Package 'ReacTran'

October 15, 2008

Version 1.1

Title Reactive transport modelling in 1D

Author Karline Soetaert <k.soetaert@nioo.knaw.nl>

Maintainer Karline Soetaert <k.soetaert@nioo.knaw.nl>

Depends R (>= 2.01), rootSolve, deSolve

Description Routines for developing models that describe reaction and advective-diffusive transport in one or two dimensions. Includes transport routines in porous media, in estuaries, and in bodies with variable shape.

License GPL

LazyData yes

R topics documented:

`OMEXDIAmodel` *application of the OMEXDIA diagenetic model - fortran DLL*

Description

A 1-D model of Carbon, nitrogen and oxygen diagenesis in a marine sediment.
The model describes six state variables, in **100** layers:

2 fractions of organic carbon (FDET,SDET): fast and slow decaying, solid substance

Oxygen (O2), dissolved substance

Nitrate (NO3), dissolved substance

Ammonium (NH3), dissolved substance

Oxygen demand unit (ODU), dissolved substance, as lump sum of all reduced substances other than ammonium

See Soetaert et al., 1996 for further details of the model.

Usage

```
OMEXDIAmodel(pars, grid, porgrid, Db)
```

Arguments

`pars`	The model parameters, a vector such as `OMEXDIAparms`
`grid`	The discretisation grid, e.g. generated with `setup.grid`; should contain 100 boxes
`porgrid`	The porosity grid, e.g. generated with `setup.prop`; should contain 100 boxes and be consistent with `grid`
`Db`	The bioturbation profile, defined on box interfaces, e.g. containing 101 values
`Dyna`	logical; if TRUE: also runs the model dynamically

Details

The model application starts by estimating the steady-state condition of the model, after which a dynamic simulation, with sinusoidal forcing of organic carbon deposition flux is run.

First one year of spinup is executed, and the final condition used as initial value for the subsequent run, again for one year and which is written to the output.

For efficiency reasons, the OMEXDIA diagenetic model was written in Fortran, and this code linked to the package. The Fortran code can be found in subdirectory 'dynload' of the packages subdirectory, ('omexdia.f') and an R-file showing how to compile this file to a dynamically linked library (DLL), and how to use this in R is also included ('omexdia.r').

To solve the model, a steady-state solver from package rootSolve (here we used `steady.band`) and an integration routines in package deSolve (we use `ode.band`) are used. The routines in these packages have been devised for linking with compiled code (here fortran).

The execution speed of this version of the model is similar to the original application where all code was in Fortran; it is almost two orders of magnitude faster than the implementation in R.

An implementation of the same model, in R can be found as one of the examples of `tran1D.solid`

Value

a list containing:

steady	The steady-state condition of the state variables, a vector containing steady-state concentrations of FDET(0-100), SDET(101-200), O2 (201-300), NO3 (301-400), NH3 (401-500) and ODU (501-600)
precis	Only if Dyna is TRUE: the precision of the steady-state solution
Solved	Only if Dyna is TRUE: a logical, TRUE when steady-state has been reached
times	the time values for which the model output has been produced
FDET	The concentration profile (columns) of Fast decaying organic carbon, one for each time value (rows)
SDET	The concentration profile (columns) of slow decaying organic carbon, one for each time value (rows)
O2	The concentration profile (columns) of oxygen, one for each time value (rows)
NO3	The concentration profile (columns) of nitrate, one for each time value (rows)
NH3	The concentration profile (columns) of ammonium, one for each time value (rows)
ODU	The concentration profile (columns) of ODU, one for each time value (rows)
O2flux	The sediment-water exhange rate of oxygen, one per time value
NO3flux	The sediment-water exhange rate of nitrate, one per time value
NH3flux	The sediment-water exhange rate of ammonium, one per time value
ODUflux	The sediment-water exhange rate of ODU, one per time value

Author(s)

Karline Soetaert <k.soetaert@nioo.knaw.nl>

References

Soetaert K, PMJ Herman and JJ Middelburg, 1996a. A model of early diagenetic processes from the shelf to abyssal depths. Geochimica Cosmochimica Acta, 60(6):1019-1040.

Soetaert K, PMJ Herman and JJ Middelburg, 1996b. Dynamic response of deep-sea sediments to seasonal variation: a model. Limnol. Oceanogr. 41(8): 1651-1668.

Examples

```
# 100 layers; total length=50 cm, first box=0.01 cm
N     <- 100
Grid  <- setup.grid(N=N,dx.1=0.01,len=50,type="exp")
Depth <- Grid$x                  # depth of each box

# porosity gradients
porGrid <- setup.prop(y.0=0.9,y.INF=0.7,y.coeff=2,grid=Grid)

# Bioturbation profile (at box interfaces)
biot <- 1/365          # cm2/d      - bioturbation coefficient
mixL <- 5              # cm         - depth of mixed layer
Db   <- setup.prop(y.0=biot,y.INF=0,y.coeff=1,L=mixL,grid=Grid)$int

#----------------------------------------
```

```
# Three dynamic runs, with different flux
#-----------------------------------------

# Application 1:
pars <- OMEXDIAparms
pars["MeanFlux"] <- 50000/12*100/365  # nmol/cm2/day
out1 <- OMEXDIAmodel(pars,Grid,porGrid,Db)

pars["MeanFlux"] <- 15000/12*100/365  # nmol/cm2/day
out2 <- OMEXDIAmodel(pars,Grid,porGrid,Db)

pars["MeanFlux"] <- 2000/12*100/365  # nmol/cm2/day
out3 <- OMEXDIAmodel(pars,Grid,porGrid,Db)

# Steady-state concentrations in sediment
CONC  <- cbind(out1$steady,out2$steady,out3$steady)

FDET  <- CONC[1:N,]
SDET  <- CONC[(N+1)  :(2*N),]
O2    <- CONC[(2*N+1):(3*N),]
NO3   <- CONC[(3*N+1):(4*N),]
NH3   <- CONC[(4*N+1):(5*N),]
ODU   <- CONC[(5*N+1):(6*N),]

TOC  <- (FDET+SDET)*1200/10^9/2.5     #

par(mfrow=c(2,2))
matplot(TOC,Depth,ylim=c(15,0),xlab="procent" ,main="TOC",
        type="l",lwd=2)
matplot(O2,Depth,ylim=c(15,0),xlab="mmol/m3" ,main="O2",
        type="l",lwd=2)
matplot(NO3,Depth,ylim=c(15,0),xlab="mmol/m3" ,main="NO3",
        type="l",lwd=2)
matplot(NH3,Depth,ylim=c(15,0),xlab="mmol/m3" ,main="NH3",
        type="l",lwd=2)

legend ("bottom",col=1:3,lty=1:3,lwd=2,
legend=c("15gC/m2/yr","50gC/m2/yr","2gC/m2/yr"),title="flux")

mtext(outer=TRUE,side=3,line=-2,cex=1.5,"OMEXDIAmodel")

#====================#
# Dynamic output     #
#====================#

par(mfrow=c(3,3))
par(oma = c(0,0,3,0))

femmecol<- colorRampPalette(c("#00007F", "blue", "#007FFF", "cyan",
                 "#7FFF7F", "yellow", "#FF7F00", "red", "#7F0000"))
times<-out1$times
image(x=times,y=Depth,z=out1$O2,col= femmecol(100),xlab="time, days",
        ylim=c(5,0),ylab= "Depth, cm",main="Oxygen, mmol/m3")
contour(x=times,y=Depth,z=out1$O2, add = TRUE)
image(x=times,y=Depth,z=out2$O2,col= femmecol(100),xlab="time, days",
        ylim=c(5,0),ylab= "Depth, cm",main="Oxygen, mmol/m3")
contour(x=times,y=Depth,z=out2$O2, add = TRUE)
```

```
image(x=times,y=Depth,z=out3$O2,col= femmecol(100),xlab="time, days",
        ylim=c(5,0),ylab= "Depth, cm",main="Oxygen, mmol/m3")
contour(x=times,y=Depth,z=out3$O2, add = TRUE)

image(x=times,y=Depth,z=out1$NO3,col= femmecol(100),xlab="time, days",
        ylim=c(15,0),ylab= "Depth, cm",main="Nitrate, mmol/m3")
contour(x=times,y=Depth,z=out1$NO3, add = TRUE)
image(x=times,y=Depth,z=out2$NO3,col= femmecol(100),xlab="time, days",
        ylim=c(15,0),ylab= "Depth, cm",main="Nitrate, mmol/m3")
contour(x=times,y=Depth,z=out2$NO3, add = TRUE)
image(x=times,y=Depth,z=out3$NO3,col= femmecol(100),xlab="time, days",
        ylim=c(15,0),ylab= "Depth, cm",main="Nitrate, mmol/m3")
contour(x=times,y=Depth,z=out3$NO3, add = TRUE)

image(x=times,y=Depth,z=out1$NH3,col= femmecol(100),xlab="time, days",
        ylim=c(15,0),ylab= "Depth, cm",main="Ammonium, mmol/m3")
contour(x=times,y=Depth,z=out1$NH3, add = TRUE)
image(x=times,y=Depth,z=out2$NH3,col= femmecol(100),xlab="time, days",
        ylim=c(15,0),ylab= "Depth, cm",main="Ammonium, mmol/m3")
contour(x=times,y=Depth,z=out2$NH3, add = TRUE)
image(x=times,y=Depth,z=out3$NH3,col= femmecol(100),xlab="time, days",
        ylim=c(15,0),ylab= "Depth, cm",main="Ammonium, mmol/m3")
contour(x=times,y=Depth,z=out3$NH3, add = TRUE)

mtext(outer=TRUE,side=3,line=-1,cex=1.15,at=c(0.2,0.5,0.85),
       c("High flux","Medium flux","low flux"))
mtext(outer=TRUE,side=3,line=1,cex=1.5,"OMEXDIAmodel")
```

`OMEXDIAparms` *OMEXDIA diagenetic model parameters*

Description

a vector with parameters of the OMEXDIA diagenetic model

Usage

```
OMEXDIAparms
```

Format

a vector with the names and values of the OMEXDIA diagenetic model parameters. See Soetaert et al., 1996 for further details

Note

To be used as input to `OMEXDIAmodel`

Author(s)

Karline Soetaert <k.soetaert@nioo.knaw.nl>

References

Soetaert K, PMJ Herman and JJ Middelburg, 1996a. A model of early diagenetic processes from the shelf to abyssal depths. Geochimica Cosmochimica Acta, 60(6):1019-1040.

See Also

OMEXDIAmodel

Examples

```
OMEXDIAparms
OMEXDIAparms["rFast"]
as.list(OMEXDIAparms)$rFast
```

---

fiadeiro *Upstream weighing coefficients for advection*

Description

Upstream weighing coefficients for advection that reduce numerical dispersion whilst conserving positivity

Usage

```
fiadeiro(v, disp, dx.int, grid=list(dx.int=dx.int))
```

Arguments

v	velocity, advection rate, [L/t], either one value or a vector of length N+1, with N the number of boxes
disp	diffusion rate, [L2/t], either one value or a vector of length N+1
dx.int	distances, [L], between centre of boxes (including two interfaces), either one value or a vector of length N+1
grid	discretisation grid, e.g. as calculated by setup.grid

Details

The Fiadeiro and Veronis (1977) weighing scheme reduces numerical diffusion, due to advection. It is based on the following rationale:

When weighing is by backward differences (weighing coefficient of the upstream box = 1) state variables remain positive, but this scheme has the highest numerical (= artificial) diffusion.

When weighing is by centered differences (weighing coefficient = 0.5), the numerical diffusion is lower, but state variables may become negative.

In the Fiadeiro and Veronis (1977) weighing scheme, the weighing coefficients of the upstream boxes depend on the magnitude of the additional true diffusion (D):
the higher the true diffusion, the closer the weighing coefficients are to centered weighing...
The weighing coefficients thus vary from 0.5 (very high true diffusion) to 1 (absence of true diffusion)

Note: if the substance concentrations decline in the direction of the axis, then centered differences will never lead to negative concentration. Thus, under these circumstances, centered differences are to be preferred over the fiadeiro scheme.

Value

the weighing coefficient(s) for the upstream compartments, either one value or a vector of length N+1

Note

A certain amount of numerical diffusion always remains.
Reducing grid size may be a more effective way of reducing numerical dispersion.

Author(s)

Karline Soetaert <k.soetaert@nioo.knaw.nl>

References

Fiadeiro Me, Veronis G, 1977
Weighted-mean schemes for finite-difference approximation to advection-diffusion equation. Tellus v 29, pp 512-522

Examples

```
##########################################################################
######   EXAMPLE : advection weighing schemes                       ######
##########################################################################

#====================#
# Model equations    #
#====================#

model <- function (time,OC,pars,decay=0.05,weight.up=1)
{
 # model describing the concentration of particulate organic C (OC)
 # that sink out of the water and decay

 dOC <- tran1D(OC,flux.up=10,disp=disp,v =sink,
                weight.up=weight.up,dx=dx)$dy - decay*OC
 return(list(dOC))
}

#====================#
# Model application  #
#====================#

dx    <- 100                  # thickness of boxes
Dist  <- seq(0,1000,by=dx)    # water depth at modeled box interface
Depth <- seq(dx/2,1000,by=dx) # water depth at centre of modeled box
N     <- length(Depth)
cc    <- -0.005

# exponentially declining sinking rate, maximal 100 m/day
sink <- 100*exp(cc*Dist)
disp <- 1000
Weights <-fiadeiro(sink,disp,dx)

# plot fiadeiro weighting coefficients as a function of sinking rate
plot(sink,Weights,type="b",main="Fiadeiro and Veronis scheme",
```

```
     xlab="sinking rate,m/day",
     ylab="upstream weighing coefficient")

# estimate the steady-state solution using a coarse grid (a0 boxes)
# and based on:
# (1) backward differences (2) centered diff, (3) fiadeiro scheme
ss   <- steady.band(y=runif(N),func=model,nspec=1)$y
ss   <- cbind(ss,steady.band(y=runif(N),func=model,
                             weight.up=0.5,nspec=1)$y)
ss   <- cbind(ss,steady.band(y=runif(N),func=model,
                             weight.up=Weights,nspec=1)$y)

#====================#
# Plotting output    #
#====================#

matplot(ss,Depth,pch=16,type="b",main="numerical diffusion",
      ylab="water depth, m",
      xlab="concentration of sinking particles",
      ylim=c(1000,0))

# now with increased resolution (1000 boxes)
dx    <- 0.1                  # thickness of boxes only 0.1 m
Dist  <- seq(0,1000,by=dx)
Depth <- seq(dx/2,1000,by=dx)
N     <- length(Depth)

# exponentially declining sinking rate, maximal 100 m/day
sink <- 100*exp(cc*Dist)

# with this resolution, the weighing coeff does not matter
st   <- steady.band(y=runif(N),fun=model,weight.up=1,nspec=1)$y
lines(st,Depth,col="darkblue",lty=1)
legend("bottomright",title="numerical diffusion",
      legend=c("upstream","centered","fiadeiro","large resolution"),
      col=c(1:3,"darkblue"),lty=c(1:3,1),pch=c(16,16,16,NA))
```

`setup.grid` *Create a numerical grid*

Description

Subdivides a certain size into grid cells, usable for transport routines. Grid cells are characterised by

thicknesses (dx), and distances from centre to centre (dx.int)

distance from upstream boundary to middle of cell (x) and from upstream boundary to upstream interface of cell (x.int)

Usage

```
setup.grid(len=NULL, N=NULL, dx.1=NULL, layer=NULL, type="ct",
dx.int.up=NULL,dx.int.down=NULL,x.0=0)
```

Arguments

len	total size to divide; not used if type = "layer"
N	number of grid cells; not used if type = "layer"; can be null if type == "ct" and dx.1 is given a value
dx.1	size of first grid cell; only used if type = "ct", "exp" or "sine"; in the latter case, it is also the size of the last grid cell. If type =="ct" and both dx.1 and N are specified, then N is re-estimated as round(len/dx.1); if N is rounded, then also dx.1 will be re-estimated
layer	3-columned matrix, specifying the number of cells (col 1), the size of the first grid cell (col 2) and the total length (col 3) in a number of discrete layers; usually the size of the first grid cell will only be specified for the first layer, it can be set NA for the other layers.
type	type of grid, one of "ct","exp","sine","layer", for constant, exponential, sine and layer type respectively
dx.int.up	the "interface" distance to the upstream boundary, i.e. the distance from the centre of 1st box to the upper interface; the default is half the size of the first box.
dx.int.down	the "interface" distance to the downstream boundary, i.e. the distance from the centre of last box to the lower interface; the default is half the size of the first box.
x.0	distance of upstream interface; defaults to 0 (start of x-axis)

Details

The interface distance dx.int.up and dx.int.down are by the default is half the size of the first/last box.
For sediments for instance, this can be set equal to the size of the diffusive boundary layer (+ 0.5*dx.1)

Value

a list containing:

x	distance from origin to centre of boxes, vector of length N
x.int	distance from origin to box upper interface, vector of length N+1
dx	thickness of boxes, vector length N
dx.int	distance over the interface, i.e. between centre of adjacent boxes, vector of length N+1
N	the number of boxes

Author(s)

Karline Soetaert <k.soetaert@nioo.knaw.nl>

Examples

```
# example
Exp    <- setup.grid(len=100,dx.1=1,N=20,type="exp")
Sine   <- setup.grid(len=100,dx.1=1,N=20,type="sine")
Ct     <- setup.grid(len=100,dx.1=1,N=20,type="ct")
```

```
# grid composed of two layers; with the following specifications:
# layer 1: 10 boxes, size of 1st box= 1, total size in layer= 40;
# layer 2: 10 boxes, size of 1st box not specified (NA), total size=60
Twolay <- setup.grid(len=100,layer=matrix(ncol=3,byrow=TRUE,
                     data=c(10,1,40,10,NA,60)),type="layer")

plot(Exp$dx,main="setup.grid",ylab="dx",xlab="box")
plot(Sine$dx,pch=16,main="setup.grid",ylab="dx",xlab="box")
plot(Ct$dx,pch=18,main="setup.grid",ylab="dx",xlab="box")
plot(Twolay$dx,pch=20,main="setup.grid",ylab="dx",xlab="box")
```

`setup.prop` *Creates a profile of properties over a grid*

Description

Estimates the values of certain properties at the centre of boxes and box interfaces, at upstream boundary and at infinite depth.
Input is either as a et of coefficient defining and exponential function or by xy-values

For instance for porous media, the function can be used to specify porosities, or bioturbation coefficients over a numerical grid.

Usage

```
setup.prop(y.0=NULL, y.INF=y.0, y.coeff=0, L=0,
  xy=NULL, type="spline", grid)
```

Arguments

`y.0`	value of property y at upstream interface (x=0)
`y.INF`	value of property y at infinite distance (x=inf)
`y.coeff`	rate of change with distance x
`L`	if specified, the thickness of the (upper) layer where the property has value y0; by default this layer is not present
`xy`	if present, a two-columned matrix, with x-y (distance-property) values, which will be inter/extrapolated to the discretisation grid overrules all other settings
`type`	only if xy is present: how the inter/extrapolation should be done, one of "spline" or "linear"
`grid`	Discretisation grid, a list containing at least elements dx and dx.int, e.g. as calculated by setup.grid, [L]

Details

If input is as data, the data should be specified in xy, a 2-columned matrix with x(distances, 1st column) and y(property values, 2nd column). These data are then inter and extrapolated to the numerical grid. To interpolate, there are two options:

"spline" gives a smooth profile, but can generate strange values - always check the result!

"linear" gives a segmened profile

If input is specified as an exponential it is estimated as:

$$y = yInf + (y0 - yInf) * exp(x * abs(ycoeff))$$

for x>L

$$y = y0$$

for x<=L

Value

a list containing:

`up`	property value at upstream boundary, one value
`INF`	property value at infinite distance, one value
`mid`	property value at middle of boxes, vector of length N
`int`	property value at upstream box interfaces, vector of length N+1

Author(s)

Karline Soetaert <k.soetaert@nioo.knaw.nl>

Examples

```
# example
Grid    <- setup.grid(len=10,N=50)
Poro    <- setup.prop(y.0=0.95,y.INF=0.8,y.coeff=1,grid=Grid)
Db      <- setup.prop(y.0=1,y.INF=0,y.coef=1,L=5,grid=Grid)
mat     <- matrix(ncol=2,data=c(1,3,5,9,0.1,0.5,0.6,0.9))
Pspline <- setup.prop(xy=mat,grid=Grid)
Plin    <- setup.prop(xy=mat,grid=Grid,type="linear")

plot(Grid$x,Poro$mid, type="l",main="setup.prop, L=0")
plot(Grid$x,Db$mid, type="l",main="setup.prop, L=5")
plot(Grid$x,Pspline$mid, type="l",main="setup.prop, spline interpolation")
points(mat,cex=1.5,pch=16)
plot(Grid$x,Plin$mid, type="l",main="setup.prop, linear interpolation")
points(mat,cex=1.5,pch=16)
```

---

tran1D *General 1-D transport*

Description

Estimates the rate of change of a substance due to diffusive and advective 1-D transport in a shape.
The shape is defined by the surface areas, which do not need to be constant.
The default considers a constant surface area equal to 1.
The axis points from upstream (box 1) to downstream (last box)

Usage

```
tran1D(y, y.up=y[1], y.down=y[length(y)],
flux.up=NA, flux.down=NA, disp, v=0, v.up=0, weight.up=1,
surf.mid=1,surf.int=surf.mid,
dx=NULL, dx.int=dx, grid=list(dx=dx,dx.int=dx.int))
```

Arguments

y	concentration (or other variable), defined in centre of boxes. A vector of length N, [Mass/Length3]
y.up	concentration at upstream boundary interface. One value, [M/L3]
y.down	concentration at downstream boundary interface. One value, [M/L3]
flux.up	flux across the upper boundary interface, positive = IN medium. One value, [M/L2/Time]
flux.down	flux across the lower boundary interface, positive= OUT of medium. One value, [M/L2/T]
disp	dispersion, diffusion coefficients, defined on box interfaces. One value or a vector of length N+1, [L2/T]
v	velocity in the axis direction, defined on box interfaces. One value or a vector of length N+1, [L/T]
v.up	velocity, against the axis direction, defined on box interfaces. One value or a vector of length N+1, [L/T]
weight.up	upstream weighing coefficient for velocity and upward velocity, defined on box interfaces; default is backward differences. One value or a vector of length N+1, [-]
surf.mid	surface area, defined in centre of boxes. One value or a vector of length N, [L2]
surf.int	surface area, defined at box interfaces. One value or a vector of length N+1, [L2]
dx	thickness of boxes. One value or a vector of length N, [L]
dx.int	distance over the interfaces, i.e. from centre to centre of boxes. One value or a vector of length N+1, [L]
grid	discretisation grid, a list containing at least elements dx (box thicknesses) and dx.int (interface distances), e.g. as calculated by setup.grid, [L] If present, overrules dx and dx.int

Details

The **boundary conditions** are either

(1) zero-gradient (default)

(2) concentration boundary

(3) flux boundary

if the flux boundary is specified, it overrules the other specifications.

Transport properties The *dispersion coefficient* (disp), *velocity* (v) and *upstream velocity*, v.up can be either one value or a vector. If they are a vector,they must be of length N+1, defined at all box interfaces, including upper and lower boundary.

The **surface area** needs to be defined:

(1) at the centre of boxes (surf.mid), either one value or a vector of length N

(2) at the box interfaces (surf.int), either one value, or a vector of length N+1

If surf.int and surf.mid are one value, they should be equal.
No attempt is done to test whether the surfaces are compatible.

The **spatial discretisation** is specified either:

by a grid, as generated by `setup.grid` or

separate values for the thicknesses: dx.int, the "dispersion distance", i.e. the distance from the centre of one box to the centre of the box below. Either one value or a vector of length N+1. Take care at boundaries!, and dx, the thickness of each box, either one value or a vector of length N.

If dx.int and dx are one value, they should be equal
No attempt is done to test whether the discretisation is compatible.

Value

a list containing:

`dy`	rate of change of y in each layer due to transport, [M/L3 liquid/T]
`flux`	Fluxes across each box interface. A vector of length N+1, [M/L2/Time]
`flux.up`	Flux across the upper boundary interface, positive = IN medium. One value, [M/L2/Time]
`flux.down`	Flux across the lower boundary interface, positive= OUT of medium. One value, [M/L2/T]

Note

The advection equation is not conservative.

This is ok if the advection is for instance a sinking or swimming velocity, then mass conservation is not an issue.

However, if v is liquid flow, and it is not constant, then the rate of change should be adapted to conserve mass. There should be inwelling if flux increases, outwelling if it decreases. This extra flux should be added to the rate of change.

(For the same reason you should be careful when specifying advection rates for bodies whose surface is not constant).

Author(s)

Karline Soetaert <k.soetaert@nioo.knaw.nl>

References

Soetaert and Herman, a guide to ecological modelling - using R as a simulation platform, 2008. Springer

Examples

```
######################################################################
###### EXAMPLE: O2 in a cylindrical and spherical organism  ######
######################################################################

#====================#
# Model equations    #
#====================#

# the numerical model - rate of change=transport-consumption
CylinderMod <- function(time,O2,pars)
      return (list(tran1D(O2,y.down=BW,disp=Da,surf.mid=A,
      surf.int=Ai,dx=dx,dx.int=dxi)$dy-Q))

SphereMod <- function(time,O2,pars)
      return (list(tran1D(O2,y.down=BW,disp=Da,surf.mid=As,
      surf.int=Asi,dx=dx,dx.int=dxi)$dy-Q))

#====================#
# Model application  #
#====================#

# parameters

BW     <- 2         # mmol/m3,  oxygen conc in surrounding water
Da     <- 0.5       # cm2/d     effective diffusion coeff in organism
R      <- 0.0025    # cm        radius of organism
Q      <- 250000    # nM/cm3/d  oxygen consumption rate/ volume / day
L      <- 0.05      # cm        length of organism (if a cylinder)

# the analytical cylindrical and spherical model
cylinder <- function(Da,Q,BW,R,r)  BW+Q/(4*Da)*(r^2-R^2)
sphere   <- function(Da,Q,BW,R,r)  BW+Q/(6*Da)*(r^2-R^2)

# the numerical model: parameters
N  <- 40                           # layers in the body
dx <- R/N                          # thickness of each layer
x  <- seq(dx/2,by=dx,length.out=N) # distance of center to mid-layer
xi <- seq(0,by=dx,length.out=N+1)  # distance to layer interface
dxi<- c(rep(dx,N),dx/2)            # dispersion distances

# Cylindrical surfaces
A   <- 2*pi*x *L              # surface at mid-layer depth
Ai  <- 2*pi*xi*L              # surface at layer interface

# Spherical surfaces
As   <- 4*pi*x^2              # surface of sphere, at each mid-layer
Asi  <- 4*pi*xi^2             # surface at layer interface

# solve the model numerically for a cylinder
CONC  <- steady.1D (runif(N),fun=CylinderMod,nspec=1,atol=1e-10)
O2    <- CONC$y

# solve the model numerically for a sphere
CONC2 <- steady.1D (runif(N),fun=SphereMod,nspec=1,atol=1e-10)
O2b   <- CONC2$y
```

```
#====================#
# Plotting output    #
#====================#

plot(x,O2,xlab="distance from centre, cm",ylab="mmol/m3",
main="tran1D",sub="diffusion-reaction in a cylinder and sphere")

lines(x, cylinder(Da,Q,BW,R,x))

points(x, O2b, pch=18,col="red" )
lines(x, sphere(Da,Q,BW,R,x),col="red")

legend ("topleft",lty=c(1,NA),pch=c(NA,1),
        c("analytical solution","numerical approximation"))
legend ("bottomright",pch=c(1,18),lty=1,col=c("black","red"),
        c("cylinder","sphere"))
```

`tran1D.solid` *Transport of solid substances in a porous medium*

Description

Estimates the rate of change of solid (particulate) substances due to 1-D diffusive and advective transport in porous media

Usage

```
tran1D.solid(y, y.up=y[1],y.down=y[length(y)],
flux.up=NA, flux.down=NA, disp, v=0, v.up=0, weight.up=1,
por.INF, por.int=por.INF, por.mid=por.INF,
porgrid = list(INF=por.INF,int=por.int,mid=por.mid),
dx=NULL, dx.int=dx, grid=NULL)
```

Arguments

`y`	concentration per unit of solid in the porous medium, defined in centre of boxes. A vector of length N, [Mass/Length3], e.g. [nmol/cm3 solid]
`y.up`	concentration at upstream boundary interface. One value, [M/L3 solid]
`y.down`	concentration at downstream boundary interface. One value, [M/L3 solid]
`flux.up`	flux across the upper boundary interface, positive = IN medium. One value, [M/L2/Time]
`flux.down`	flux across the lower boundary interface, positive = OUT of medium. One value, [M/L2/T]
`disp`	dispersion coefficients (e.g. sediment turbation rate), defined on box interfaces. One value or a vector of length N+1, [L2/T]
`v`	velocity in the axis direction (e.g. sedimentation rate), defined on box interfaces. One value or a vector of length N+1, [L/T]
`v.up`	upward velocity, against the axis direction, defined on box interfaces. One value or a vector of length N+1, [L/T]

weight.up	upstream weighing coefficient for advection and upward flow, defined on box interfaces; default is backward differences. One value or a vector of length N+1, [-]
por.INF	(volumetric) porosity at infinite depth. One value, [-]
por.int	(volumetric) porosity at box interfaces. One value or a vector of length N+1, [-]
por.mid	(volumetric) porosity in middle (centre) of boxes. One value or a vector of lenght N, [-]
porgrid	porosity grid, a list containing the elements INF, int and mid, e.g. as calculated by setup.prop, [-]
dx	thickness of boxes. One value or a vector of length N, [L]
dx.int	distance over the interfaces, i.e. from centre to centre of boxes. One value or a vector of length N+1, [L]
grid	discretisation grid, a list containing at least elements dx and dx.int, e.g. as calculated by setup.grid, [L]

Details

For a specification of the *boundary conditions* and *spatial discretisation* see details of function tran1D

The *bioturbation coefficient* (disp), and *sedimentation rate* (v) can be either one value or a vector. If they are a vector, they must be of length N+1, defined at all box interfaces, including upper and lower boundary.

Porosity needs to be defined:

1. at the box interfaces (por.int), either one value, or a vector of length N+1

2. at the centre of boxes (por.mid), either one value or a vector of length N

3. and at infinite depth (por.INF).

Alternatively, porosity can be inputted as a list, e.g. created by `setup.prop`

If por.int and por.mid are one value, they should be equal to por.INF .
No attempt is done to test whether the inputted porosities are compatible.

Value

a list containing:

dy	Rate of change of y in each layer due to transport, [M/L3 solid/T]
flux	Fluxes across each box interface. A vector of length N+1, [M/L2 BULK/Time]
flux.up	Flux across the upper boundary interface, positive = IN medium. One value, [M/L2 BULK/Time]
flux.down	Flux across the lower boundary interface, positive= OUT of medium. One value, [M/L2 BULK/T]

Note

This is just a particular application of the general transport routine tran1D, where *1-porosity* provides the *surface area* and where the advection rate is corrected for the porosity gradient (to account for steady-state compaction).

Author(s)

Karline Soetaert <k.soetaert@nioo.knaw.nl>

References

Berner R.A., 1980. Early Diagenesis- A Theoretical Approach. Princeton Univ. Press
Boudreau, B.P., 1997. Diagenetic Models and their Implementation. Modelling transport and Reactions in Aquatic Sediments. Springer, Berlin, 414p.
Soetaert and Herman, a guide to ecological modelling - using R as a simulation platform, 2008. Springer.

Reference for the OMEXDIA model (example 2):
Soetaert K, PMJ Herman and JJ Middelburg, 1996a. A model of early diagenetic processes from the shelf to abyssal depths. Geochimica Cosmochimica Acta, 60(6):1019-1040.
Soetaert K, PMJ Herman and JJ Middelburg, 1996b. Dynamic response of deep-sea sediments to seasonal variation: a model. Limnol. Oceanogr. 41(8): 1651-1668.

See Also

tran1D.solute, tran1D.volume, tran1D

Examples

```
##########################################################################
######  EXAMPLE 1: Boundary conditions                              ######
##########################################################################

#====================#
# Model equations    #
#====================#
# model of 1-D transport and 1-st order decay
# upper boundary conc imposed, zero-gradient lower boundary

model <- function (t,Conc,pars,w)
   return (list(tran1D.solid(Conc,flux.up=10,disp=disp,v=w,
                         por.INF =poro ,dx=dx)$dy-Conc*rate))

#====================#
# Model application  #
#====================#

y.down <- y.up <- 100       # mmol/m3
poro     <- 0.9             # constant porosity
disp     <- 0.01            # cm2/day
w        <- 0.01            # cm/day
dx       <- 0.01            # 0.01 cm thick slices, constant thickness
N        <- 1000            # 1000 layers
rate     <- 0.1             #/day 1st-order consumption rate
Depth    <- seq(from=dx/2,by=dx,len=N)

# Steady-state solution, different values of advection rate
Conc1    <- steady.band(y=runif(N),func=model,nspec=1,w=0.01)
Conc2    <- steady.band(y=runif(N),func=model,nspec=1,w=0.001)
Conc3    <- steady.band(y=runif(N),func=model,nspec=1,w=0.1)

#====================#
```

```
# Plotting output    #
#====================#

matplot(cbind(Conc1$y,Conc2$y,Conc3$y),Depth,ylim=rev(range(Depth)),
        ylab="depth, cm",xlab="Concentration",type="l",lwd=2,
        main="tran1D.solid")
legend ("right",c("w=0.01","w=0.001","w=0.1"),
 title="sedimentation rate",lty=1:3,col=1:3,lwd=2)

########################################################################
######  EXAMPLE 2: OMEXDIA: C, N, O2 diagenesis              ######
########################################################################

#====================#
# Model equations    #
#====================#

OMEXDIAmod <-function(time=0,     # time, not used here
                      state,      # FDET, SDET, O2, NO3, NH3, ODU
                      parms,      # parameter values
                      Full=FALSE) # if true: returns output vars
{
with (as.list(pars),{

Flux  <- MeanFlux * (1+sin(2*pi*time/365))

FDET  <- state[1:N]
SDET  <- state[(N+1)  :(2*N)]
O2    <- state[(2*N+1):(3*N)]
NO3   <- state[(3*N+1):(4*N)]
NH3   <- state[(4*N+1):(5*N)]
ODU   <- state[(5*N+1):(6*N)]

# Rate of change due to transport
# Solid substances
FDETtran<- tran1D.solid(FDET,flux.up=Flux*pFast   ,disp=Db,v=w,
                     weight.up=0.5,porgrid=porGrid, grid=Grid)

SDETtran<- tran1D.solid(SDET,flux.up=Flux*(1-pFast),disp=Db,v=w,
                     weight.up=0.5,porgrid=porGrid, grid=Grid)

# Solute substances
O2tran  <- tran1D.solute(y=O2, y.up=bwO2, disp=DispO2, v=w,
                     porgrid=porGrid, grid=Grid)

NO3tran <- tran1D.solute(y=NO3, y.up=bwNO3,disp=DispNO3,v=w,
                     porgrid=porGrid, grid=Grid)

NH3tran <- tran1D.solute(y=NH3,y.up=bwNH3,disp=DispNH3/(1+NH3Ads),
                     v=w,porgrid=porGrid, grid=Grid)

ODUtran <- tran1D.solute(y=ODU,y.up=bwODU,disp=DispODU,v=w,
                     porgrid=porGrid, grid=Grid)

# production of DIC and DIN, expressed per cm3 LIQUID/day
DICprod_Min <- (rFast*FDET         +rSlow*SDET       )*
               (1.-porGrid$mid)/porGrid$mid
```

```
DINprod_Min <- (rFast*FDET*NCrFdet+rSlow*SDET*NCrSdet)*
               (1.-porGrid$mid)/porGrid$mid

# oxic mineralisation, denitrification, anoxic mineralisation
Oxicminlim <- O2/(O2+ksO2oxic)               # limitation terms
Denitrilim <- (1-O2/(O2+kinO2denit))*NO3/(NO3+ksNO3denit)
Anoxiclim  <- (1-O2/(O2+kinO2anox))*(1-NO3/(NO3+kinNO3anox))
Rescale    <- 1/(Oxicminlim+Denitrilim+Anoxiclim)

OxicMin    <- DICprod_Min*Oxicminlim*Rescale  # oxic mineralisation
Denitrific <- DICprod_Min*Denitrilim*Rescale  # Denitrification
AnoxicMin  <- DICprod_Min*Anoxiclim *Rescale  # an0xic mineralisation

# reoxicidation and ODU deposition
Nitri      <- rnit  *NH3*O2/(O2+ksO2nitri)
OduOx      <- rODUox*ODU*O2/(O2+ksO2oduox)

pDepo      <- min(1,0.233*(w*365)^0.336 )
OduDepo    <- AnoxicMin*pDepo

# Update the rate of change
dFDET <- FDETtran$dy - rFast*FDET
dSDET <- SDETtran$dy - rSlow*SDET
dO2   <- O2tran$dy  -  OxicMin      -2* Nitri -      OduOx
dNH3  <- NH3tran$dy + (DINprod_Min  - Nitri) / (1.+NH3Ads)
dNO3  <- NO3tran$dy - 0.8*Denitrific + Nitri
dODU  <- ODUtran$dy + AnoxicMin  - OduOx - OduDepo

#
if (!Full)return(list(c(dFDET,dSDET,dO2,dNO3,dNH3,dODU))) else

return(list(c(dFDET,dSDET,dO2,dNO3,dNH3,dODU),
            c(O2flux=O2tran$flux.up,        #O2 sediment-water flux
              O2deepflux=O2tran$flux.down,  #O2 deep(burial) flux
              NO3flux=NO3tran$flux.up,      #NO3 sediment-water flux
              NO3deepflux=NO3tran$flux.down,#NO3 deep(burial) flux
              NH3flux=NH3tran$flux.up,      #NH3 sediment-water flux
              NH3deepflux=NH3tran$flux.down,#NH3 deep (burial) flux
              ODUflux=ODUtran$flux.up,      #ODU sediment-water flux
              ODUdeepflux=ODUtran$flux.down,#ODU deep(burial) flux
              OxicMin=OxicMin,              #oxic mineralisation
              Denitrific=Denitrific,        #denitrification rates
              AnoxicMin=AnoxicMin,          #anoxic mineralisation
              Nitri=Nitri,                  #nitrification rates
              OduOx=OduOx)))                #ODU oxidation rates

})

}


#====================#
# Model applications #
#====================#

# 50 layers; total length=15 cm, first box=0.1 cm
Grid  <- setup.grid(N=50,dx.1=0.1,len=15,type="exp")
```

```
Depth <- Grid$x                  # depth of each box
N     <- Grid$N

# porosity gradients
porGrid <- setup.prop(y.0=0.9,y.INF=0.7,y.coeff=2,grid=Grid)

# Bioturbation profile (at box interfaces)
biot <- 1/365          # cm2/d      - bioturbation coefficient
mixL <- 5              # cm         - depth of mixed layer
Db   <- setup.prop(y.0=biot,y.INF=0,y.coeff=1,L=mixL,grid=Grid)$int

# organic matter dynamics  #

MeanFlux <- 20000/12*100/365 # nmol/cm2/d - C deposition: 20gC/m2/yr
rFast    <- 0.01             #/day        - decay rate fast decay det.
rSlow    <- 0.00001          #/day        - decay rate slow decay det.
pFast    <- 0.9              #-           - fraction fast det. in flux
w        <- 0.1/1000/365     # cm/d       - advection rate
NCrFdet  <- 0.16             # molN/molC  - NC ratio fast decay det.
NCrSdet  <- 0.13             # molN/molC  - NC ratio slow decay det.

# oxygen and DIN dynamics  #

# Nutrient bottom water conditions
bwO2            <- 300       #mmol/m3     Oxygen conc in bottom water
bwNO3           <- 10        #mmol/m3
bwNH3           <- 1         #mmol/m3
bwODU           <- 0         #mmol/m3

# Nutrient parameters
NH3Ads          <- 1.3    #-           Adsorption coeff ammonium
rnit            <- 20.    #/d          Max nitrification rate
ksO2nitri       <- 1.     #umolO2/m3   half-sat O2 in nitrification
rODUox          <- 20.    #/d          Max rate oxidation of ODU
ksO2oduox       <- 1.     #mmolO2/m3   half-sat O2 in oxidation of ODU
ksO2oxic        <- 3.     #mmolO2/m3   half-sat O2 in oxic mineralis
ksNO3denit      <- 30.    #mmolNO3/m3  half-sat NO3 in denitrif
kinO2denit      <- 1.     #mmolO2/m3   half-sat O2 inhib denitrif
kinNO3anox      <- 1.     #mmolNO3/m3  half-sat NO3 inhib anoxic min
kinO2anox       <- 1.     #mmolO2/m3   half-sat O2 inhib anoxic min

# Diffusion coefficients, temp = 10dgC
Temp            <- 10                      # temperature
DispO2          <- 0.955    +Temp*0.0386
DispNO3         <- 0.844992 +Temp*0.0336
DispNH3         <- 0.84672  +Temp*0.0336
DispODU         <- 0.8424   +Temp*0.0242

# parameter vector
pars<- c(
MeanFlux  =  MeanFlux   ,rFast     =  rFast      ,
rSlow     =  rSlow      ,pFast     =  pFast      ,
w         =  w          ,NCrFdet   =  NCrFdet    ,
NCrSdet   =  NCrSdet    ,bwO2      =  bwO2       ,
bwNO3     =  bwNO3      ,bwNH3     =  bwNH3      ,
bwODU     =  bwODU      ,NH3Ads    =  NH3Ads     ,
rnit      =  rnit       ,ksO2nitri =  ksO2nitri  ,
```

```
rODUox    =  rODUox     ,ksO2oduox =  ksO2oduox  ,
ksO2oxic  =  ksO2oxic   ,ksNO3denit=  ksNO3denit ,
kinO2denit=  kinO2denit ,kinNO3anox=  kinNO3anox ,
kinO2anox =  kinO2anox  ,DispO2    =  DispO2     ,
DispNO3   =  DispNO3    ,DispNH3   =  DispNH3    ,
DispODU   =  DispODU  )

# STEADY-STATE CALCULATIONS USING R-function
#-----------------------------------------------
# Three runs, with different flux
pars["MeanFlux"] <- 15000/12*100/365  # nmol/cm2/day - C deposition: 15gC/m2/yr
OC   <- rep(10,6*N)
DIA  <- steady.1D (y=OC, func=OMEXDIAmod,Full=FALSE,
                   parms=pars, maxiter=100,
                   atol=1e-8,nspec=6,positive=TRUE)
CONC <- DIA$y

pars["MeanFlux"] <- 50000/12*100/365  # nmol/cm2/day - C deposition: 50gC/m2/yr
OC   <- rep(10,6*N)
DIA  <- steady.1D (y=OC, func=OMEXDIAmod,Full=FALSE,
                   parms=pars, maxiter=100,
                   atol=1e-8,nspec=6,positive=TRUE)
CONC <- cbind(CONC,DIA$y)

pars["MeanFlux"] <- 2000/12*100/365  # nmol/cm2/day - C deposition: 2gC/m2/yr
OC   <- rep(10,6*N)
DIA  <- steady.1D (y=OC, func=OMEXDIAmod,Full=FALSE,
                   parms=pars, maxiter=100,
                   atol=1e-8,nspec=6,positive=TRUE)
CONC <- cbind(CONC,DIA$y)


# Concentrations in sediment
FDET  <- CONC[1:N,]
SDET  <- CONC[(N+1)  :(2*N),]
O2    <- CONC[(2*N+1):(3*N),]
NO3   <- CONC[(3*N+1):(4*N),]
NH3   <- CONC[(4*N+1):(5*N),]
ODU   <- CONC[(5*N+1):(6*N),]

TOC  <- (FDET+SDET)*1200/10^9/2.5     #

par(mfrow=c(2,2))
matplot(TOC,Depth,ylim=c(15,0),xlab="procent" ,main="TOC",
        type="l",lwd=2)
matplot(O2,Depth,ylim=c(15,0),xlab="mmol/m3" ,main="O2",
        type="l",lwd=2)
matplot(NO3,Depth,ylim=c(15,0),xlab="mmol/m3" ,main="NO3",
        type="l",lwd=2)
matplot(NH3,Depth,ylim=c(15,0),xlab="mmol/m3" ,main="NH3",
        type="l",lwd=2)

legend ("bottom",col=1:3,lty=1:3,lwd=2,
legend=c("15gC/m2/yr","50gC/m2/yr","2gC/m2/yr"),title="flux")

mtext(outer=TRUE,side=3,line=-2,cex=1.5,"OMEXDIAmodel")
```

```
#---------------------------------------
# FOR an application of a DYNAMIC RUN
# SEE the example of OMEXDIAmodel
# (a Fortran implementation of OMEXDIA
#---------------------------------------
```

`tran1D.solute` *Transport of dissolved substances in a porous medium*

Description

Estimates the rate of change of dissolved substances due to 1-D diffusive and advective transport in porous media

Usage

```
tran1D.solute(y, y.up, y.down=y[length(y)], flux.up=NA,
flux.down=NA, disp, v=0, v.up=0, weight.up=1,
por.INF , por.int=por.INF , por.mid=por.INF ,
porgrid = list(INF=por.INF,int=por.int,mid=por.mid),
tortuosity=1,dx=NULL, dx.int=dx, grid=NULL)
```

Arguments

`y`	concentration per unit of solute in the porous medium, defined in centre of boxes. A vector of length N, [Mass/Length3], e.g. [nmol/cm3 liquid]
`y.up`	concentration at upstream boundary interface. One value, [M/L3 liquid]
`y.down`	concentration at downstream boundary interface. One value, [M/L3 liquid]
`flux.up`	flux across the upper boundary interface, positive = IN medium. One value, [M/L2/Time]
`flux.down`	flux across the lower boundary interface, positive= OUT of medium. One value, [M/L2/T]
`disp`	molecular diffusion coefficients, defined on box interfaces. One value or a vector of length N+1, [L2/T]
`v`	velocity in the axis direction (e.g. sedimentation rate), defined on box interfaces. One value or a vector of length N+1, [L/T]
`v.up`	upwelling rate, against the axis direction, defined on box interfaces. One value or a vector of length N+1, [L/T]
`weight.up`	upstream weighing coefficient for advection and upward flow, defined on box interfaces; default is backward differences. One value or a vector of length N+1, [-]
`por.INF`	(volumetric) porosity at Infinite depth. One value, [-]
`por.int`	(volumetric) porosity at box interfaces. One value or a vector of length N+1, [-]
`por.mid`	(volumetric) porosity in centre of boxes. One value or a vector of lenght N, [-]
`porgrid`	porosity grid, a list containing the elements INF, int and mid, e.g. as calculated by setup.prop, [-]
`tortuosity`	sediment tortuosity coefficient

dx	thickness of boxes. One value or a vector of length N, [L]
dx.int	distance over the interfaces, i.e. from centre to centre of boxes. One value or a vector of length N+1, [L]
grid	discretisation grid, a list containing at least elements dx and dx.int, e.g. as calculated by setup.grid, [L]

Details

For a specification of the *boundary conditions* and *spatial discretisation* see function tran1D

See tran1D.solid for details about *transport coefficients* and *porosity*.

Value

a list containing:

dy	rate of change of y in each layer due to transport, [M/L3 liquid/T]
flux	Fluxes across each box interface. A vector of length N+1, [M/L2/Time]
flux.up	Flux across the upper boundary interface, positive = IN medium. One value, [M/L2 BULK/Time]
flux.down	Flux across the lower boundary interface, positive= OUT of medium. One value, [M/L2 BULK/T]

Note

This is just a particular application of the general transport routine tran1D, where *porosity* provides the *surface area* and where the advection rate is corrected for porosity (to account for steady-state compaction).

Author(s)

Karline Soetaert <k.soetaert@nioo.knaw.nl>

References

Berner R.A., 1980. Early Diagenesis- A Theoretical Approach. Princeton Univ. Press
Boudreau, B.P., 1997. Diagenetic Models and their Implementation. Modelling transport and Reactions in Aquatic Sediments. Springer, Berlin, 414p.
Soetaert and Herman, a guide to ecological modelling - using R as a simulation platform, 2008. Springer.

Reference for the SEEPDIA model (example 3):
van Oevelen, 2007. HERMES report Deliverable 47

See Also

tran1D.solid, tran1D.volume, tran1D

Examples

```
##################################################################
######  EXAMPLE 1: Different boundary conditions            ######
##################################################################

#====================#
# Model equations    #
#====================#
# model of transport and first-order decay
# MODEL 1: two imposed boundary concentrations
model1 <- function (t,Conc,pars)
   return (list(tran1D.solute(Conc,y.up,y.down,disp=disp,por.INF =poro ,
                         dx=dx)$dy-Conc*rate))

# MODEL 2: upper boundary conc imposed, zero-gradient lower boundary
model2 <- function (t,Conc,pars)
   return (list(tran1D.solute(Conc,y.up,disp=disp,por.INF =poro ,
                         dx=dx)$dy-Conc*rate))
#====================#
# Model application  #
#====================#

y.down <- y.up <- 100       # mmol/m3
poro     <- 0.9             # constant porosity
disp     <- 1               # cm2/day
dx       <- 0.01            # 0.01 cm thick slices, constant thickness
N        <- 1000            # 1000 layers
rate     <- 0.1             #/day 1st-order consumption rate
Depth    <- seq(from=dx/2,by=dx,len=N)

# upper boundary conc imposed, zero-gradient lower boundary
model3 <- function (t,Conc,pars)
   return (list(tran1D.solute(Conc,flux.up=10,disp=disp,
                     por.INF =poro,dx=dx)$dy-Conc*rate))

# steady-state solution of the different models
Conc1     <- steady.band(runif(N),func=model1,nspec=1)
Conc2     <- steady.band(runif(N),func=model2,nspec=1)
Conc3     <- steady.band(runif(N),func=model3,nspec=1)

#====================#
# model output       #
#====================#

matplot(cbind(Conc1$y,Conc2$y,Conc3$y),Depth,ylab="depth, cm",
        ylim=rev(range(Depth)),xlab="Concentration",type="l",
        lwd=2,main="tran1D.solute")
legend ("right", title="Boundaries",lty=1:3,col=1:3,lwd=2,
        c("upper and lower: conc","upper: conc, lower: 0-gradient",
          "upper:flux, lower: 0-gradient"))

##################################################################
######  EXAMPLE 2: OXYGEN diagenesis                        ######
##################################################################

#====================#
```

```
# Model equations    #
#====================#

O2DIA <- function (time=0,      # time, not used here
                   O2,          # oxygen concentration
                   parms=NULL,  # parameter values; not used
                   k=0,         # 1st-order O2 consumption rate
                   Q=0)         # 0-order consumption rate

#================================================
# Oxygen is transported and consumed with
# 1-st order rate k and 0-th order rate Q
#================================================

{
# Rate of change due to transport
O2tran  <- tran1D.solute(O2,y.up=bwO2,disp=dispO2,
                         porgrid=porGrid,grid=Grid)
return (list(O2tran$dy - k*O2 - Q)           )
}

#====================#
# Model application  #
#====================#
# parameters and grid
# model grid includes diffusive boundary layer
DBL      <- 0.2
Grid     <- setup.grid(dx.1=0.05,len=10,
                       dx.int.up = DBL+0.05/2)

# sediment parameters
N        <- Grid$N
Depth    <- Grid$x  # depth at middle of each layer

# porosity gradient
porGrid  <- setup.prop(y.0=0.8,y.INF=0.6,y.coeff=2,grid=Grid)

# porosity in diffusive boundary layer = 1
porGrid$int[1] <- 1

# biogeochemical parameters
dispO2   <- 0.4   # diffusion coefficient, cm2/d
bwO2     <- 250   # bottom water oxygen concentration, mmol/m3

# 4 model applications: different consumption rate
# steady-state solution
# constant 1-st order decay
k    <- rep(0.3,N)

# initial guess: N random numbers between 0,1
sol  <- steady.band (runif(N), fun=O2DIA, pos=TRUE, k=k,nspec=1)
O2   <- c(bwO2,sol$y)    # add bwO2 concentration to result

# hyperbolic declining 1-st order decay
k2   <- 5*(1-Depth/(Depth+0.1))
sol  <- steady.band (runif(N), fun=O2DIA, pos=TRUE, k=k2,nspec=1)
O2   <- cbind(O2,c(bwO2,sol$y))
```

```
# hyperbolic increasing 1-st order decay
k3   <- 1*(Depth/(Depth+2.5))   # hyperbolic increase
sol  <- steady.band (runif(N), fun=O2DIA, pos=TRUE, k=k3,nspec=1)
O2   <- cbind(O2,c(bwO2,sol$y))

O2cons <- O2[-1,] * cbind(k,k2,k3)  # true o2-consumption rate

# 0-order decay
Q     <- 10
sol   <- steady.band (runif(N), fun=O2DIA, k=0,Q=Q,nspec=1)
O2    <- cbind(O2,c(bwO2,pmax(0,sol$y)))  # negative O2 removed
O2cons<- cbind(O2cons,rep(Q,N))

#====================#
# model output       #
#====================#

pm   <- par(mfrow = c(1,2))
matplot(O2,c(-DBL,Depth),ylim=c(10,-DBL),main="O2",type="l",
xlab="mmolO2/m3 Liquid",ylab="depth, cm",lwd=c(1,2,1,1),col=1:4)
abline(h=0)

matplot(O2cons,Depth,ylim=c(10,-DBL),main="O2 consumption rate",
        type="l",log="x",xlab="mmolO2/m3/d",
        ylab="depth, cm",lwd=c(1,2,1,1),col=1:4)
abline(h=0)

mtext(outer=TRUE,side=3,"O2 diagenesis")
par(mfrow=pm$mfrow)

######################################################################
######  EXAMPLE 3: SEEP diagenesis                              ######
######################################################################

#====================#
# Model equations    #
#====================#

SEEPDIAmodel <- function (time=0,      # time, not used here
                          state,       # concentrations: O2,HS,SO4,CH4
                          parms=NULL)  # parameter values; not used

#=======================================================================
# Estimates the rate of change of Oxygen, hydrogen sulphide,
# sulphate, and methane under seepage conditions.
# Reactions are anaerobic oxidation of methane (AOM) and
# the reoxidation of sulphide (HSox)
#=======================================================================

{

 O2  <- state[1:N]
 HS  <- state[(N+1)  :(2*N)]
 SO4 <- state[(2*N+1):(3*N)]
 CH4 <- state[(3*N+1):(4*N)]
```

```
# Rate of change due to transport
 O2tran  <- tran1D.solute(O2,y.up=bwO2,y.down=deepO2,disp=dispO2,
                          v=w,v.up=upwel,porgrid=porGrid,grid=Grid)

 HStran <- tran1D.solute(HS,y.up=bwHS,y.down=deepHS,disp=dispHS,
                          v=w,v.up=upwel,porgrid=porGrid,grid=Grid)

 SO4tran <- tran1D.solute(SO4,y.up=bwSO4,y.down=deepSO4,disp=dispSO4,
                          v=w,v.up=upwel,porgrid=porGrid,grid=Grid)

 CH4tran <- tran1D.solute(CH4,y.up=bwCH4,y.down=deepCH4,disp=dispCH4,
                          v=w,v.up=upwel,porgrid=porGrid,grid=Grid)

# Anaerobic oxidation of methane
 AOM  <- rAOM * SO4 * CH4

# reoxidation of sulphide
 HSox <- rHSox * HS * O2

# Update the rate of change
#dSV  =  transport     + reaction
 dO2  <- O2tran$dy  -2*HSox
 dHS  <- HStran$dy  -  HSox + AOM
 dSO4 <- SO4tran$dy +  HSox - AOM
 dCH4 <- CH4tran$dy         - AOM

# returning values

return(list(c(dO2=dO2,dHS=dHS,dSO4=dSO4,dCH4=dCH4), # rates of change
          O2flux =O2tran$flux.up ,       #O2 sediment-water flux
          O2deepflux =O2tran$flux.down,  #O2 deep (burial) flux
          HSflux =HStran$flux.up ,       #sulphide sediment-water flux
          HSdeepflux =HStran$flux.down,  #sulphide deep(burial) flux
          SO4flux=SO4tran$flux.up,       #Sulphate sediment-water flux
          SO4deepflux=SO4tran$flux.down, #Sulphate deep (burial) flux
          CH4flux=CH4tran$flux.up,       #Methane sediment-water flux
          CH4deepflux=CH4tran$flux.down, #Methane deep(burial) flux
          AOM=AOM,          # profile of anoxic oxidation of methane
          HSox=HSox))       # profile of sulphide reoxidation rates

}

#====================#
# Model run          #
#====================#
# sediment parameters
Grid  <- setup.grid(dx.1=0.2,len=100)
Depth <- Grid$x   # depth of each box
N     <- Grid$N

# porosity gradient: surf por=0.9,deep por=0.7)
porGrid <- setup.prop(y.0=0.9,y.INF=0.7,y.coeff=2,
                   grid=Grid)

w       <- 0           # sediment advection rate, cm/hr
upwel   <- 10/365/24   # upwelling rate, cm/hr
```

```
# deep concentrations
deepO2  <-         0   # mmol/m3
deepHS  <-         0   # mmol/m3
deepSO4 <-         0   # mmol/m3
deepCH4 <-     68000   # mmol/m3

# bottom water concentrations
bwO2    <-      180    # mmol/m3
bwHS    <-         0   # mmol/m3
bwSO4   <-     28900   # mmol/m3
bwCH4   <-         0   # mmol/m3

# diffusion coefficients
dispCH4  <- 314/365/24 # cm2/hr
dispSO4  <- 169/365/24 # cm2/hr
dispO2   <- 265/365/24 # cm2/hr
dispHS   <- 346/365/24 # cm2/hr

# process rates
rAOM    <-  80e-6/365/24  #/(mmol/m3)/hr  rate anaerobic oxid methane
rHSox   <-  176e-6/365/24 #/(mmol/m3)/hr  rate reoxid hydrogen sulphate

# Solve steady-state; initialise with random numbers between 0,1
# Three applications, upwelling rate varying from 10 -> 1000 cm/year

# require(inverse.R)
# scenario 1
upwel   <- 10/365/24    # cm/hr
CONC10  <- steady.1D (runif(4*N),fun=SEEPDIAmodel,atol=1e-8,nspec=4,
                      positive=TRUE)$y
AOM10   <- SEEPDIAmodel(state=CONC10)$AOM

# scenario 2
upwel   <- 100/365/24   # cm/hr
CONC100 <- steady.1D (CONC10,fun=SEEPDIAmodel,atol=1e-8,nspec=4,
                      positive=TRUE)$y
AOM100  <- SEEPDIAmodel(state=CONC100)$AOM

# scenario 3
upwel   <- 1000/365/24  # cm/hr
CONC1000<- steady.1D (CONC100,fun=SEEPDIAmodel,atol=1e-8,nspec=4,
                      positive=TRUE)$y
AOM1000 <- SEEPDIAmodel(state=CONC1000)$AOM

# all scenarios combined in one matrix
CONC <- cbind(CONC10,CONC100,CONC1000)
AOM  <- cbind(AOM10,AOM100,AOM1000)

# 3 columns for each substance
O2  <- CONC[1:N,]
HS  <- CONC[(N+1)  :(2*N),]
SO4 <- CONC[(2*N+1):(3*N),]
CH4 <- CONC[(3*N+1):(4*N),]

#====================#
# plotting output    #
#====================#
```

```
# layout: 1st row: 3 columns merged and small (0.2);
#         2nd row: higher (0.8) with 3 columns, equally large (2)
nf <- layout(matrix(c(1,1,1,2,3,4), ncol=3, nrow=2, byrow = TRUE),
             width=c(2,2,2),height=c(.25,.75))

cextxt <- 1.2
cextxt2 <- 1.3

## first the triangle plot

plot(0,xlim=c(0,1),ylim=c(0,1),type="n",axes=FALSE,xlab="",ylab="")
polygon(x=c(0,1,1),y=c(0.5,1,0),col="darkblue")
text(1,0.5,"fluid flow",col="white",cex=1.8,adj=1)

mtext(side=1,outer=FALSE,line=3, at=0.3,"methane",cex=cextxt,
      col="darkblue")
mtext(side=1,outer=FALSE,line=5, at=0.3,"sulphate",cex=cextxt,
      col="darkgreen")
mtext(side=1,outer=FALSE,line=4, at=0.5,"mmol /m3",cex=cextxt)

upwel<-c("10 cm /yr","100 cm /yr","1000 cm /yr")

## then the three sediment depth profiles
for (i in 1:3)
{
plot (AOM[,i],Depth,ylim=c(100,0),col="red",type="l",lwd=3,xlab="",
          ylab="Sedimentdepth, cm")
legend("bottom",upwel[i])
par(new=TRUE)
plot(CH4[,i],Depth,ylim=c(100,0),xlim=c(0,70000),
          col="darkblue",type="l",lwd=3,axes=FALSE,xlab="",ylab="")
axis(side=3)
lines(SO4[,i],Depth,col="darkgreen",lwd=3)
}

mtext(side=1,outer=TRUE,line=-2, "mmol/m3/hr",cex=cextxt)
mtext(side=1,at=0.3,outer=TRUE,line=-2, "AOM",cex=cextxt,
      col="red",adj=0)
mtext(outer=TRUE,side=3,line=-2,cex=1.5,"SEEPDIAmodel")
```

`tran1D.volume` *1-D advective-diffusive volumetric transport*

Description

Estimates the rate of change of substances due to dispersive and advective transport in 1-D water bodies whose volume is not necessarily constant, e.g. estuaries

Usage

```
tran1D.volume(y, y.up=y[1], y.down=y[length(y)],
input.up=NA, input.down=NA, flow, flow.up=0,
weight.up=1, bulkdisp, volume)
```

Arguments

y	concentration (or other variable), defined in centre of boxes. A vector of length N, [Mass/Length3]
y.up	concentration (or other variable), at upstream boundary interface. One value, [M/L3]
y.down	concentration (or other variable), at downstream boundary interface. One value, [M/L3]
input.up	total input across the upper boundary interface, positive = IN system. One value, [M/Time]
input.down	total input across the lower boundary interface, positive= IN of system. One value, [M/T]
flow	flow rate, defined on box interfaces. One value or a vector of length N+1, [L3/T]
flow.up	upward flow rate, against the axis, defined on box interfaces. One value or a vector of length N+1, [L3/T]
weight.up	upstream weighing coefficient for flow and upward flow, defined on box interfaces; default is backward differences. One value or a vector of length N+1, [-]
bulkdisp	BULK dispersion coefficient (=disp*A.int/dx.int), defined on box interfaces. One value or a vector of length N+1, [L3/T]
volume	box volume. One value or a vector of length N, [L3]

Details

The **boundary conditions** are either of type

1. zero-gradient (default)

2. concentration boundary

3. flux boundary

If the flux boundary is specified, it overrules the other specification

Often both boundaries will consist of a concentration boundary

transport properties The *bulk dispersion coefficient* (disp), and *flow rate*, flow can be either one value or a vector.

If they are a vector, they must be of length N+1, defined at all box interfaces, including upstream and downstream boundary.

The spatial discretisation is given by:

volume, the volume of each box; either one number or one value for each box.

Value

a list containing:

dy	rate of change of y in each layer due to transport, [M/L3/T]
input	Total inputs across each box interface. A vector of length N+1, [M/Time]
input.up	Total input across the upper boundary interface, positive = IN system. One value, [M/Time]
input.down	Total input across the lower boundary interface, positive= IN system. One value, [M/T]

Author(s)

Karline Soetaert <k.soetaert@nioo.knaw.nl>

References

Soetaert and Herman, a guide to ecological modelling - using R as a simulation platform, 2008. Springer.

See Also

tran1D.solute, tran1D.solid, tran1D

Examples

```
######################################################################
######   EXAMPLE : Boundary conditions                          ######
######################################################################

#====================#
# Model equations    #
#====================#

# two imposed boundary concentrations and rate
model1 <- function (t,Conc,pars=NULL,y.up,y.down,rate)
   return (list(tran1D.volume(Conc,y.up=y.up,y.down=y.down,flow=Flow,
                         bulkdisp=disp,volume=Volume)$dy-Conc*rate))

# two imposed boundary concentrations and rate
model2 <- function (t,Conc,pars=NULL,y.up,rate)
   return (list(tran1D.volume(Conc,y.up=y.up,flow=Flow,
                         bulkdisp=disp,volume=Volume)$dy-Conc*rate))

#====================#
# Model application  #
#====================#

# Initialising morphology: #

nbox       <- 500

# parameters defining the morphology

lengthEstuary <- 100000               # m total length of estuary
BoxLength     <- lengthEstuary/nbox
Distance      <- seq(BoxLength/2,by=BoxLength, len=nbox)

# Cross section at middle of boxes         (m2)
# cross sectional surface area: sigmoid function of estuarine distance
CrossSurfArea <- 4000 + 72000 * Distance^5 /(Distance^5+50000^5)

# Volume of boxes                          (m3)
Volume  <- CrossSurfArea*BoxLength

# Transport coefficients
disp    <- 50000   # m3/s, bulk dispersion coefficient
Flow    <- 180     # m3/s, mean river flow
```

```
# RUNNING the model:   #

Conc1     <- steady.band(runif(nbox),fun=model1,y.up=0, y.down=50,
                         rate=0,nspec=1)
Conc2     <- steady.band(runif(nbox),fun=model1,y.up=0, y.down=50,
                         rate=0.025/3600/24,nspec=1)
Conc3     <- steady.band(runif(nbox),fun=model1,y.up=50,y.down=50,
                         rate=0.025/3600/24,nspec=1)
# model 2
Conc4     <- steady.band(runif(nbox),fun=model2,y.up=50,
                         rate=0.025/3600/24,nspec=1)

#====================#
# Plotting output    #
#====================#

matplot(Distance,cbind(Conc1$y,Conc2$y,Conc3$y,Conc4$y),lwd=2,
main="tran1D.volume",xlab="distance, km",ylab="Concentration",
type="l",sub="reactive transport in estuary")
legend ("top",c("rate=0/d","rate=0.025/d","rate=0.025/d",
                "rate=0.025/d, 0-gradient downstream"),
         title="Boundaries",lty=1:5,col=1:5,lwd=2)
```

Index