grid.x	seq(xlim[1], xlim[2], step)
grid.y	seq(ylim[1], xlim[2], step)
idx.rf	<p>integer. Matrix of size <math>\text{length}(\text{grid.x}) \times \text{length}(\text{grid.y})</math> indicates the affiliation to a horizon or polygon (number is decremented by 1, i.e., 0, ..., <math>h\\$n - 1</math>), see also the output variable border within the specific horizons.</p> <p>It allows also for the indication of the presence of a stone; then, if a stone is present, the horizon number is incremented by <math>h\\$max.horizon</math>. The modulus by <math>h\\$max.horizon</math> gives the by 1 decremented number of the horizon.</p>
n	current number of horizons
step	input parameter step
max.horizons	maximum number of horizons; currently 10. The value of this variable may never be changed!
beta	input parameter beta
root	<p>list, where <math>\text{length}(\text{root})</math> equals the input parameter <code>plant.types</code>. Each component of the list is a list with the same components as the input parameter <code>root</code>. Additionally, it has the components</p> <ul style="list-style-type: none"> <li>• <code>plant.type</code>, a user defined name of the species.</li> <li>• the parameters within the <code>param</code> list of the function <code>Kf</code></li> <li>• the parameters within the <code>param</code> list of the function <code>beta</code></li> </ul>
atmophere	input parameter atmosphere
atm.data	matrix of 10 columns, corresponding to the input parameter <code>atm.data</code> and <code>atm.periods</code> .
water	input parameter water
chem	input parameter chemical
Kf	input parameter Kf
RF	matrix of size $\text{length}(\text{grid.x}) \times \text{length}(\text{grid.y})$ or NULL. RF contains the Gaussian random field resulting from all definitions for the horizons. RF is NULL if <code>simulate.horizon</code> has not been called yet.
Stones.RF	<p>matrix of size <math>\text{length}(\text{grid.x}) \times \text{length}(\text{grid.y})</math> or NULL. The value of <code>Stones.RF</code> is the Gaussian random field modified by the stones (concerning hydraulic conductivity). Stones have usually value NA, whereas air has value NaN.</p> <p><code>Stones.RF</code> is NULL if <code>simulate.horizon</code> has not been called yet or terminated with an error. Further <code>Stones.RF</code> is set to NULL if <code>materials\$sharpness</code> has been changed, or 'structure', 'undo', 'horizon', or 'polygon' has been called.</p>
Root.RF	<p>matrix of size <math>\text{length}(\text{grid.x}) \times \text{length}(\text{grid.y})</math> or NULL. The value of <code>Root.RF</code> is the value of <code>Stones.RF</code> modified by the roots (concerning hydraulic conductivity). That is, if <code>root\$rf.Kf</code> is TRUE the values at the respective root segments are given by <math>h\\$KF</math>.</p> <p><code>Root.RF</code> is NULL if <code>simulate.horizon</code> has not been called yet or terminated with an error. Further <code>Root.RF</code> is set to NULL if <code>materials\$sharpness</code> has been changed, or 'structure', 'undo', 'horizon', or 'polygon' has been called.</p>

<code>m.link</code>	link function used in the submenu 'structure', currently the function is identical to <code>miller.link</code>
<code>miller.link</code>	input parameter <code>miller.link</code>
<code>col.rf</code>	input parameter <code>col.rf</code> if not NULL and the internally defined sequence of colors otherwise
<code>col.simu</code>	input parameter <code>col.simu</code> if not NULL and the internally defined sequence of colors otherwise
<code>random.seed</code>	list or NULL. Each element contains the random seed for the simulation of the Gaussian random field in the respective horizon. <code>random.seed</code> is NULL if <code>simulateHorizons</code> has not been called yet.
<code>stone.random.seed</code>	The random seed for the simulation of the stones
<code>root.random.seed</code>	The random seed for the simulation of the roots
<code>rf.complete</code>	logical. TRUE if the stochastic simulation has been performed completely.
<code>plants</code>	list or NULL. Each component contains the information on the roots of a single plant. Each component is a matrix of 8 columns and rows according to the number of root segments. The 8 columns are <ol style="list-style-type: none"> <li>1. horizontal coordinate of a root segment. The coordinate is given in pixel units.</li> <li>2. vertical coordinate of a root segment. For the first segment it equals 1 plus the vertical position of the first pixel that is not air at the chosen horizontal position.</li> <li>3. row index for the parent root pixel</li> <li>4. row index where the subsequent root pixel is found; NA if it is a terminating root segment. If the pixel is a knot with <math>k</math> children then the value gives the position of the first child. The other children are given in consecutive rows.</li> <li>5. number of children in case it is a knot, 1 otherwise</li> <li>6. the number of preceding knots until and including this position</li> <li>7. aimed random total length of the roots if it is the first pixel, and the current distance to the surface along the roots, otherwise.</li> <li>8. distance to the preceding knot</li> </ol> <p><code>plants</code> is NULL if <code>simulateHorizons</code> has not been called yet, or any parameter in <code>root</code> has been changed.</p>
<code>plants.idx</code>	vector or NULL. <code>plant.idx[i]</code> gives the plant type (1, ..., <code>plant.types</code> ) of the $i$ th simulated plant. <code>plants.idx</code> is NULL if <code>create.roots</code> or <code>simulateHorizons</code> has not been called yet.
<code>water.x</code>	the coordinates of <code>grid.x</code> after thinning by factor <code>water\$red</code>
<code>water.y</code>	the coordinates of <code>grid.y</code> after thinning by factor <code>water\$red</code>
<code>flux</code>	logical vector of length( <code>water.x</code> ) $\times$ length( <code>water.y</code> ) elements. <code>flux</code> indicates which pixels of the length( <code>water.x</code> ) $\times$ length( <code>water.y</code> ) grid are used in the SWMS2D simulation. (Pixel are left out if they indicate stones (NA) or air (NaN).)

swms2d three dimensional array or NULL. The first dimension has 6 components which are

1. the water potential  $H$
2. discharge/recharge rates  $Q$  for internal sink/sources
3. the water contents  $\theta$
4. the x-components of the Darcian flux vector  $v_x$
5. the z-components of the Darcian flux vector  $v_z$
6. the concentration

The second component corresponds to the sequence of pixels used in the SWMS2D simulation and has length `sum(flux)`.

swms2s is NULL if the water flux has not been simulated yet because no simulation has been performed or one of the stochastic simulations (Gaussian random field, stones, roots) failed, or 'water flow' has been set to 'no'. Further swms2d is set to NULL if an error occurs when trying to plot the SWMS2D simulation results (by `plotWater`); this happens if the result has no finite values. Finally, swms2d is set to NULL if a relevant parameters was changed, such as those in `atmosphere`, `atm.data`, `materials`, `chem` or `water`.

Any of the lists that contain input parameters may have the additional component `.history`, an internal variable used by the interactive menu algorithm that gives the last entries by the user; see `eval.parameters`.

### Warning

The user should consider the following parameters of `h[[i]]` as being read-only:

- `cut.x`
- `cut.y`
- `border`
- `idx`

Further, `h$max.horizon` may never be changed.

### Note

The phrases in brackets in the argument section of this documents give the respective menu points of the interactive plot.

### Author(s)

Martin Schlather, <[martin.schlather@math.uni-goettingen.de](mailto:martin.schlather@math.uni-goettingen.de)> <http://www.stochastik.math.uni-goettingen.de/~schlather>

## References

- Simunek, J., Vogel, T., and van Genuchten, M.Th. (1994) *The SWMS2D code for simulating water flow and solute transport in two-dimensional variably saturated media, Version 1.21*. Research Report No. 132, 197 p., U.S. Salinity Laboratory, USDA, ARS, Riverside, California.
- Stoyan, D., Kendall, W.S., Mecke, J. (1995) *Stochastic Geometry and its Applications* Chichester: Wiley, 2nd edition.
- Schlather, M., Huwe, B. (2005) SOPHY - an interactive programme for water flow modelling. *In preparation*

## See Also

[calculate.horizons](#), [create.roots](#), [create.stones](#), [create.waterflow](#), [draw.horizons](#), [modify.horizons](#), [plotRF](#), [plotWater](#), [simulateHorizons](#), [SoPhy](#), [swms2d](#)

## Examples

```
## Not run:
### without underlying profile
h0 <- xswms2d(xlim=c(1, 100), ylim=c(1, 100), step=1)
# or, equivalently:
# h1 <- sophy(xlim=c(1, 100), ylim=c(1, 100), step=1)

### underlying profile
### the profile was taken by M. Flury, J. Leuenberger,
### B. Studer, W.A. Jury and H. Flühler, see also URL
### \url{http://www.ito.umw.ethz.ch/SoilPhys/Fliessmuster/projekt_flury.html}
pic <- paste(system.file(package='SoPhy'), 'tracer', 'K06', sep="/")
h <- xswms2d(xlim=c(0,160), step=1, aniso=TRUE,
            update=FALSE, waterflow=FALSE, pict=pic)

### repeated call
h <- xswms2d(h=h, update=TRUE, waterflow=TRUE)

### an example for non-atmospheric root uptake
h <- sophy(xl=c(1,10), yl=c(1,10), step=0.1, new=NULL) ## get standard
h$root[[1]]$root.condition <- 1 ## dirichlet condition for roots
h$root[[1]]$root.uptake <- -200 ## value for dirichlet condition
h$root[[1]]$plants.lambda <- 0.07 ## intensity of plants
h$water$TPrint <- 2 ## end point of simulation (film lasts about 2 min)
h$water$red <- 1 ## precise simulation
```

```
h$water$breakpoint <- 3 ## high image frequency (close to a film)
h$water$top.bound <- 2 ## dirichlet
h$water$top.value <- -8 ## value on the boundary
h <- xswms2d(h=h, update=TRUE, waterflow=TRUE)

## End(Not run)
```

# Index

- \*Topic **IO**
  - read.picture, 41
  - write.picture, 66
- \*Topic **datasets**
  - Tracer experiments, 62
- \*Topic **distribution**
  - Pareto, 31
- \*Topic **file**
  - read.picture, 41
  - write.picture, 66
- \*Topic **hplot**
  - my.legend, 30
  - plotFlow, 32
  - plotRF, 34
  - plotRGB, 36
  - plotWater, 37
  - Quader, 39
- \*Topic **iplot**
  - Locator, 27
- \*Topic **spatial**
  - ADE, 2
  - analyse.profile, 3
  - calculate.horizons, 9
  - create.roots, 10
  - create.stones, 13
  - create.waterflow, 15
  - draw.horizons, 17
  - eval.parameters, 18
  - flowpattern, 22
  - modify.horizons, 28
  - read.swms2d.table, 42
  - risk.index, 46
  - sh, 49
  - simulateHorizons, 51
  - SoPhy, 53
  - swms2d, 55
  - tortuosity, 60
  - xswms2d, 67
- \*Topic **utilities**
  - getactions, 27
  - Locator, 27
  - Readline, 45
  - useraction, 63
- ADE, 2
- analyse.profile, 3, 49, 54
- belichtet (Quader), 39
- BrilliantBlue, 54
- BrilliantBlue (Tracer experiments), 62
- brilliantblue (Tracer experiments), 62
- calculate.horizons, 9, 29, 54, 86
- CDE (ADE), 2
- CovarianceFct, 23, 72, 82
- create.roots, 10, 52, 84, 86
- create.stones, 13, 52, 86
- create.waterflow, 15, 37, 70, 86
- Dev, 32, 50
- dpareto (Pareto), 31
- draw.horizons, 10, 17, 86
- eval.parameters, 18, 65, 78, 85
- F04 (Tracer experiments), 62
- F06 (Tracer experiments), 62
- flowpattern, 22, 33, 34, 54
- GaussRF, 24
- getactionlist (getactions), 27
- getactions, 27, 28, 46, 64, 65
- image, 36, 79
- K06 (Tracer experiments), 62
- Locator, 27, 27, 46, 63–65
- locator, 28, 64
- modify.horizons, 10, 16, 18, 28, 86

- my.legend, 30, 54
- palettes, 30
- par, 18, 32, 36, 40
- Pareto, 31, 54
- pareto (Pareto), 31
- plotFlow, 32
- plotFlow2d, 26, 51
- plotFlow2d (plotFlow), 32
- plotFlow3d, 26, 51
- plotFlow3d (plotFlow), 32
- plotRF, 10, 16, 34, 39, 86
- plotRGB, 36, 42, 54, 67
- plotWater, 10, 16, 35, 37, 70, 79, 85, 86
- pos3D (Quader), 39
- ppareto (Pareto), 31
- putactions (useraction), 63
  
- qpareto (Pareto), 31
- Quader, 39
- quader, 54
- quader (Quader), 39
  
- RandomFields, 11, 26, 48
- read.picture, 4, 36, 41, 54, 67
- read.swms2d.table, 42, 59, 70
- Readline, 27, 28, 45, 63–65
- readline, 46, 64
- rev, 40
- RFMethods, 24
- RFparameters, 11, 48
- risk.index, 5, 8, 23–25, 46, 50, 54
- rpareto (Pareto), 31
  
- sh, 49
- sh.jch, 54
- sh.jh, 54
- ShowModels, 78–80
- simulateHorizons, 10, 16, 51, 82, 84, 86
- SoilPhysics (SoPhy), 53
- SOPHY (SoPhy), 53
- SoPhy, 8, 10, 13, 15, 16, 18, 29, 49, 51, 53, 53, 59, 86
- sophy (xswms2d), 67
- split.screen, 79
- Sweave, 11, 48
- SWMS2D (swms2d), 55
- swms2d, 10, 15, 16, 37, 45, 54, 55, 70, 86
  
- title, 35, 38
  
- tortuosity, 54, 60
- tracer (Tracer experiments), 62
- Tracer experiments, 62
  
- useraction, 20, 27, 28, 45, 46, 63
- userinput, 28, 46
  
- write.picture, 36, 42, 66
  
- xswms2d, 9–11, 13, 15–18, 29, 34, 35, 37–39, 49, 52–54, 58, 59, 67