Package 'raem'

August 23, 2024

Title Analytic Element Modeling of Steady Single-Layer Groundwater Flow

Version 0.1.0

Description A model of single-layer groundwater flow in steady-state under the Dupuit-Forchheimer assumption can be created by placing elements such as wells, area-sinks and line-sinks at arbitrary locations in the flow field. Output variables include hydraulic head and the discharge vector. Particle traces can be computed numerically in three dimensions. The underlying theory is described in Haitjema (1995) <doi:10.1016/B978-0-12-316550-3.X5000-4> and references therein.

License MIT + file LICENSE

URL `https://github.com/cneyens/raem`, `https://cneyens.github.io/raem/`

BugReports `https://github.com/cneyens/raem/issues`

Imports deSolve, graphics, parallel, stats

Suggests isoband, knitr, rmarkdown, sf, terra, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.1

Collate 'aem.R' 'areasink.R' 'constant.R' 'flow-variables.R' 'linesink.R' 'tracelines.R' 'plot.R' 'state-variables.R' 'uniformflow.R' 'utils.R' 'well.R'

NeedsCompilation no

Author Cas Neyens [aut, cre, cph]

Maintainer Cas Neyens <`cas.neyens@gmail.com`>

Repository CRAN

Date/Publication 2024-08-23 09:00:02 UTC

Contents

`add_element` *Add or remove an element from an existing* `aem` *object*

Description

`add_element()` adds a new element to an `aem` object.

`remove_element()` removes an element from an `aem` object based on its name or type.

Usage

```
add_element(aem, element, name = NULL, solve = FALSE, ...)

remove_element(aem, name = NULL, type = NULL, solve = FALSE, ...)
```

Arguments

`aem`	`aem` object.
`element`	analytic element of class `element`.
`name`	optional name of the element as character. Duplicate element names in `aem` are not allowed..
`solve`	logical, should the model be solved after adding or removing the element? Defaults to `FALSE`.

`...`	ignored
`type`	class of the element(s) to remove. Either `name` or `type` should be specified in `remove_element()`.

Value

The aem model with the addition of `element` or with the removal of element(s). If `solve = TRUE`, the model is solved using `solve.aem()`. The name of the new element is taken from the `name` argument, the object name or set to `element_1` with `1` being the index of the new element in the element list. See examples.

See Also

`aem()`

Examples

```
m <- aem(k = 10, top = 10, base = 0, n = 0.2)
mnew <- add_element(m, constant(xc = 0, yc = 1000, hc = 12), name = 'rf')

# if name not supplied, tries to obtain it from object name
rf <- constant(xc = 0, yc = 1000, hc = 12)
mnew <- add_element(m, rf)

# or else sets it sequentially from number of elements
mnew <- add_element(m, constant(xc = 0, yc = 1000, hc = 12))


# add_element() adn remove_element() are pipe-friendly
mnew <- aem(k = 10, top = 10, base = 0, n = 0.2) |>
    add_element(rf, name = 'rf') |>
    add_element(headwell(xw = 0, yw = 100, rw = 0.3, hc = 8),
                name = 'headwell', solve = TRUE)


# removing elements
mnew <- remove_element(mnew, name = 'rf')
mnew <- remove_element(mnew, type = 'headwell')
```

`aem` *Create an analytic element model*

Description

`aem()` creates an analytic element model to which elements can be added

`solve.aem()` solves the system of equations as constructed by the elements in the `aem` model

`plot.element()` plots the location of an analytic element with point or line geometry.

`plot.aem()` plots the locations of all analytic elements with a point or line geometry in an `aem` object by calling `plot.element()` on them, or adds them to an existing plot.

Usage

```
aem(
  k,
  top,
  base,
  n,
  ...,
  type = c("variable", "confined"),
  verbose = FALSE,
  maxiter = 10
)

## S3 method for class 'aem'
solve(a, b, maxiter = 10, verbose = FALSE, ...)

## S3 method for class 'element'
plot(
  x,
  y = NULL,
  add = FALSE,
  pch = 16,
  cex = 0.75,
  use.widths = TRUE,
  col = "black",
  xlim,
  ylim,
  ...
)

## S3 method for class 'aem'
plot(x, y = NULL, add = FALSE, xlim, ylim, ...)
```

Arguments

`k`	numeric, hydraulic conductivity of the aquifer.
`top`	numeric, top elevation of the aquifer.
`base`	numeric, bottom elevation of the aquifer.
`n`	numeric, effective porosity of the aquifer as a fraction of total unit volume. Used for determining flow velocities with `velocity()`.
`...`	for `aem()`, objects of class `element`, or a single (named) list with `element` objects. Otherwise, ignored.
`type`	character specifying the type of flow in the aquifer, either `variable` (default) or `confined`. See details.
`verbose`	logical indicating if information during the solving process should be printed. Defaults to `FALSE`.
`maxiter`	integer specifying the maximum allowed iterations for a non-linear solution. Defaults to 10. See details.

`a`	`aem` object.
`b`	ignored
`x`	`aem` object, or analytic element of class `element` to plot. If not a point or line geometry, nothing is plotted.
`y`	ignored
`add`	logical, should the plot be added to the existing plot? Defaults to `FALSE`.
`pch`	numeric point symbol value, defaults to `16`. For a reference point, a value of `4` is used.
`cex`	numeric symbol size value, defaults to `0.75`.
`use.widths`	logical, if line elements with non-zero width are plotted, should they be plotted as polygons including the width (`TRUE`; default) or as infinitesimally thin lines (`FALSE`)?
`col`	color of element. Defaults to `'black'`.
`xlim`	numeric, plot limits along the x-axis. Required if `add = FALSE`.
`ylim`	numeric, plot limits along the y-axis. Required if `add = FALSE`.

Details

The default `type = 'variable'` allows for unconfined/confined flow, i.e. flow with variable saturated thickness. If `type = 'confined'`, the saturated thickness is always constant and equal to the aquifer thickness. This results in a linear model when head-specified elements with a resistance are used, whereas `type = 'variable'` would create a non-linear model in that case.

`solve.aem()` is called on the `aem` object before it is returned by `aem()`, which solves the system of equations.

Solving:

`solve.aem()` sets up the system of equations, and calls `solve()` to solve. If head-specified elements are supplied, an element of class `constant` as created by `constant()` (also called the reference point), should be supplied as well. Constructing an `aem` object by a call to `aem()` automatically calls `solve.aem()`.

If the system of equations is non-linear, i.e. when the flow system is unconfined (variable saturated thickness) and elements with hydraulic resistance are specified, a Picard iteration is entered. During each Picard iteration step (outer iteration), the previously solved model parameters are used to set up and solve a linear system of equations. The model parameters are then updated and the next outer iteration step is entered, until `maxiter` iterations are reached. For an linear model, `maxiter` is ignored.

Plotting:

If the analytic element has a point geometry and has a collocation point (e.g. `headwell()`), that point is also plotted with `pch = 1`.

A reference point (as created by `constant()`) is never plotted when plotting the model as it is not a hydraulic feature. Area-sinks (as created by `areasink()` or `headareasink()`) are also never plotted as they would clutter the plot. These elements can be plotted by calling `plot()` on them directly.

Value

aem() returns an object of class aem which is a list consisting of k, top, base, n, a list containing all elements with the names of the objects specified in ..., and a logical solved indicating if the model is solved.

solve.aem() returns the solved aem object, i.e. after finding the solution to the system of equations as constructed by the contained elements.

See Also

add_element() contours()

Examples

```
k <- 10
top <- 10
base <- 0
n <- 0.2
TR <- k * (top - base)

w <- well(xw = 50, yw = 0, Q = 200)
rf <- constant(xc = -500, yc = 0, h = 20)
uf <- uniformflow(gradient = 0.002, angle = -45, TR = TR)
hdw <- headwell(xw = 0, yw = 100, rw = 0.3, hc = 8)
ls <- linesink(x0 = -200, y0 = -150, x1 = 200, y1 = 150, sigma = 1)

# Creating aem ----
m <- aem(k, top, base, n, w, rf, uf, hdw, ls)

# or with elements in named list
m <- aem(k, top, base, n,
         list('well' = w, 'constant' = rf, 'flow' = uf, 'headwell' = hdw, 'river' = ls),
         type = 'confined')

# Solving ----
m <- solve(m)

# solving requires a reference point (constant) element if head-specified elements are supplied
try(
  m <- aem(k = k, top = top, base = base, n = n, w, uf, hdw)
)

# Plotting ----
plot(ls)
plot(w, add = TRUE)
plot(uf) # empty

plot(m, xlim = c(-500, 500), ylim = c(-250, 250))

xg <- seq(-500, 500, length = 200)
yg <- seq(-250, 250, length = 100)
```

```
contours(m, x = xg, y = yg, col = 'dodgerblue', nlevels = 20)
plot(m, add = TRUE)
```

areasink — *Create a circular area-sink analytic element with specified recharge*

Description

`areasink()` creates a circular area-sink analytic element with constant, uniform specified recharge.

Usage

```
areasink(xc, yc, N, R, location = c("top", "base"), ...)
```

Arguments

`xc`	numeric, x location of the center of the area-sink.
`yc`	numeric, y location of the center of the area-sink.
`N`	numeric, uniform constant leakage value (positive is into aquifer) in length per time.
`R`	numeric, radius of the circular area-sink.
`location`	character, either `top` (default) or `base` specifying the vertical position of the area-sink.
`...`	ignored

Details

Area-sinks can be used to simulate areal recharge or seepage at the aquifer top, or leakage into or out of the aquifer at its base. The `location` argument is used when calculating the vertical flow component.

Value

Circular area-sink analytic element which is an object of class `areasink` and inherits from `element`.

See Also

`headareasink()`

Examples

```
as <- areasink(xc = -500, yc = 0, N = 0.001, R = 500)

# flux assuming a constant head difference over a confining unit
dh <- 3
res <- 10 / 0.0001
as <- areasink(xc = -500, yc = 0, N = -dh/res, R = 500, location = 'base')
```

`capzone` *Calculate the capture zone of a well element*

Description

`capzone()` determines the capture zone of a well element in the flow field by performing backward particle tracking until the requested time is reached.

Usage

```
capzone(aem, well, time, npar = 15, dt = time/10, zstart = aem$base, ...)
```

Arguments

`aem`	`aem` object.
`well`	analytic element of class `well`.
`time`	numeric, time of the capture zone.
`npar`	integer, number of particles to use in the backward tracking. Defaults to 15.
`dt`	numeric, time step length used in the particle tracking. Defaults `time / 10`.
`zstart`	numeric value with the starting elevation of the particles. Defaults to the base of the aquifer.
`...`	additional arguments passed to `tracelines()`.

Details

`capzone()` is a thin wrapper around `tracelines()`. Backward particle tracking is performed using `tracelines()` and setting `forward = FALSE`. Initial particle locations are computed by equally spacing `npar` locations at the well radius at the `zstart` elevation. To obtain a sharper delineation of the capture zone envelope, try using more particles or decreasing `dt`.

Note that different `zstart` values only have an effect in models with vertical flow components.

Value

`capzone()` returns an object of class `tracelines`.

See Also

`tracelines()`

Examples

```
# A model with vertical flow components
k <- 10
top <- 10; base <- 0
n <- 0.3

uf <- uniformflow(TR = 100, gradient = 0.001, angle = -10)
rf <- constant(TR, xc = -1000, yc = 0, hc = 20)
w1 <- well(200, 50, Q = 250)
w2 <- well(-200, -100, Q = 450)
as <- areasink(0, 0, N = 0.001, R = 1500)

m <- aem(k, top, base, n = n, uf, rf, w1, w2, as)

# 5-year capture zone at two different starting levels
# here, the number of particles are set to small values to speed up the examples
# increase the number of particles to obtain a sharper delineation of the envelope
cp5a <- capzone(m, w1, time = 5 * 365, zstart = base, npar = 6, dt = 365 / 4)
cp5b <- capzone(m, w1, time = 5 * 365, zstart = 8, npar = 6, dt = 365 / 4)

xg <- seq(-800, 800, length = 100)
yg <- seq(-500, 500, length = 100)
contours(m, xg, yg, col = 'dodgerblue', nlevels = 20)
plot(cp5a, add = TRUE)
plot(cp5b, add = TRUE, col = 'forestgreen') # smaller zone

# plot the convex hull of the endpoints as a polygon
endp <- endpoints(cp5b)
hull <- chull(endp[, c('x', 'y')])
polygon(endp[hull, c('x', 'y')], col = adjustcolor('forestgreen', alpha.f = 0.7))
```

`constant` *Create a constant-head analytic element*

Description

`constant()` creates an analytic element containing a constant head, often referred to as *the reference point*.

Usage

```
constant(xc, yc, hc, ...)
```

Arguments

`xc`	numeric, x location of the reference point.
`yc`	numeric, y location of the reference point.
`hc`	numeric, hydraulic head at the reference point.
`...`	ignored

Value

Constant-head analytic element point which is an object of class `constant` and inherits from `element`.

Examples

```
rf <- constant(xc = -100, yc = 0, hc = 10)
```

`contours` *Plot contours of a state-variable of the analytic element model*

Description

`contours()` creates a contour plot of a state-variable computed by the analytic element model `aem`, or adds the contour lines to an existing plot.

Usage

```
contours(
  aem,
  x,
  y,
  variable = c("heads", "streamfunction", "potential"),
  asp = 1,
  ...
)
```

Arguments

`aem`	`aem` object.
`x`	numeric, vector or marginal x coordinates at which the gridded values are computed. These must be in ascending order.
`y`	numeric, vector or marginal y coordinates at which the gridded values are computed. These must be in ascending order.
`variable`	character indicating which state-variable to plot. Possible values are `heads` (default), `streamfunction` and `potential`.
`asp`	the `y/x` aspect ratio, see `plot.window()`. Defaults to 1 (equal unit lengths).
`...`	additional arguments passed to `contour()`.

Details

`contours()` is a wrapper around `contour()`. It obtains the values of `variable` at the grid points defined by marginal vectors `x` and `y` and constructs the matrix supplied to `contour()` by reversing the rows and transposing the matrix (see also the documentation of `image()`).

Value

A contour plot of the selected variable.

See Also

`aem()` `contour()` `image()` `heads()`

Examples

```
w <- well(xw = 50, yw = 0, Q = 200)
wi <- well(xw = -200, yw = 0, Q = -100)
uf <- uniformflow(gradient = 0.002, angle = -45, TR = 100)
rf <- constant(-1000, 0, hc = 10)
ml <- aem(k = 10, top = 10, base = 0, n = 0.2, w, wi, uf, rf)

# grid points
xg <- seq(-350, 200, length = 100)
yg <- seq(-125, 125, length = 100)

contours(ml, xg, yg, nlevels = 20, col = 'dodgerblue', labcex = 1)
contours(ml, xg, yg, 'streamfunction', nlevels = 20, col = 'orange',
         drawlabels = FALSE, add = TRUE)

# Not to be confused by contour()
try(
contour(ml, xg, yg, nlevels = 20, col = 'dodgerblue', labcex = 1)
)

# For image() or filled.contour()
library(graphics)
h <- heads(ml, xg, yg, as.grid = TRUE)
h_im <- t(h[dim(h)[1]:1,])
image(xg, yg, h_im, asp = 1)
contour(xg, yg, h_im, asp = 1, add = TRUE) # contours() is a wrapper for this
filled.contour(xg, yg, h_im, asp = 1)
```

`dirflow` *Compute flow in the direction of a given angle*

Description

`dirflow()` computes a flow variable at the given points in the direction of the supplied angle.

Usage

```
dirflow(
  aem,
  x,
  y,
  angle,
  flow = c("discharge", "darcy", "velocity"),
  as.grid = FALSE,
  ...
)
```

Arguments

`aem`	`aem` object.
`x`	numeric x coordinates to evaluate `flow` at.
`y`	numeric y coordinates to evaluate `flow` at.
`angle`	numeric, angle of the direction to evaluate `flow`, in degrees counterclockwise from the x-axis.
`flow`	character specifying which flow variable to use. Possible values are `discharge` (default), `darcy` and `velocity`. See `flow()`.
`as.grid`	logical, should a matrix be returned? Defaults to FALSE. See details.
`...`	additional arguments passed to `discharge()`, `darcy()` or `velocity()`.

Details

The `x` and `y` components of `flow` are used to calculate the directed value using `angle`. The `z` coordinate in `discharge()`, `darcy()` or `velocity()` is set at the aquifer base. Under Dupuit-Forchheimer, the x and y components of the flow vector do not change along the vertical axis.

Value

A vector of `length(x)` (equal to `length(y)`) with the flow values at `x` and `y` in the direction of `angle`. If `as.grid` = `TRUE`, a matrix of dimensions `c(length(y), length(x))` described by marginal vectors `x` and `y` containing the directed flow values at the grid points.

See Also

`flow()`, `flow_through_line()`

Examples

```
rf <- constant(-1000, 0, hc = 10)
uf <- uniformflow(TR = 100, gradient = 0.001, angle = -45)
w <- well(10, -50, Q = 200)

m <- aem(k = 10, top = 10, base = 0, n = 0.2, rf, uf)
dirflow(m, x = c(0, 100), y = 50, angle = -45)

m <- aem(k = 10, top = 10, base = 0, n = 0.2, rf, uf, w, type = 'confined')
dirflow(m, x = c(0, 50, 100), y = c(0, 50), angle = -90,
flow = 'velocity', as.grid = TRUE)
```

element_discharge *Get the computed discharge from an element*

Description

`element_discharge()` obtains the computed discharge into or out of the aquifer for a individual analytic element or all elements of a given type.

Usage

```
element_discharge(aem, name = NULL, type = NULL, ...)
```

Arguments

`aem`	`aem` object.
`name`	character vector with the name of the element(s) as available in `aem$elements`.
`type`	character with the type (class) of element to obtain the summed discharge from. See details.
`...`	ignored

Details

Either `name` or `type` should be specified. If `type` is specified, only one type is allowed. Possible values are `'headwell'`, `'well'`, `'linesink'`, `'headlinesink'`, `'areasink'` or `'headareasink'`.

Only elements that add or remove water from the aquifer will return a non-zero discharge value.

Value

A numeric named vector of length `length(name)` with the discharge into (negative) or out of (positive) the aquifer. If `type` is specified, a single named numeric value with the total discharge into (negative) or out of (positive) the aquifer which is the sum of all individual elements of class `type`.

Examples

```
k <- 10
top <- 10
base <- 0
n <- 0.2
TR <- k * (top - base)

rf <- constant(xc = -500, yc = 0, h = 20)
uf <- uniformflow(gradient = 0.002, angle = -45, TR = TR)
w1 <- well(xw = 50, yw = 0, Q = 200)
w2 <- well(xw = 0, yw = 100, Q = 400)
hw <- headwell(xw = -100, yw = 0, hc = 7.5)
hls <- headlinesink(x0 = -200, y0 = -150, x1 = 200, y1 = 150, hc = 8)
as <- areasink(xc = 0, yc = 0, N = 0.0005, R = 500)
```

```
m <- aem(k, top, base, n, rf, uf, w1, w2, hw, hls, as)

element_discharge(m, name = c('hls', 'as'))
element_discharge(m, type = 'well')

# zero discharge for uniform flow element as it does not add or remove water
element_discharge(m, name = 'uf')
```

`flow` *Calculate flow variables*

Description

`discharge()` computes the `x, y` and `z` components of the discharge vector for an `aem` object at the given x, y and z coordinates.

`darcy()` computes the `x, y` and `z` components of the Darcy flux vector (also called *specific discharge vector*) for an `aem` object at the given x, y and z coordinates.

`velocity()` computes the `x, y` and `z` components of the average linear groundwater flow velocity vector for an `aem` object at the given x, y and z coordinates.

`domega()` computes the complex discharge for an `aem` or `element` object at the given x and y coordinates.

Usage

```
discharge(...)

darcy(...)

velocity(...)

domega(...)

## S3 method for class 'aem'
discharge(
  aem,
  x,
  y,
  z,
  as.grid = FALSE,
  magnitude = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'aem'
darcy(aem, x, y, z, as.grid = FALSE, magnitude = FALSE, ...)
```

```
## S3 method for class 'aem'
velocity(aem, x, y, z, as.grid = FALSE, magnitude = FALSE, R = 1, ...)

## S3 method for class 'aem'
domega(aem, x, y, as.grid = FALSE, ...)

## S3 method for class 'element'
domega(element, x, y, ...)
```

Arguments

`...`	ignored or arguments passed from `velocity()` or `darcy()` to `discharge()`.
`aem`	`aem` object.
`x`	numeric x coordinates to evaluate the flow at.
`y`	numeric y coordinates to evaluate the flow at.
`z`	numeric z coordinates to evaluate at
`as.grid`	logical, should a matrix be returned? Defaults to `FALSE`. See details.
`magnitude`	logical, should the magnitude of the flow vector be returned as well? Default to `FALSE`. See details.
`verbose`	logical, if `TRUE` (default), warnings with regards to setting `Qz` to `NA` are printed. See details.
`R`	numeric, retardation coefficient used in `velocity()`. Defaults to 1 (no retardation).
`element`	analytic element of class `element`.

Details

There is no `discharge()`, `darcy()` or `velocity()` method for an object of class `element` because an `aem` object is required to obtain the aquifer base and top.

If the `z` coordinate is above the saturated aquifer level (i.e. the water-table for unconfined conditions or the aquifer top for confined conditions), or below the aquifer base, `Qz` values are set to `NA` with a warning (if `verbose = TRUE`). The `Qx` and `Qy` values are not set to `NA`, for convenience in specifying the `z` coordinate when only lateral flow is of interest.

Value

For `discharge()`, a matrix with the number of rows equal to the number of points to evaluate the discharge vector at, and with columns `Qx`, `Qy` and `Qz` corresponding to `x, y` and `z` components of the discharge vector at coordinates `x`, `y` and `z`. If `as.grid = TRUE`, an array of dimensions `c(length(y), length(x), length(z), 3)` described by marginal vectors `x`, `y` and `z` (columns, rows and third dimension) containing the `x, y` and `z` components of the discharge vector (`Qx`, `Qy` and `Qz`) as the fourth dimension.

The `x` component of `discharge()` is the real value of `domega()`, the `y` component the negative imaginary component and the `z` component is calculated based on area-sink strengths and/or the curvature of the phreatic surface.

If `magnitude = TRUE`, the last dimension of the returned array is expanded to include the magnitude of the discharge/Darcy/velocity vector, calculated as `sqrt(Qx^2 + Qy^2 + Qz^2)` (or `sqrt(qx^2 + qy^2 + qz^2)` or `sqrt(vx^2 + vy^2 + vz^2)`, respectively).

For `darcy()`, the same as for `discharge()` but with the `x`, `y` and `z` components of the Darcy flux vector (`qx`, `qy` and `qz`). The values are computed by dividing the values of `discharge()` by the saturated thickness at `x`, `y` and `z`.

For `velocity()`, the same as for `discharge()` but with the `x`, `y` and `z` components of the average linear groundwater flow velocity vector (`vx`, `vy` and `vz`). The values are computed by dividing the `darcy()` values by the effective porosity (`aem$n`) and the retardation coefficient `R`.

For `domega()`, a vector of `length(x)` (equal to `length(y)`) with the complex discharge values at `x` and `y`, If `as.grid = TRUE`, a matrix of dimensions `c(length(y), length(x))` described by marginal vectors `x` and `y` containing the complex discharge values at the grid points. `domega()` is the derivative of `omega()` in the x and y directions.

See Also

`state-variables()`, `satthick()`, `dirflow()`, `flow_through_line()`

Examples

```
w <- well(xw = 55, yw = 0, Q = 200)
uf <- uniformflow(gradient = 0.002, angle = -45, TR = 100)
as <- areasink(xc = 0, yc = 0, N = 0.001, R = 500)
rf <- constant(xc = -1000, yc = 1000, hc = 10)
ml <- aem(k = 10, top = 10, base = -15, n = 0.2, w, uf, as, rf)

xg <- seq(-100, 100, length = 5)
yg <- seq(-75, 75, length = 3)

# Discharge vector
discharge(ml, c(150, 0), c(80, -80), z = -10)
discharge(ml, c(150, 0), c(80, -80), z = c(2, 5), magnitude = TRUE)
discharge(ml, xg, yg, z = 2, as.grid = TRUE)
discharge(ml, c(150, 0), c(80, -80), z = ml$top + c(-5, 0.5)) # NA for z > water-table

# Darcy flux
darcy(ml, c(150, 0), c(80, -80), c(0, 5), magnitude = TRUE)

# Velocity
velocity(ml, c(150, 0), c(80, -80), c(0, 5), magnitude = TRUE, R = 5)

# Complex discharge
domega(ml, c(150, 0), c(80, -80))

# Complex discharge for elements
domega(w, c(150, 0), c(80, -80))
```

`flow_through_line` *Calculate the total flow passing through a line*

Description

`flow_through_line()` computes the integrated flow passing through a straight line at a right angle.

Usage

```
flow_through_line(
  aem,
  x0,
  y0,
  x1,
  y1,
  flow = c("discharge", "darcy"),
  split = FALSE,
  ...
)
```

Arguments

`aem`	`aem` object
`x0`	numeric, starting x location of line.
`y0`	numeric, starting y location of line.
`x1`	numeric, ending x location of line.
`y1`	numeric, ending y location of line.
`flow`	character specifying which flow variable to use. Possible values are `discharge` (default) and `darcy`. See `flow()`.
`split`	logical, should the flow be split up into positive and negative flows (`TRUE`) or should they be summed (`FALSE`; default)? See details.
`...`	ignored

Details

The flow is computed normal to the line and integrated along the line length using `stats::integrate()`. The flow value is positive going to the left when looking in the direction of the line (i.e. to the left going from `x0`-`y0` to `x1`-`y1`).

If `split = FALSE` (the default), a single value is returned which is the sum of the positive and negative flows perpendicular to the line. If `split = TRUE`, both the positive and negative component of the total flow through the line are returned.

If the line corresponds to a line element, the integration might fail. Try to perturbate the line vertices slightly in that case.

Value

If `split` = `FALSE`, a single value with the total flow of variable `flow` passing through the line at a right angle. If `split` = `TRUE` a named vector with the total positive and total negative value of `flow` passing through the line.

See Also

`flow()`, `dirflow()`

Examples

```
rf <- constant(-1000, 0, hc = 10)
uf <- uniformflow(TR = 100, gradient = 0.001, angle = -45)
m <- aem(k = 10, top = 10, base = 0, n = 0.2, rf, uf)

xg <- seq(-500, 500, l=100); yg <- seq(-300, 300, l=100)
contours(m, xg, yg, col='dodgerblue', nlevels=20)

x0 <- -200
y0 <- -50
x1 <- 300
y1 <- 100
lines(matrix(c(x0, y0, x1, y1), ncol = 2, byrow = TRUE))

flow_through_line(m, x0, y0, x1, y1)
flow_through_line(m, x1, y1, x0, y0) # reverse direction of line

w <- well(125, 200, 150)
m <- aem(k = 10, top = 10, base = 0, n = 0.2, rf, uf, w)
contours(m, xg, yg, col='dodgerblue', nlevels=20)
lines(matrix(c(x0, y0, x1, y1), ncol = 2, byrow = TRUE))

flow_through_line(m, x0, y0, x1, y1, flow = 'darcy')
flow_through_line(m, x0, y0, x1, y1, flow = 'darcy', split = TRUE)
```

`headareasink` *Create a head-specified area-sink analytic element*

Description

`headareasink()` creates a circular area-sink analytic element with constant specified head. The constant leakage flux into or out of the aquifer from the area-sink is computed by solving the corresponding `aem` model.

Usage

```
headareasink(xc, yc, hc, R, resistance = 0, location = c("top", "base"), ...)
```

Arguments

`xc`	numeric, x location of the center of the area-sink.
`yc`	numeric, y location of the center of the area-sink.
`hc`	numeric, specified hydraulic head at the center of the area-sink.
`R`	numeric, radius of the circular area-sink.
`resistance`	numeric, hydraulic resistance of the area-sink at its connection with the aquifer. Defaults to 0 (no resistance).
`location`	character, either `top` (default) or `base` specifying the vertical position of the area-sink.
`...`	ignored

Details

The constant leakage flux from the area-sink is computed by solving the `aem` model given the specified head `hc` for the area-sink. This head is located at the so-called collocation point, which is placed at the center of the area-sink. A positive flux is into the aquifer. Note that this head-dependent flux is constant over the domain and computed only at the collocation point. The flux is therefore determined by the difference in aquifer head and specified head at that location only, and does not vary across the domain with varying aquifer head.

The resistance can be increased for a area-sink in poor connection with the aquifer, e.g. because of a confining unit of low hydraulic conductivity between the aquifer and the area-sink. If the aquifer is unconfined (i.e. has a variable saturated thickness), the system of equations will then become non-linear with respect to the hydraulic head and iteration is required to solve the model.

Value

Circular head-specified area-sink analytic element which is an object of class `headareasink` and inherits from `areasink`.

See Also

`areasink()`

Examples

```
has <- headareasink(xc = -500, yc = 0, hc = 3, R = 500, res = 1000)
has <- headareasink(xc = -500, yc = 0, hc = 3, R = 500, location = 'base')
```

headlinesink *Create a head-specified line-sink analytic element*

Description

`headlinesink()` creates a line-sink analytic element with constant specified head. The discharge into the line-sink per unit length is computed by solving the corresponding `aem` model.

Usage

```
headlinesink(x0, y0, x1, y1, hc, resistance = 0, width = 0, ...)
```

Arguments

`x0`	numeric, starting x location of line-sink.
`y0`	numeric, starting y location of line-sink.
`x1`	numeric, ending x location of line-sink.
`y1`	numeric, ending y location of line-sink.
`hc`	numeric, specified hydraulic head of the line-sink.
`resistance`	numeric, hydraulic resistance of the line-sink at its connection with the aquifer. Defaults to 0 (no resistance).
`width`	numeric, width of the line-sink. Used with `resistance` to calculate the line-sink strength, and by `tracelines()` to determine if a particle has reached the line. Defaults to zero (infinitesimally narrow line).
`...`	ignored

Details

The strength of the line-sink (discharge per unit length of line-sink) is computed by solving the `aem` model given the specified head `hc` for the line-sink. This head is located at the so-called collocation point, which is placed at the center of the line-sink.

The resistance can be increased for a line-sink in poor connection with the aquifer. The effect of a larger or smaller wetted perimeter can be mimicked by adjusting the `resistance` and/or `width` accordingly. If `width = 0` (the default) it is removed from the conductance calculation. If the aquifer is unconfined (i.e. has a variable saturated thickness), the system of equations becomes non-linear with respect to the hydraulic head and iteration is required to solve the model.

Value

Head-specified line-sink analytic element which is an object of class `headlinesink` and inherits from `linesink`.

See Also

`linesink()`

Examples

```
hls <- headlinesink(-75, 50, 100, 50, hc = 10)
hls <- headlinesink(-75, 50, 100, 50, hc = 10, resistance = 10, width = 4)
```

---

headwell — *Create a analytic element of a well with a constant head*

Description

`headwell()` creates an analytic element of a well with a constant, specified head. The discharge into the well is computed by solving the corresponding `aem` model. The head can be specified at the well or at any other location.

Usage

```
headwell(xw, yw, hc, rw = 0.3, xc = xw, yc = yw, rc = rw, resistance = 0, ...)
```

Arguments

`xw`	numeric, x location of the well.
`yw`	numeric, y location of the well.
`hc`	numeric, specified hydraulic head at the collocation point.
`rw`	numeric, radius of the well. Defaults to 0.3 (meter).
`xc`	numeric, x location of the collocation point. See details. Defaults to `xw`.
`yc`	numeric, y location of the collocation point. See details. Defaults to `yw`.
`rc`	numeric, radius of the collocation point. See details. Defaults to `rw`.
`resistance`	numeric, hydraulic resistance at the collocation point. Defaults to 0 (no resistance).
`...`	ignored

Details

The discharge from the well at location `xw - yw` is computed by solving the `aem` model given the specified head `hc`. This head is specified at `xc + rc - yc`, called the collocation point. This can be used to compute the discharge of the well by specifying the head at some other location. By default, the location of the well and the collocation point are the same.

The hydraulic resistance of the well screen at the collocation point can be increased for a well in poor connection with the aquifer. If the aquifer is unconfined (i.e. has a variable saturated thickness), the system of equations becomes non-linear with respect to the hydraulic head and iteration is required to solve the model.

Value

Analytic element of a well with constant head which is an object of class `headwell` and inherits from `well`.

See Also

`well()`

Examples

```
hw <- headwell(xw = 400, yw = 300, hc = 20, rw = 0.3)
hw <- headwell(xw = 400, yw = 300, hc = 20, rw = 0.3, resistance = 10)
hw <- headwell(xw = 400, yw = 300, hc = 20, rw = 0.3, xc = 500, yc = 500, rc = 0)
```

`head_to_potential` *Convert hydraulic head to potential and vice versa*

Description

`head_to_potential()` calculates the discharge potential from the hydraulic head.

`potential_to_head()` calculates the hydraulic head from the discharge potential.

Usage

```
head_to_potential(aem, h, ...)

potential_to_head(aem, phi, na.below = TRUE, ...)
```

Arguments

`aem`	`aem` object.
`h`	numeric hydraulic head values as vector or matrix.
`...`	ignored
`phi`	numeric discharge potential values as vector or matrix.
`na.below`	logical indicating if calculated head values below the aquifer base should be set to `NA`. Defaults to `TRUE`. See details.

Value

`head_to_potential()` returns the discharge potentials calculated from `h`, in the same structure as `h`.

`potential_to_head()` returns the hydraulic heads calculated from `phi`, in the same structure as `phi`.

The conversion of potential to head or vice versa is different for confined (constant saturated thickness) and unconfined (variable saturated thickness) aquifers as set by the `type` argument in `aem()`.

If `na.below = FALSE`, negative potentials can be converted to hydraulic heads if flow is unconfined (`aem$type = 'variable'`). The resulting heads are below the aquifer base. This may be useful for some use cases, e.g. in preliminary model construction or for internal functions. In most cases however, these values should be set to `NA` (the default behavior) since other analytic elements will continue to extract or inject water even though the saturated thickness of the aquifer is negative, which is not realistic. In those cases, setting `aem$type = 'confined'` might prove useful. Also note that these heads below the aquifer base will not be correctly re-converted to potentials using `head_to_potential()`. As such, caution should be taken when setting `na.below = FALSE`.

Examples

```
k <- 10
top <- 10; base <- 0
uf <- uniformflow(TR = 100, gradient = 0.001, angle = -45)
rf <- constant(TR, xc = -1000, yc = 0, hc = 10)
w1 <- well(200, 50, Q = 250)
m <- aem(k, top, base, n = 0.2, uf, rf, w1, type = 'variable') # variable saturated thickness
mc <- aem(k, top, base, n = 0.2, uf, rf, w1, type = 'confined') # constant saturated thickness
xg <- seq(-500, 500, length = 100)
yg <- seq(-250, 250, length = 100)

h <- heads(m, x = xg, y = yg, as.grid = TRUE)
hc <- heads(mc, x = xg, y = yg, as.grid = TRUE)
pot <- head_to_potential(m, h)
potc <- head_to_potential(mc, hc)

phi <- potential(m, x = xg, y = yg, as.grid = TRUE)
phic <- potential(mc, x = xg, y = yg, as.grid = TRUE)
hds <- potential_to_head(m, phi)
hdsc <- potential_to_head(mc, phic)

# Converting negative potentials results in NA's with warning
try(
potential_to_head(m, -300)
)

# unless na.below = FALSE
potential_to_head(m, -300, na.below = FALSE)
```

`linesink` *Create a strength-specified line-sink analytic element*

Description

`linesink()` creates a line-sink analytic element with constant specified strength.

Usage

```
linesink(x0, y0, x1, y1, sigma, width = 0, ...)
```

Arguments

`x0`	numeric, starting x location of line-sink.
`y0`	numeric, starting y location of line-sink.
`x1`	numeric, ending x location of line-sink.
`y1`	numeric, ending y location of line-sink.
`sigma`	numeric, specific strength of the line-sink, i.e. discharge per unit length of line-sink. Positive is out of aquifer.
`width`	numeric, width of the line-sink. Only used in `tracelines()` to determine if a particle has reached the line. Defaults to zero (infinitesimally narrow line).
`...`	ignored

Value

Strength-specified line-sink analytic element which is an object of class `linesink` and inherits from `element`.

See Also

`headlinesink()`

Examples

```
ls <- linesink(-75, 50, 100, 50, sigma = 1, width = 3)
```

`satthick` *Compute the saturated thickness*

Description

`satthick()` computes the saturated thickness of the aquifer from an `aem` object at the given x and y coordinates.

Usage

```
satthick(aem, x, y, as.grid = FALSE, ...)
```

Arguments

`aem`	`aem` object.
`x`	numeric x coordinates to evaluate at.
`y`	numeric y coordinates to evaluate at.
`as.grid`	logical, should a matrix be returned? Defaults to `FALSE`. See details.
`...`	additional arguments passed to `heads()` when `aem$type = 'variable'`.

Details

If the aquifer is confined at x and y, the saturated thickness equals the aquifer thickness. For flow with variable saturated thickness (`aem$type = 'variable'`), if the aquifer is unconfined at x and y, the saturated thickness is calculated as the hydraulic head at x and y minus the aquifer base.

Value

A vector of `length(x)` (equal to `length(y)`) with the saturated thicknesses at x and y. If `as.grid = TRUE`, a matrix of dimensions `c(length(y), length(x))` described by marginal vectors x and y containing the saturated thicknesses at the grid points.

See Also

`flow()`, `state-variables()`

Examples

```
uf <- uniformflow(100, 0.001, 0)
rf <- constant(-1000, 0, 11)
m <- aem(k = 10, top = 10, base = 0, n = 0.2, uf, rf, type = 'confined')

satthick(m, x = c(-200, 0, 200), y = 0) # confined
s <- satthick(m, x = seq(-500, 500, length = 100),
              y = seq(-250, 250, length = 100), as.grid = TRUE)
str(s)

mv <- aem(k = 10, top = 10, base = 0, n = 0.2, uf, rf, type = 'variable')
satthick(mv, x = c(-200, 0, 200), y = 0) # variable
```

`state-variables` *Calculate state-variables*

Description

`heads()` computes the hydraulic head at the given x and y coordinates for an `aem` object.

`omega()` computes the complex potential for an `aem` or `element` object at the given x and y coordinates.

`potential()` computes the discharge potential for an `aem` or `element` object at the given x and y coordinates.

`streamfunction()` computes the stream function for an `aem` or `element` object at the given x and y coordinates.

Usage

```
heads(aem, x, y, as.grid = FALSE, na.below = TRUE, ...)

omega(...)

potential(...)

streamfunction(...)

## S3 method for class 'aem'
omega(aem, x, y, as.grid = FALSE, ...)

## S3 method for class 'aem'
potential(aem, x, y, as.grid = FALSE, ...)

## S3 method for class 'aem'
streamfunction(aem, x, y, as.grid = FALSE, ...)

## S3 method for class 'element'
omega(element, x, y, ...)

## S3 method for class 'element'
potential(element, x, y, ...)

## S3 method for class 'element'
streamfunction(element, x, y, ...)
```

Arguments

`aem`	`aem` object.
`x`	numeric x coordinates to evaluate the variable at.
`y`	numeric y coordinates to evaluate the variable at.
`as.grid`	logical, should a matrix be returned? Defaults to `FALSE`. See details.
`na.below`	logical indicating if calculated head values below the aquifer base should be set to `NA`. Defaults to `TRUE`. See `potential_to_head()`.
`...`	ignored
`element`	analytic element of class `element`.

Details

`heads()` should not to be confused with `utils::head()`, which returns the first part of an object.

Value

For `heads()`, a vector of `length(x)` (equal to `length(y)`) with the hydraulic head values at `x` and `y`. If `as.grid = TRUE`, a matrix of dimensions `c(length(y), length(x))` described by marginal vectors x and y containing the hydraulic head values at the grid points. The heads are computed from potential() and the aquifer parameters using potential_to_head().

For omega(), the same as for heads() but containing the complex potential values evaluated at x and y.

For potential(), the same as for heads() but containing the discharge potential values evaluated at x and y, which are the real components of omega().

For streamfunction(), the same as for heads() but containing the stream function values evaluated at x and y, which are the imaginary components of omega().

See Also

flow(), satthick(), head_to_potential()

Examples

```
w <- well(xw = 55, yw = 0, Q = 200)
uf <- uniformflow(gradient = 0.002, angle = -45, TR = 100)
rf <- constant(xc = -1000, yc = 1000, hc = 10)
ml <- aem(k = 10, top = 10, base = -15, n = 0.2, w, uf, rf)

xg <- seq(-100, 100, length = 5)
yg <- seq(-75, 75, length = 3)

# Hydraulic heads
heads(ml, c(50, 0), c(25, -25))
heads(ml, xg, yg, as.grid = TRUE)

# do not confuse heads() with utils::head, which will give an error
try(
head(ml, c(50, 0), c(25, -25))
)

# Complex potential
omega(ml, c(50, 0), c(25, -25))

# Discharge potential
potential(ml, c(50, 0), c(25, -25))

# Stream function
streamfunction(ml, c(50, 0), c(25, -25))

# For elements
omega(w, c(50, 0), c(-25, 25))

potential(w, c(50, 0), c(-25, 25))

streamfunction(w, c(50, 0), c(-25, 25))
```

tracelines — *Compute tracelines of particles*

Description

tracelines() tracks particle locations moving forward or backward with the advective groundwater flow by numerically integrating the velocity vector. The resulting set of connected coordinates produces the tracelines.

endpoints() obtains the final time and locations of tracked particles.

Usage

```
tracelines(
  aem,
  x0,
  y0,
  z0,
  times,
  forward = TRUE,
  R = 1,
  tfunc = NULL,
  tol = 0.001,
  ncores = 0,
  ...
)

endpoints(tracelines, ...)

## S3 method for class 'tracelines'
plot(x, y = NULL, add = FALSE, type = "l", arrows = FALSE, marker = NULL, ...)
```

Arguments

aem	aem object.
x0	numeric vector with starting x locations of the particles.
y0	numeric vector with starting y locations of the particles.
z0	numeric vector with starting z locations of the particles.
times	numeric vector with the times at which particle locations should be registered.
forward	logical, should forward (TRUE; default) or backward (FALSE) tracking be performed.
R	numeric, retardation coefficient passed to velocity(). Defaults to 1 (no retardation).
tfunc	function or list of functions with additional termination events for particles. See details. Defaults to NULL.

`tol`	numeric tolerance used to define when particles have crossed a line element. Defaults to 0.001 length units.
`ncores`	integer, number of cores to use when running in parallel. Defaults to 0 (no parallel computing). See details.
`...`	additional arguments passed to `plot()` or `arrows()` when plotting. Otherwise ignored.
`tracelines`	object of class `tracelines` as returned by `tracelines()`.
`x`	object of class `tracelines`.
`y`	ignored
`add`	logical, should the plot be added to the existing plot? Defaults to `FALSE`.
`type`	character indicating what type of plot to draw. See `plot()`. Defaults to `'l'` (lines).
`arrows`	logical indicating if arrows should be drawn using `arrows()`. Defaults to `FALSE`.
`marker`	numeric, time interval at which to plot point markers. Defaults to `NULL` (no markers). See details.

Details

`deSolve::lsoda()` is used to numerically integrate the velocity vector.

Particles are terminated prematurely when they have reached the inner annulus of well elements, when they have crossed a line element (or enter half its non-zero width on either side) or when they travel above the saturated aquifer level (i.e. the water-table for unconfined conditions or the aquifer top for confined conditions), or below the aquifer base. Note that these last two conditions can only occur in models with vertical flow components. The returned time value is the time of termination.

The `tfunc` argument can be used to specify additional termination events. It is a function (or a list of functions) that takes arguments `t`, `coords` and `parms`. These are, respectively, a numeric value with the current tracking time, a numeric vector of length 3 with the current `x`, `y` and `z` coordinate of the particle, and a list with elements `aem` and `R` (named as such). It should return a single logical value indicating if the particle should terminate. See examples.

If initial particle locations are above the saturated aquifer level, they are reset to this elevation with a warning. Initial particle locations below the aquifer base are reset at the aquifer base with a warning. A small perturbation is added to these elevations to avoid the particle tracking algorithm to get stuck at these locations. If the algorithm does get stuck (i.e. excessive run-times), try resetting the `z0` values to elevations well inside the saturated domain.

Initial particle locations inside a termination point are dropped with a warning.

Backward particle tracking is performed by reversing the flow field (i.e. multiplying the velocities with `-1`).

Traceline computation is embarrassingly parallel. When `ncores` > `0`, the `parallel` package is used to set up the cluster with the requested nodes and the tracelines are computed using `parallel::parLapplyLB()`. `ncores` should not exceed the number of available cores as returned by `parallel::detectCores()`.

Plotting:

The `marker` value can be used to plot point markers at given time intervals, e.g. every 365 days (see examples). The x and y locations of each particle at the marked times are obtained by linearly interpolating from the computed particle locations.

Value

`tracelines()` returns an object of class `tracelines` which is a list with length equal to the number of particles where each list element contains a matrix with columns `time`, `x`, `y` and `z` specifying the registered time and coordinates of the particle as it is tracked through the flow field.

The final row represents either the location at the maximum `times` value or, if the particle terminated prematurely, the time and location of the termination.

The matrices are ordered in increasing time. By connecting the coordinates, the tracelines can be produced.

`endpoints()` returns a matrix with columns `time`, `x`, `y` and `z` specifying the final time and coordinates of the particles in the `tracelines` object.

See Also

`capzone()`

Examples

```
# create a model with uniform background flow
k <- 10
top <- 10; base <- 0
n <- 0.2
R <- 5
hc <- 20

uf <- uniformflow(TR = 100, gradient = 0.001, angle = -10)
rf <- constant(TR, xc = -1000, yc = 0, hc = hc)

m <- aem(k, top, base, n = n, uf, rf)

# calculate forward particle traces
x0 <- -200; y0 <- seq(-200, 200, 200)
times <- seq(0, 25 * 365, 365 / 4)
paths <- tracelines(m, x0 = x0, y0 = y0, z = top, times = times)
endp <- endpoints(paths)

xg <- seq(-500, 500, length = 100)
yg <- seq(-300, 300, length = 100)

# plot
contours(m, xg, yg, col = 'dodgerblue', nlevels = 20)
plot(paths, add = TRUE, col = 'orange')
points(endp[, c('x', 'y')])

# Backward tracking with retardation; plot point marker every 5 years
paths_back <- tracelines(m, x0 = x0, y0 = y0, z0 = top, times = times, R = R, forward = FALSE)
plot(paths_back, add = TRUE, col = 'forestgreen', marker = 5*365, cex = 0.5)

# -------
# Termination at wells, line-sinks and user-defined zone
w1 <- well(200, 50, Q = 250)
```

```
w2 <- well(-200, -100, Q = 450)
ls <- headlinesink(x0 = -100, y0 = 100, x1 = 400, y1 = -300, hc = 7)

m <- aem(k, top, base, n = n, uf, rf, w1, w2, ls)

# User-defined termination in rectangular zone
tzone <- cbind(x = c(-300, -200, -200, -300), y = c(150, 150, 100, 100))
termf <- function(t, coords, parms) {
  x <- coords[1]
  y <- coords[2]
  in_poly <- x <= max(tzone[,'x']) & x >= min(tzone[,'x']) &
    y <= max(tzone[,'y']) & y >= min(tzone[,'y'])
  return(in_poly)
}

x0 <- c(-300, -200, 0, 200, 300)
y0 <- 200
times <- seq(0, 5 * 365, 365 / 15)
paths <- tracelines(m, x0 = x0, y0 = y0, z0 = top, times = times, tfunc = termf)

contours(m, xg, yg, col = 'dodgerblue', nlevels = 20)
plot(m, add = TRUE)
polygon(tzone)
plot(paths, add = TRUE, col = 'orange')

# -------
# model with vertical flow due to area-sink
as <- areasink(xc = 0, yc = 0, N = 0.001, R = 1500)
m <- aem(k, top, base, n = n, uf, rf, w1, w2, as)

# starting z0 locations are above aquifer top and will be reset to top with warning
x0 <- seq(-400, 200, 200); y0 <- 200
times <- seq(0, 5 * 365, 365 / 4)
paths <- tracelines(m, x0 = x0, y0 = y0, z0 = top + 0.5, times = times)

contours(m, xg, yg, col = 'dodgerblue', nlevels = 20)
plot(m, add = TRUE)
plot(paths, add = TRUE, col = 'orange')

# -------
# plot vertical cross-section of traceline 4 along increasing y-axis (from south to north)
plot(paths[[4]][,c('y', 'z')], type = 'l')

# -------
# parallel computing by setting ncores > 0
mp <- aem(k, top, base, n = n, uf, rf)
pathsp <- tracelines(mp, x0 = x0, y0 = y0, z = top, times = times, ncores = 2)

# -------
# plot arrows
contours(m, xg, yg, col = 'dodgerblue', nlevels = 20)
plot(paths, add = TRUE, col = 'orange', arrows = TRUE, length = 0.05)
```

```
# plot point markers every 2.5 years
contours(m, xg, yg, col = 'dodgerblue', nlevels = 20)
plot(paths, add = TRUE, col = 'orange', marker = 2.5 * 365, pch = 20)

# plot point markers every 600 days
plot(paths, add = TRUE, col = 'forestgreen', marker = 600, pch = 1)
```

`uniformflow` *Create an analytic element with uniform flow*

Description

`uniformflow()` creates an analytic element of constant uniform background flow.

Usage

```
uniformflow(TR, gradient, angle, ...)
```

Arguments

`TR`	numeric, constant transmissivity value used to define the discharge.
`gradient`	numeric, hydraulic gradient. Positive in the direction of flow.
`angle`	numeric, angle of the primary direction of background flow in degrees counter-clockwise from the x-axis.
`...`	ignored

Details

`TR` and `gradient` are multiplied to obtain the discharge which remains constant throughout the system, independent of the saturated thickness of the aquifer.

Groundwater flow is always in the direction of the *negative* hydraulic gradient. Note that `gradient` is specified here as positive in the direction of flow for convenience.

Value

Analytic element of constant uniform flow which is an object of class `uniformflow` and inherits from `element`.

Examples

```
uf <- uniformflow(TR = 100, gradient = 0.002, angle = -45) # South-eastern direction
```

`well` *Create an analytic element of a constant-discharge well*

Description

`well()` creates an analytic element of a well with constant discharge.

Usage

```
well(xw, yw, Q, rw = 0.3, ...)
```

Arguments

`xw`	numeric, x location of the well.
`yw`	numeric, y location of the well.
`Q`	numeric, volumetric discharge of the well (positive is out of aquifer).
`rw`	numeric, radius of well. Defaults to 0.3 length units.
`...`	ignored

Details

The inner annulus of a well element constitutes a singularity in the equations as the hydraulic head is undefined at a distance smaller than `rw` from the well center. If a state- or flow-variable is calculated within this annulus, its location is reset to its nearest location on the well screen.

The well is assumed to fully penetrate the saturated aquifer.

Value

Analytic element of a well with constant discharge which is an object of class `well` and inherits from `element`.

See Also

`headwell()`

Examples

```
w <- well(xw = 50, yw = 0, Q = 200, rw = 0.3)
```

Index