Package 'ReacTran'

January 22, 2009

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:

`fiadeiro` *Upstream weighing coefficients for advection*

Description

Upstream weighing coefficients used in the finite difference scheme for advection. These AFVW (advective finite difference weights) reduce numerical dispersion while conserving positivity.

Usage

```
fiadeiro(v, D, dx.aux, grid=list(dx.aux=dx.aux))
```

Arguments

`v`	advective velocity, either one value or a vector of length N+1 with N the number of grid cells [L/T]
`D`	diffusion coefficient, either one value or a vector of length N+1 [L2/T]
`dx.aux`	auxiliary vector containing the distances between the locations where the concentration is defined (i.e. the grid cell centers and the two outer interfaces), either one value or a vector of length N+1
`grid`	discretization grid 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 concentrations. 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>, Filip Meysman <f.meysman@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

```
#================================================================================
# Model formulation (set of differential equations)
#================================================================================

# Test model to evaluate different weighing coefficients for finite difference
# formulas and their effect on munerical diffusion. The model describes the
# decay of organic carbon (OC) as it settles through the ocean water column.

model <- function (time,OC,pars,AFDW=1)
{
dOC <- tran.1D(OC,flux.up=F_OC,D=D.eddy,v=v.sink,AFDW=AFDW,dx=dx)$dC - k*OC
return(list(dOC))
}
#================================================================================
# Parameter set
#================================================================================

L <- 1000         # water depth model domain [m]
x.att <- 200      # attenuation depth [m]
v.sink.0 <- 10   # sinking velocity at the surface [m d-1]
D.eddy <- 10       # eddy diffusion coefficient [m2 d-1]
F_OC <- 10        # particle flux [mol m-2 d-1]
k <- 0.1          # decay coefficient [d-1]

#================================================================================
# Model solution for a coarse grid (10 grid cells)
#================================================================================

# Setting up the grid
N <- 10                              # number of grid layers
dx <- L/N                            # thickness of boxes[m]
dx.aux <- rep(dx,(N+1))              # auxilliary grid vector
x.int <- seq(from=0,to=L,by=dx)      # water depth at box interfaces [m]
x.mid <- seq(from=dx/2,to=L,by=dx)   # water depth at box centres [m]

# Exponetially declining sink velocity
v.sink <- v.sink.0*exp(-x.int/x.att)  # sink velocity [m d-1]
Pe <- v.sink*dx/D.eddy

# Calculate the weighing coefficients
AFDW <- fiadeiro(v=v.sink,D=D.eddy,dx.aux=dx.aux)

par(mfrow=c(2,1),cex.main=1.2,cex.lab=1.2)

matplot(Pe,x.int,log="x",pch=19,ylim=c(L,0),xlim=c(0.1,1000),
```

```
xlab="",ylab="depth [m]",main=expression("Peclet number"),axes=FALSE)
abline(h = 0)
axis(pos=NA, side=2)
axis(pos=0, side=3)

matplot(AFDW,x.int,pch=19,ylim=c(L,0),xlim=c(0.5,1),
xlab="",ylab="depth [m]",main=expression("AFDW coefficient"),axes=FALSE)
abline(h = 0)
axis(pos=NA, side=2)
axis(pos=0, side=3)

# Three steady-state solutions for a coarse grid based on:
# (1) backward differences (BD)
# (2) centered differences (CD)
# (3) Fiadeiro & Veronis scheme (FV)
BD <- steady.band(y=runif(N), func=model, AFDW=1.0, nspec=1)$y
CD <- steady.band(y=runif(N), func=model, AFDW=0.5, nspec=1)$y
FV <- steady.band(y=runif(N), func=model, AFDW=AFDW, nspec=1)$y
CONC <- cbind(BD,CD,FV)

windows()

par(mfrow=c(1,2))

# Plotting output
matplot(CONC,x.mid,pch=16,type="b",ylim=c(L,0),
xlab="",ylab="depth [m]",main=expression("conc (Low resolution grid)"),axes=FALSE)
abline(h = 0)
axis(pos=0, side=2)
axis(pos=0, side=3)
legend("bottomright",legend=c("backward diff","centred diff","Fiadeiro&Veronis"),
col=c(1:3),lty=c(1:3),pch=c(16,16,16))

#=========================================================================
# Model solution for a fine grid (1000 grid cells)
#=========================================================================

# Setting up the grid
N <- 1000                              # number of grid layers
dx <- L/N                             # thickness of boxes[m]
dx.aux <- rep(dx,(N+1))               # auxilliary grid vector
x.int <- seq(from=0,to=L,by=dx)       # water depth at box interfaces [m]
x.mid <- seq(from=dx/2,to=L,by=dx)    # water depth at box centres [m]

# Exponetially declining sink velocity
v.sink <- v.sink.0*exp(-x.int/x.att)  # sink velocity [m d-1]
Pe <- v.sink*dx/D.eddy

# Calculate the weighing coefficients
AFDW <- fiadeiro(v=v.sink,D=D.eddy,dx.aux=dx.aux)

# Three steady-state solutions for a coarse grid based on:
# (1) backward differences (BD)
# (2) centered differences (CD)
# (3) Fiadeiro & Veronis scheme (FV)
BD <- steady.band(y=runif(N), func=model, AFDW=1.0, nspec=1)$y
CD <- steady.band(y=runif(N), func=model, AFDW=0.5, nspec=1)$y
```

```
FV <- steady.band(y=runif(N), func=model, AFDW=AFDW, nspec=1)$y
HR_CONC <- cbind(BD,CD,FV)

# Plotting output
matplot(HR_CONC,x.mid,pch=16,type="b",ylim=c(L,0),
xlab="",ylab="depth [m]",main=expression("conc (High resolution grid)"),axes=FALSE)
abline(h = 0)
axis(pos=0, side=2)
axis(pos=0, side=3)
legend("bottomright",legend=c("backward diff","centred diff","Fiadeiro&Veronis"),
col=c(1:3),lty=c(1:3),pch=c(16,16,16))

# Results and conclusions:
# - For this fine grid all three solutions are identical
# - For the coarse grid, the BD and FV solutions show numerical dispersion
```

`setup.1D.advection` *Calculates the advective velocities of the pore water and the solid phase in a sediment assuming steady state compaction*

Description

This routine calculates the advective velocities of the pore water and the solid phase in a sediment based on the assumption of steady state compaction. The velocities of the pore water ('u') and the solid phase ('v') are calculated in the middle ('mid') of the grid cells and the interfaces ('int'). One requires the specification of the porosity at the interface and at infinite depth, as well as the advective velocity of teh solid phase at the interface.

Usage

```
setup.1D.advection(v.0 = NULL, v.inf = NULL, por.0, por.inf, por.grid)
```

Arguments

`v.0`	Advective velocity of the solid phase at the sediment-water interface (also referred to as the sedimentation velocity)
`v.inf`	Advective velocity of the solid phase at infinite depth (also referred to as the burial velocity)
`por.0`	Porosity at the sediment-water interface
`por.inf`	Porosity at infinite depth
`por.grid`	Porosity profile specified as a 1D grid property (see 'setup.1D.prop' for details on the structure of this list)

Value

A list containing:

`u`	list with pore water advective velocities at the middle of the grid cells ('umid')$and at the interfaces ('$
`v`	list with solid phase advective velocities at the middle of the grid cells ('umid')$and at the interfaces (

Author(s)

Filip Meysman <f.meysman@nioo.knaw.nl>, Karline Soetaert <k.soetaert@nioo.knaw.nl>

Examples

```
# setup of the 1D grid
L <- 10
grid <- setup.1D.grid(x.up=0,L=L,N=20)

# attaching an exponential porosity profile to the 1D grid

exp.profile <- function(x,y.0=NULL,y.inf=NULL,x.att=NULL)
{return(y.inf + (y.0-y.inf)*exp(-x/x.att))}

por.grid <- setup.1D.prop(func=exp.profile,grid=grid,y.0=0.9,y.inf=0.5,x.att=3)

# calculate the advective velocities

dummy <- setup.1D.advection(v.0 = 1, por.0=0.9, por.inf=0.5, por.grid = por.grid)
u.grid <-dummy$u
v.grid <-dummy$v

# plotting the results

par(mfrow=c(2,1),cex.main=1.2,cex.lab=1.2)

matplot(por.grid$int,grid$x.int,pch=19,ylim=c(L,0), xlim=c(0,1),
xlab="",ylab="depth [cm]",main=expression("porosity"),axes=FALSE)
abline(h = 0)
axis(pos=0, side=2)
axis(pos=0, side=3)

matplot(u.grid$int,grid$x.int,type="l",lwd=2,col="blue",ylim=c(L,0), xlim=c(0,max(u.grid$
xlab="",ylab="depth [cm]",main=expression("advective velocity [cm yr-1]"),axes=FALSE)
abline(h = 0)
axis(pos=0, side=2)
axis(pos=0, side=3)
lines(v.grid$int,grid$x.int,lwd="2",col="red")
legend(x="bottomright", legend=c("pore water","solid phase"),col=c("blue","red"),lwd=c(2,
```

`setup.1D.grid` *Creation of a one-dimensional finite difference grid*

Description

Subdivides the one-dimensional model domain into several zones with each N grid cells. The resulting grid structure can be used in the transport routines. The grid structure is characterized by the position of the middle of the grid cells (x.mid) and the position of the interfaces between cells (x.int). Distances are calculated between the interfaces (dx), i.e. the thickness of the grid cells. An auxiliary set of distances is calculated (dx.aux) between the points where the concentrations are specified (at the centers of the grid cell and the two external interfaces). A more complex grid

can be specified consisting of multiple zones when using vectors as arguments. In each zone, one can control the grid resolution near the upstream and downstraem boundary. The grid resolution at the upstream interface changes according to the formula dx[i+1] = min(max.dx.1,pdx.1*dx[i]). A similar formula controls the resoltion at the downstream interface.

Usage

```
setup.1D.grid(x.up=0, x.down=NULL, L=NULL, N=NULL, dx.1=NULL, p.dx.1 = rep(1,len
```

Arguments

x.up	position of the upstream interface
x.down	position of zone endpoints, one value when the model domain covers only one zone (x.down = position of downstream interface) or a vector of length N, when the model domain is divided into several consecutive zones
L	thickness of zones, one value (model domain = one zone) or a vector of length N
N	desired number of grid cells in a zone, one value or a vector of length N
dx.1	size of first grid cell in a zone, one value or a vector of length N
p.dx.1	growth factor of the grid cell size in upper half of the zone, one value or a vector of length N. The default value is 1 (constant grid cell size)
max.dx.1	maximum grid cell size in upper half of the zone, one value or a vector of length N
dx.N	size of last grid cell in a zone, one value or a vector of length N
p.dx.N	growth factor of the grid cell size in lower half of the zone, one value or a vector of length N. The default value is 1 (constant grid cell size)
max.dx.N	maximum grid cell size in lower half of the zone, one value or a vector of length N

Value

a list containing:

x.up	position of the upstream interface
x.down	position of the downstream interface
x.mid	position of the middle of the grid cells, vector of length N
x.int	position of the interfaces of the grid cells, vector of length N+1
dx	distance between adjacent cell interfaces (thickness of grid cells), vector of length N
dx.aux	auxiliary vector containing the distance between adjacent cell centers; at the upper and lower boundary calculated as (x[1]-x.up) and (x.down-x[N]) respectively; vector of length N+1
N	the total number of grid cells

Author(s)

Filip Meysman <filip.meysman@vub.ac.be>, Karline Soetaert <k.soetaert@nioo.knaw.nl>

Examples

```
# 1D Grid: one zone, constant resolution
grid1 <- setup.1D.grid(x.up=0,L=10,N=10)

# 1D Grid: one zone, constant resolution, origin not zero
setup.1D.grid(x.up=5,x.down=10,N=10)

# 1D Grid: one zone, higher resolution near the upstream interface
setup.1D.grid(x.up=0,x.down=10,dx.1=0.1,p.dx.1=1.1)

# 1D Grid: one zone, higher resolution near the upstream and downstream interface
setup.1D.grid(x.up=0,x.down=10,dx.1=0.1,p.dx.1=1.1,dx.N=0.1,p.dx.N=1.1)

# 1D Grid: two zones, higher resolution near the upstream and downstream interface
setup.1D.grid(x.up=0,L=c(5,5),dx.1=c(0.2,0.2),p.dx.1=c(1.1,1.1),dx.N=c(0.2,0.2),p.dx.N=c(

# 1D Grid: two zones, higher resolution near the upstream and downstream interface
# the number of grid cells in each zone is imposed via N
setup.1D.grid(x.up=0,L=c(5,5),N=c(20,10),dx.1=c(0.2,0.2),p.dx.1=c(1.1,1.1),dx.N=c(0.2,0.2
```

---

setup.1D.prop *Attaches a property to a one-dimensional grid*

Description

This routine estimates the value of a given property at the middle of grid cells ('mid') and at the interfaces of the grid cells ('int'). Two possibilities are available: either specifying a mathematical function ('func') that describes the spatial dependency or by interpolation of a data series (data matrix 'xy'). For example, the routine can be used to specify the porosity, the mixing intensity or other parameters over a grid.

Usage

```
setup.1D.prop(func = (F <- function(x) return(rep(value,times=length(x)))), valu
```

Arguments

func	Function that describes the depth dependency
value	Constant value given to the property
xy	A two-column matrix where the first column ('x') provides the position, and the second column ('y') provides the values that need interpolation over the grid
type	Specifies how the interpolation should be done, one of "spline" or "linear"; only used if xy is present
grid	List specifying the 1D grid characteristics, see 'setup.1D.grid' for details on the structure of this list
...	Additional arguments that are passed on to func

Details

To interpolate, there are two options:

"spline" gives a smooth profile, but sometimes generates strange profiles - always check the result!

"linear" gives a segmented profile

Value

A list containing:

mid	property value at the middle of the grid cells, vector of length N
int	property value at at the interface of the grid cells, vector of length N+1

Author(s)

Karline Soetaert <k.soetaert@nioo.knaw.nl>, Filip Meysman <filip.meysman@vub.ac.be>

Examples

```
# Construction of the 1D grid

grid <- setup.1D.grid(x.up=0,L=10,N=10)

# Porosity profile via function specification

exp.profile <- function(x,y.0=NULL,y.inf=NULL,x.att=NULL)
{return(y.inf + (y.0-y.inf)*exp(-x/x.att))}

P.func <- setup.1D.prop(func=exp.profile,grid=grid,y.0=0.9,y.inf=0.5,x.att=3)

# Porosity profile via data series interpolation

P.data <- matrix(ncol=2,data=c(0,3,6,10,0.9,0.65,0.55,0.5))
P.spline <- setup.1D.prop(xy=P.data,grid=grid)
P.linear <- setup.1D.prop(xy=P.data,grid=grid,type="linear")

# Plot different profiles

plot(grid$x.int,P.func$int,type="l",main="setup.prop, function evaluation")
points(P.data,cex=1.5,pch=16)
lines(grid$x.int,P.spline$int,lty="dashed")
lines(grid$x.int,P.linear$int,lty="dotdash")
```

---

setup.2D.grid *Creation of a two-dimensional finite difference grid*

Description

Creates a finite difference grid over a two-dimensional model domain staring from two separate one-dimensional grids (for example as created by setup.1D.grid).

Usage

```
setup.2D.grid(x.grid=NULL, y.grid=NULL)
```

Arguments

x.grid	list containing the one-dimensional grid in the vertical direction - see setup.1D.grid for the structure of the list
y.grid	list containing the one-dimensional grid in the horizontal direction - see setup.1D.grid for the structure of the list

Value

a list containing:

x.up	vertical position of the upper interface
x.down	vertical position of the lower interface
x.mid	vertical position of the middle of the grid cells, vector of length N
x.int	vertical position of the horizontal interfaces of the grid cells, vector of length N+1
dx	distance between adjacent cell interfaces (thickness of grid cells), vector of length N
dx.aux	auxiliary vector containing the distance between adjacent cell centers; at the upper and lower boundary calculated as (x[1]-x.up) and (x.down-x[N]) respectively; vector of length N+1
x.N	the total number of grid cells in the vertical direction
y.left	horizontal position of the left interface
y.right	horizontal position of the right interface
y.mid	horizontal position of the middle of the grid cells, vector of length N
y.int	horizontal position of the vertical interfaces of the grid cells, vector of length N+1
dy	distance between adjacent cell interfaces (thickness of grid cells), vector of length N
dy.aux	auxiliary vector containing the distance between adjacent cell centers; at the left and right boundary calculated as (y[1]-xyup) and (y.down-y[N]) respectively; vector of length N+1
y.N	the total number of grid cells in the horizontal direction

Author(s)

Filip Meysman <filip.meysman@vub.ac.be>, Karline Soetaert <k.soetaert@nioo.knaw.nl>

Examples

```
# test of the setup.2Dgrid functionality
x.grid <- setup.1D.grid(x.up=0,L=10,N=5)
y.grid <- setup.1D.grid(x.up=0,L=20,N=10)
grid2D <- setup.2D.grid(x.grid,y.grid)
```

---

setup.2D.prop *Attaches a property to a two-dimensional grid*

Description

This routine calculates the value of a given property at the middle of grid cells ('mid') and at the interfaces of the grid cells ('int'). Two possibilities are available: either specifying a mathematical function ('func') that describes the spatial dependency or by interpolation of a data series (data matrix 'xy'). For example, the routine can be used to specify the porosity, the mixing intensity or other parameters over a grid.

Usage

```
setup.2D.prop(func = (F <- function(x) return(rep(value,times=length(x)))), valu
```

Arguments

func	Function that describes the spatial dependency
value	Constant value given to the property
grid	List specifying the 2D grid characteristics, see 'setup.2D.grid' for details on the structure of this list
...	Additional arguments that are passed on to func

Value

A list containing:

mid	property value at the middle of the grid cells, NxN matrix
x.int	property value at the horizontal interfaces of the grid cells, (N+1)xN matrix
y.int	property value at the vertical interfaces of the grid cells, Nx(N+1) matrix

Author(s)

Filip Meysman <f.meysman@nioo.knaw.nl>, Karline Soetaert <k.soetaert@nioo.knaw.nl>

Examples

```
# Inverse quadratic function
inv.quad <- function(x,y,a=NULL,b=NULL)
{
return(1/((x-a)^2+(y-b)^2))
}

# Construction of the 2D grid
x.grid <- setup.1D.grid(x.up=0,L=10,N=10)
y.grid <- setup.1D.grid(x.up=0,L=10,N=10)
grid2D <- setup.2D.grid(x.grid,y.grid)

# Attaching the inverse quadratic function to the 2D grid
setup.2D.prop(func=inv.quad,grid=grid2D,a=5,b=5)
```

---

tran.1D *General one-dimensional advective-diffusive transport*

Description

Estimates the transport term (i.e. the rate of change of the concentration due to diffusion and advection) in a one-dimensional model where the interfaces between grid cells can have a variable cross-sectional area.

Usage

```
tran.1D(C, C.up=C[1],C.down=C[length(C)],
flux.up=NA, flux.down=NA, a.bl.up=NULL, C.bl.up=NULL, a.bl.down=NULL, C.bl.down=
D=NULL, D.grid=list(int=D), v=0, v.grid=list(int=v), AFDW=1, AFDW.grid=list(int=
VF=1, VF.grid=list(int=rep(VF,length.out=(length(C)+1)),mid=0.5*(rep(VF,length.o
A=1, A.grid=list(int=rep(A,length.out=(length(C)+1)),mid=0.5*(rep(A,length.out=(
dx=NULL,grid=list(dx=rep(dx,length.out=length(C)),dx.aux=0.5*(c(0,rep(dx,length.
full.check = FALSE, full.output = FALSE)
```

Arguments

C	concentration, expressed per unit volume, defined at the centre of each grid cell. A vector of length N [M/L3]
C.up	concentration at upstream boundary. One value [M/L3]
C.down	concentration at downstream boundary. One value [M/L3]
flux.up	flux across the upstream boundary, positive = INTO model domain. One value [M/L2/T]
flux.down	flux across the downstream boundary, positive = OUT of model domain. One value [M/L2/T]
a.bl.up	transfer coefficient across the upstream boundary layer. Flux = a.bl.up*(C.bl.up-C[1]). One value [L/T]
C.bl.up	concentration at the top of the upstream boundary layer. One value [M/L3]
a.bl.down	transfer coefficient across the downstream boundary layer; Flux = a.bl.down*(C[N]-C.bl.down). One value [L/T]
C.bl.down	concentration at the top of the downstream boundary layer. One value [M/L3]
D	diffusion coefficient, defined on grid cell interfaces. One value or a vector of length N+1 [L2/T]
D.grid	diffusion coefficient packaged as a grid list; the list contains at least the element 'int' (see 'setup.1D.prop') [L2/T]
v	advective velocity in the x-axis direction, defined on grid cell interfaces. Can be positive (downstream flow) or negative (upstream flow). One value or a vector of length N+1 [L/T]
v.grid	advective velocity packaged as a grid list; the list contains at least the element 'int' (see 'setup.1D.prop') [L/T]
AFDW	weight used in the finite difference scheme for advection, defined on grid cell interfaces; backward = 1, centred = 0.5, forward = 0; default is backward. One value or a vector of length N+1 [-]
AFDW.grid	AFDW coefficients packaged as a grid list; the list contains at least the element 'int' (see 'setup.1D.prop') [-]
VF	Volume fraction defined at the grid cell interfaces, one value or a vector of length N+1 [L2]
VF.grid	Volume fraction packaged as a grid list; the list contains at least the elements 'int' and 'mid' (see 'setup.1D.prop') [L2]
A	Interface area defined at the grid cell interfaces, one value or a vector of length N+1 [L2]
A.grid	Interface area packaged as a grid list; the list contains at least the elements 'int' and 'mid' (see 'setup.1D.prop') [L2]

dx	distance between adjacent cell interfaces (thickness of grid cells). One value or vector of length N [L]
grid	discretization grid, a list containing at least elements 'dx' and 'dx.aux' (see 'setup.1D.grid') [L]
full.check	logical flag enabling a full check of the consistency of the arguments (default = FALSE, TRUE slows down execution by 50 percent)
full.output	logical flag enabling a full return of the output (default = FALSE, TRUE slows down execution by 20 percent)

Details

The **boundary conditions** are either

(1) zero-gradient

(2) fixed concentration

(3) convective boundary layer

(4) fixed flux

The above order also shows the priority. The default condition is the zero gradient. The convective boundary layer condition overrules the fixed concentration. The fixed flux overrules all other specifications.

Transport properties The *diffusion coefficient* (D), the *advective velocity* (v), the *volume fraction* (VF), the *interface surface* (A), and the *advective finite difference weight* (AFDW) can be either one value or a vector. When a vector, this vector must be of length N+1, defined at all grid cell interfaces, including upper and lower boundary.

The **finite difference grid** (grid) is specified either:

as a grid list, as generated by `setup.1D.grid` or

by the parameter dx representing the thickness of the grid cells (one value or a vector of length N+1)

Value

when (full.output = FALSE) then a list containing:

dC	The rate of change of the concentration C due to transport, defined in the centre of each grid cell [M/L3/T]
dC	The rate of change of the concentration C due to transport, defined in the centre of each grid cell [M/L3/T]
C.up	Concentration at the upstream interface. One value [M/L3]
C.down	Concentration at the downstream interface. One value [M/L3]
adv.flux	Advective flux across at the interface of each grid cell. A vector of length N+1 [M/L2/T]
dif.flux	Diffusive flux across at the interface of each grid cell. A vector of length N+1 [M/L2/T]
flux	Total flux across at the interface of each grid cell. A vector of length N+1 [M/L2/T]
flux.up	Flux across the upstream boundary, positive = INTo model domain. One value [M/L2/T]
flux.down	Flux across the downstream boundary, positive = OUT of model domain. One value [M/L2/T]

Note

The advective-diffusion equation is not checked for mass conservation. Sometimes, this is not an issue, for instance when v represents a sinking velocity of particles or a swimming velocity of organisms. In others cases however, mass conservation needs to be accounted for. To ensure mass conservation, the advective velocity must obey certain continuity constraints: in essence the product of the volume fraction (VF), interface surface (A) and advective velocity (v) should be constant. In sediments, one can use 'setup.1D.advection' to ensure that the advective velocities for the pore water and solid phase meet these constraints.

Author(s)

Filip Meysman <f.meysman@nioo.knaw.nl>, 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 1: O2 and OC consumption in sediments        ######
##============================================================================

# this example uses only the volume fractions
# in the reactive transport term

#====================#
# Model formulation  #
#====================#

# Monod consumption of oxygen (O2)

O2.model <- function (t=0,O2,pars=NULL)
{
tran <- tran.1D(C=O2,C.up=C.ow.O2,D.grid=D.grid,v.grid=v.grid,VF.grid=por.grid,grid=grid)
reac <- - R.O2*(O2/(Ks+O2))
return(list(dCdt = tran+reac))
}

# First order consumption of organic carbon (OC)

OC.model <- function (t=0,OC,pars=NULL)
{
tran <- tran.1D(C=OC,flux.up=F.OC,D.grid=Db.grid,v.grid=v.grid,VF.grid=svf.grid,grid=gric
reac <- - k*OC
return(list(dCdt = tran + reac))
}

#======================#
# Parameter definition #
#======================#

# Parameter values

F.OC    <- 25              # input flux organic carbon [micromol cm-2 yr-1]
```

```
C.ow.O2 <- 0.25            # concentration O2 in overlying water [micromol cm-3]
por     <- 0.8             # porosity
D       <- 400             # diffusion coefficient O2 [cm2 yr-1]
Db      <- 10              # mixing coefficient sediment [cm2 yr-1]
v       <- 1               # advective velocity [cm yr-1]
k       <- 1              # decay constant organic carbon [yr-1]
R.O2    <- 10               # O2 consumption rate [micromol cm-3 yr-1]
Ks      <- 0.005           # O2 consumption saturation constant

# Grid definition

L <- 10   # depth of sediment domain [cm]
N <- 100  # number of grid layers
grid <- setup.1D.grid(x.up=0,L=L,N=N)

# Volume fractions

por.grid <- setup.1D.prop(value=por,grid=grid)
svf.grid <- setup.1D.prop(value=(1-por),grid=grid)
D.grid <- setup.1D.prop(value=D,grid=grid)
Db.grid <- setup.1D.prop(value=Db,grid=grid)
v.grid <- setup.1D.prop(value=v,grid=grid)

#====================#
# Model solution     #
#====================#

# Initial conditions + simulation O2

O2 <- rep(0,length.out=N)
O2 <- steady.band(y=O2, func=O2.model, nspec=1)$y

# Initial conditions + simulation OC

OC <- rep(0,length.out=N)
OC <- steady.band(y=OC, func=OC.model, nspec=1)$y

# Plotting output

par(mfrow=c(1,2))

matplot(O2,grid$x.mid,pch=16,type="b",ylim=c(L,0), xlim=c(min(0,min(O2)),max(O2)),
xlab="",ylab="depth [cm]",main=expression("O2 concentration"),axes=FALSE)
abline(h = 0)
axis(pos=0, side=2)
axis(pos=0, side=3)

matplot(OC,grid$x.mid,pch=16,type="b",ylim=c(L,0), xlim=c(min(0,min(OC)),max(OC)),
xlab="",ylab="depth [cm]",main=expression("OC concentration"),axes=FALSE)
abline(h = 0)
axis(pos=0, side=2)
axis(pos=0, side=3)

#####################################################################
###### EXAMPLE 2: O2 in a cylindrical and spherical organism  ######
#####################################################################
```

```
# this example uses only the surface areas
# in the reactive transport term

#====================#
# Model formulation  #
#====================#

# the numerical model - rate of change=transport-consumption
Cylinder.Model <- function(time,O2,pars)
      return (list(tran.1D(C=O2,C.down=BW,D=Da,A=A.cyl,dx=dx)$dC-Q))

Sphere.Model <- function(time,O2,pars)
      return (list(tran.1D(C=O2,C.down=BW,D=Da,A=A.sphere,dx=dx)$dC-Q))

#======================#
# Parameter definition #
#======================#

# parameter values

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 numerical model

N  <- 40                           # layers in the body
dx <- R/N                          # thickness of each layer
x.mid <- seq(dx/2,by=dx,length.out=N) # distance of center to mid-layer
x.int <- seq(0,by=dx,length.out=N+1)  # distance to layer interface

# Cylindrical surfaces
A.cyl   <- 2*pi*x.int*L               # surface at mid-layer depth

# Spherical surfaces
A.sphere <- 4*pi*x.int^2               # surface of sphere, at each mid-layer

#====================#
# Model solution     #
#====================#

# the analytical solution of 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)

# solve the model numerically for a cylinder
O2.cyl <- steady.1D (runif(N),func=Cylinder.Model,nspec=1,atol=1e-10)$y

# solve the model numerically for a sphere
O2.sphere <- steady.1D (runif(N),func=Sphere.Model,nspec=1,atol=1e-10)$y

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

```
windows()
par(mfrow=c(1,1))

plot(x.mid,O2.cyl,xlab="distance from centre, cm",ylab="mmol/m3",
main="tran.1D",sub="diffusion-reaction in a cylinder and sphere")
lines(x.mid, cylinder(Da,Q,BW,R,x.mid))

points(x.mid, O2.sphere, pch=18,col="red" )
lines(x.mid, sphere(Da,Q,BW,R,x.mid),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"))

######################################################################
###### EXAMPLE 3: O2 consumption in a spherical aggregate   ######
######################################################################

# this example uses both the surface areas and the volume fractions
# in the reactive transport term

#====================#
# Model formulation  #
#====================#

Aggregate.Model <- function(time,O2,pars)
{
tran <- tran.1D(C=O2,C.down=C.ow.O2,D.grid=D.grid,A.grid=A.grid,VF.grid=por.grid,grid=gri
reac <- - R.O2*(O2/(Ks+O2))*(O2>0)
return(list(dCdt = tran+reac))
}

#======================#
# Parameter definition #
#======================#

# Parameters

C.ow.O2 <- 0.25            # concentration O2 in overlying water [micromol cm-3]
por     <- 0.8             # porosity
D       <- 400             # diffusion coefficient O2 [cm2 yr-1]
v       <- 0               # advective velocity [cm yr-1]
R.O2    <- 1000000         # O2 consumption rate [micromol cm-3 yr-1]
Ks      <- 0.005           # O2 consumption saturation constant [micromol cm-3]

# Grid definition
R <- 0.025           # radius of the agggregate [cm]
N <- 100             # number of grid layers
grid <- setup.1D.grid(x.up=0,L=R,N=N)

# Volume fractions

por.grid <- setup.1D.prop(value=por,grid=grid)
D.grid <- setup.1D.prop(value=D,grid=grid)

# Surfaces
```

```
A.mid <- 4*pi*grid$x.mid^2               # surface of sphere, at each mid-layer
A.int <- 4*pi*grid$x.int^2               # surface of sphere, at each mid-layer
A.grid=list(int=A.int,mid=A.mid)

#====================#
# Model solution     #
#====================#

# Numerical solution: staedy state

O2.agg <- steady.1D (runif(N),func=Aggregate.Model,nspec=1,atol=1e-10)$y

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

windows()
par(mfrow=c(1,1))

plot(grid$x.mid,O2.agg,xlab="distance from centre, cm",ylab="mmol/m3",
main="Diffusion-reaction of O2 in a spherical aggregate")
legend ("bottomright",pch=c(1,18),lty=1,col=c("black"),
        c("O2 concentration"))
```

`tran.1D.river` *1-D advective-diffusive transport in a river*

Description

Estimates the transport term (i.e. the rate of change of the concentration due to diffusion and advection) in a one-dimensional model of a river with whose cross sectional area is not necessarily constant, e.g. estuaries

Usage

```
tran.1D.river(C, C.up=C[1], C.down=C[length(C)],
C.lat=0, C.lat.grid=list(mid=rep(C.lat,length.out=length(C))),
F.up=NA, F.down=NA, F.lat=NA, F.lat.grid=list(mid=rep(F.lat,length.out=length(C)
Disp=NULL, Disp.grid=list(int=rep(Disp,length.out=(length(C)+1))),
flow = 0, flow.grid=list(int=rep(flow,length.out=(length(C)+1))),
flow.lat=0, flow.lat.grid=list(mid=rep(flow.lat,length.out=length(C))),
V=NULL,V.grid=list(mid=rep(V,length.out=length(C))))
```

Arguments

`C`	Tracer concentration, defined at the centre of the grid cells. A vector of length N [M/L3]
`C.up`	Tracer concentration at the upstream interface. One value [M/L3]
`C.down`	Tracer concentration at downstream interface. One value [M/L3]
`C.lat`	Tracer concentration of lateral input, defined at grid cell centres. One value or a vector of length N [M/L3]

C.lat.grid	Tracer concentration in lateral input defined as grid property
F.up	Total tracer input at the upstream interface. One value [M/T]
F.down	Total tracer input at downstream interface. One value [M/T]
F.lat	Total lateral tracer input, defined at grid cell centres. One value or a vector of length N [M/T]
F.lat.grid	Total lateral tracer input defined as grid property
Disp	BULK dispersion coefficient (=disp*A.int/dx.int), defined on grid cell interfaces. One value or a vector of length N+1, [L3/T]
Disp.grid	BULK dispersion coefficient defined as grid property
flow	Water flow rate, defined on grid cell interfaces. One value or a vector of length N+1, [L3/T]
flow.grid	Water flow rate defined as grid property
flow.lat	Lateral water flow rate into each volume box, defined at grid cell centres. One value or a vector of length N, [L3/T]
flow.lat.grid	Lateral water flow rate defined as grid property
V	Grid cell volume, defined at grid cell centres. One value or a vector of length N [L3]
V.grid	Grid cell volume defined as grid property

Details

The **boundary conditions** are of type

1. zero-gradient (default)
2. fixed concentration
3. fixed input

The *bulk dispersion coefficient* (Disp), and *flow rate* (flow) can be either one value or a vector of length N+1, defined at all grid cell interfaces, including upstream and downstream boundary.

The spatial discretisation is given by the volume of each box (V), which can be one value or a vector of length N+1, defined at the centre of each grid cell.

Value

dC	The rate of change of the concentration C due to transport, defined in the centre of each grid cell [M/L3/T]
F	Mass flow across at the interface of each grid cell. A vector of length N+1 [M/T]
F.up	Mass flow across the upstream boundary, positive = INTO model domain. One value [M/T]
F.down	Mass flow across the downstream boundary, positive = OUT of model domain. One value [M/T]
F.lat	Lateral mass input per volume box, positive = INTO model domain. A vector of length N [M/T]

Author(s)

Filip Meysman <f.meysman@nioo.knaw.nl>, 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

tran.1D

Examples

```
#####################################################################
#  EXAMPLE : organic carbon (OC) decay in a widening river       #
#####################################################################

# Two scenarios are simulated: the baseline includes only input
# of organic matter upstream. The second scenario simulates the
# input of an important side river half way the estuary.

#====================#
# Model formulation  #
#====================#

river.model <- function (t=0,OC,pars=NULL)
{
tran <- tran.1D.river(C=OC,F.up=F.OC,F.lat=F.lat,Disp=Disp,flow=flow,V=Volume)$dC
reac <- - k*OC
return(list(dCdt = tran + reac))
}

#======================#
# Parameter definition #
#======================#

# Initialising morphology estuary:

nbox          <- 500     # number of grid cells
lengthEstuary <- 100000  # length of estuary [m]
BoxLength     <- lengthEstuary/nbox # [m]
Distance      <- seq(BoxLength/2, by=BoxLength, len=nbox) # [m]

# Cross sectional area: sigmoid function of estuarine distance [m2]
CrossArea <- 4000 + 72000 * Distance^5 /(Distance^5+50000^5)

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

# Transport coefficients
Disp    <- 1000   # m3/s, bulk dispersion coefficient
flow    <- 180    # m3/s, mean river flow

F.OC    <- 180               # input organic carbon [mol s-1]
F.lat.0 <- F.OC              # lateral input organic carbon [mol s-1]

k       <- 10/(365*24*3600)  # decay constant organic carbon [s-1]
```

```
#====================#
# Model solution     #
#====================#

F.lat <- rep(0,length.out=nbox)
Conc1 <- steady.band(runif(nbox),fun=river.model,nspec=1)$y
F.lat <- F.lat.0*dnorm(x=Distance/lengthEstuary, mean = Distance[nbox/2]/lengthEstuary, s
Conc2 <- steady.band(runif(nbox),fun=river.model,nspec=1)$y

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

matplot(Distance/1000,cbind(Conc1,Conc2),lwd=2,
main="Organic carbon decay in an estuary",xlab="distance [km]",ylab="OC Concentration [mM
type="l")
legend ("topright",lty=1,col=c("black","red"),
        c("baseline","with lateral input"))
```

`tran.2D` *General two-dimensional advective-diffusive transport*

Description

Estimates the transport term (i.e. the rate of change of the concentration due to diffusion and advection) in a two-dimensional rectangular model domain where the interfaces between grid cells can have a variable cross-sectional area.

Usage

```
tran.2D(C, C.up=C[1,], C.down=C[nrow(C),], C.left=C[,1], C.right=C[,ncol(C)],
flux.up=NA, flux.down=NA, flux.left=NA, flux.right=NA,
a.bl.up=NULL, C.bl.up=NULL, a.bl.down=NULL, C.bl.down=NULL, a.bl.left=NULL, C.bl
D.x=NULL, D.y=D.x, D.grid=list(x.int=matrix(data=D.x,nrow=(nrow(C)+1),ncol=ncol(
v.x=0, v.y=v.x, v.grid=list(x.int=matrix(data=v.x,nrow=(nrow(C)+1),ncol=ncol(C))
AFDW.x=1, AFDW.y=AFDW.x, AFDW.grid=list(x.int=matrix(data=AFDW.x,nrow=(nrow(C)+1
VF.x=1, VF.y=VF.x, VF.grid=list(x.int=matrix(data=VF.x,nrow=(nrow(C)+1),ncol=nco
dx=NULL, dy=NULL, grid=list(dx=rep(dx,length.out=nrow(C)),dx.aux=0.5*(c(0,rep(dx
full.check = FALSE, full.output = FALSE)
```

Arguments

`C`	Concentration, expressed per unit volume, defined at the centre of each grid cell. A NxM matrix [M/L3]
`C.up`	Concentration at upstream boundary. A vector of length M [M/L3]
`C.down`	Concentration at downstream boundary. A vector of length M [M/L3]
`C.left`	Concentration at left boundary. A vector of length N [M/L3]
`C.right`	Concentration at right boundary. A vector of length N [M/L3]
`flux.up`	Flux across the upstream boundary, positive = INTO model domain. A vector of length M [M/L2/T]

flux.down	Flux across the downstream boundary, positive = OUT of model domain. A vector of length M [M/L2/T]
flux.left	Flux across the left boundary, positive = INTO model domain. A vector of length N [M/L2/T]
flux.right	Flux across the right boundary, positive = OUT of model domain. A vector of length N [M/L2/T]
a.bl.up	Transfer coefficient across the upstream boundary layer. Flux = a.bl.up*(C.bl.up-C[1,]). One value [L/T]
C.bl.up	Concentration at the top of the upstream boundary layer. A vector of length M [M/L3]
a.bl.down	Transfer coefficient across the downstream boundary layer; Flux = a.bl.down*(C[N,]-C.bl.down). One value [L/T]
C.bl.down	Concentration at the top of the downstream boundary layer. A vector of length M [M/L3]
a.bl.left	Transfer coefficient across the left boundary layer. Flux = a.bl.left*(C.bl.left-C[,1]). One value [L/T]
C.bl.left	Concentration at the top of the left boundary layer. A vector of length N [M/L3]
a.bl.right	Transfer coefficient across the right boundary layer; Flux = a.bl.right*(C[,M]-C.bl.right). One value [L/T]
C.bl.right	Concentration at the top of the right boundary layer. A vector of length N [M/L3]
D.x	diffusion coefficient in x-direction, defined on grid cell interfaces. One value or an (N+1)x(M) matrix [L2/T]
D.y	diffusion coefficient in y-direction, defined on grid cell interfaces. One value or an (N)x(M+1) matrix [L2/T]
D.grid	diffusion coefficient packaged as a grid list; the list contains at least the elements 'x.int' and 'y.int' (see 'setup.2D.prop') [L2/T]
v.x	advective velocity in the x-direction, defined on grid cell interfaces. Can be positive (downstream flow) or negative (upstream flow). One value or an (N+1)x(M) matrix [L/T]
v.y	advective velocity in the y-direction, defined on grid cell interfaces. Can be positive (downstream flow) or negative (upstream flow). One value or an (N)x(M+1) matrix [L/T]
v.grid	advective velocity packaged as a grid list; the list contains at least the elements 'x.int' and 'y.int' (see 'setup.2D.prop') [L/T]
AFDW.x	weight used in the finite difference scheme for advection in the x-direction, defined on grid cell interfaces; backward = 1, centred = 0.5, forward = 0; default is backward. One value or an (N+1)x(M) matrix [-]
AFDW.y	weight used in the finite difference scheme for advection in the y-direction, defined on grid cell interfaces; backward = 1, centred = 0.5, forward = 0; default is backward. One value or an (N)x(M+1) matrix [-]
AFDW.grid	AFDW coefficients packaged as a grid list; the list contains at least the elements 'x.int' and 'y.int' (see 'setup.2D.prop') [-]
VF.x	Volume fraction at the grid cell interfaces in the x-direction. One value or an (N+1)x(M) matrix [L2]
VF.y	Volume fraction at the grid cell interfaces in the y-direction. One value or an (N)x(M+1) matrix [L2]

VF.grid	Volume fraction packaged as a grid list; the list contains at least the elements 'x.int', 'y.int', 'x.mid' and 'y.mid' (see 'setup.1D.prop') [L2]
dx	distance between adjacent cell interfaces in the x-direction (thickness of grid cells). One value or vector of length N [L]
dy	distance between adjacent cell interfaces in the y-direction (thickness of grid cells). One value or vector of length M [L]
grid	discretization grid, a list containing at least elements 'dx', 'dx.aux', 'dy', 'dy.aux' (see 'setup.2D.grid') [L]
full.check	logical flag enabling a full check of the consistency of the arguments (default = TRUE, FALSE speeds up execution by 50 percent)
full.output	logical flag enabling a full return of the output (default = FALSE, TRUE slows down execution by 20 percent)

Details

The **boundary conditions** are either

(1) zero-gradient

(2) fixed concentration

(3) convective boundary layer

(4) fixed flux

The above order also shows the priority. The default condition is the zero gradient. The convective boundary layer condition overrules the fixed concentration. The fixed flux overrules all other specifications.

Transport properties The *diffusion coefficient* (D), the *advective velocity* (v), the *volume fraction* (VF), and the *advective finite difference weight* (AFDW) can be either one value or a vector. When a vector, this vector must be of length N+1, defined at all grid cell interfaces, including upper and lower boundary.

The **finite difference grid** (grid) is specified either:

as a grid list, as generated by `setup.1D.grid` or

by the parameter dx representing the thickness of the grid cells (one value or a vector of length N+1)

Value

when (full.output = FALSE) then a list containing:

dC	The rate of change of the concentration C due to transport, defined in the centre of each grid cell [M/L3/T]
dC	The rate of change of the concentration C due to transport, defined in the centre of each grid cell [M/L3/T]
C.up	Concentration at the upstream interface. One value [M/L3]
C.down	Concentration at the downstream interface. One value [M/L3]
adv.flux	Advective flux across at the interface of each grid cell. A vector of length N+1 [M/L2/T]
dif.flux	Diffusive flux across at the interface of each grid cell. A vector of length N+1 [M/L2/T]

flux	Total flux across at the interface of each grid cell. A vector of length N+1 [M/L2/T]
flux.up	Flux across the upstream boundary, positive = INTO model domain. One value [M/L2/T]
flux.down	Flux across the downstream boundary, positive = OUT of model domain. One value [M/L2/T]

Note

The advective-diffusion equation is not necessarily mass conservative. Sometimes, this is not an issue, for instance when v represents a sinking velocity of particles or a swimming velocity of organisms. In others cases however, mass conservation needs to be accounted for. To ensure mass conservation, the advective velocity must obey certain continuity constraints: in essence the product of the interface surface (A) and advective velocity (v) should be constant. In sediments, one can use 'setup.1D.advection' to ensure that the advective velocities for the pore water and solid phase meet these constraints.

Author(s)

Filip Meysman <f.meysman@nioo.knaw.nl>, 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

```
# Parameters
F        <- 100             # input flux [micromol cm-2 yr-1]
por      <- 0.8             # constant porosity
D        <- 400             # mixing coefficient [cm2 yr-1]
v        <- 1               # advective velocity [cm yr-1]

# Grid definition
x.N <- 4   # number of cells in x-direction
y.N <- 6   # number of cells in y-direction
x.L <- 8   # domain size x-direction [cm]
y.L <- 24  # domain size y-direction [cm]
dx       <- x.L/x.N             # cell size x-direction [cm]
dy       <- y.L/y.N             # cell size y-direction [cm]

# Intial conditions
C <- matrix(nrow=x.N,ncol=y.N,data=0,byrow=FALSE)

# Boundary conditions: fixed concentration
C.up=rep(1,times=y.N)
C.down=rep(0,times=y.N)
C.left=rep(1,times=x.N)
C.right=rep(0,times=x.N)

# Only diffusion
tran.2D(full.output=TRUE,C=C,D.x=D,D.y=D,v.x=0,v.y=0,VF.x=por,VF.y=por,dx=dx,dy=dy,C.up=C

# Strong advection, backward (default), central and forward finite difference schemes
```

```
tran.2D(C=C,D.x=D,v.x=100*v,VF.x=por,dx=dx,dy=dy,C.up=C.up,C.down=C.down,C.left=C.left,C.
tran.2D(AFDW.x=0.5,C=C,D.x=D,v.x=100*v,VF.x=por,dx=dx,dy=dy,C.up=C.up,C.down=C.down,C.lef
tran.2D(AFDW.x=0,C=C,D.x=D,v.x=100*v,VF.x=por,dx=dx,dy=dy,C.up=C.up,C.down=C.down,C.left=

# Boundary conditions: fixed fluxes

flux.up=rep(200,times=y.N)
flux.down=rep(-200,times=y.N)
flux.left=rep(200,times=x.N)
flux.right=rep(-200,times=x.N)
tran.2D(C=C,D.x=D,v.x=0,VF.x=por,dx=dx,dy=dy,flux.up=flux.up,flux.down=flux.down,flux.lef

# Boundary conditions: convective boundary layer on all sides

a.bl <- 800   # transfer coefficient
C.bl.up <- rep(1,times=(y.N)) # fixed concentration on top of boundary layer
C.bl.left <- rep(1,times=(x.N)) # fixed concentration on top of boundary layer
tran.2D(full.output=TRUE,C=C,D.x=D,v.x=0,VF.x=por,dx=dx,dy=dy,C.bl.up=C.bl.up,a.bl.up=a.b

# Runtime test with and without argument checking

n.iterate <-1000

test1 <- function()
{
for (i in 1:n.iterate )
ST<-tran.2D(full.check=TRUE,C=C,D.x=D,v.x=0,VF.x=por,dx=dx,dy=dy,C.bl.up=C.bl.up,a.bl.up=
}
system.time(test1())

test2 <- function()
{
for (i in 1:n.iterate )
ST<-tran.2D(full.output=TRUE,C=C,D.x=D,v.x=0,VF.x=por,dx=dx,dy=dy,C.bl.up=C.bl.up,a.bl.up
}
system.time(test2())

test3 <- function()
{
for (i in 1:n.iterate )
ST<-tran.2D(full.output=TRUE,full.check=TRUE,C=C,D.x=D,v.x=0,VF.x=por,dx=dx,dy=dy,C.bl.up
}
system.time(test3())
```

Index