Package 'aRpsDCA'

October 12, 2022

Version 1.1.1

Date 2017-07-23

Title Arps Decline Curve Analysis in R

Description
Functions for Arps decline-curve analysis on oil and gas data. Includes exponential, hyperbolic, harmonic, and hyperbolic-to-exponential models as well as the preceding with initial curtailment or a period of linear rate buildup. Functions included for computing rate, cumulative production, instantaneous decline, EUR, time to economic limit, and performing least-squares best fits.

Imports stats, methods

License LGPL-2.1

URL https://github.com/derrickturk/aRpsDCA

BugReports https://github.com/derrickturk/aRpsDCA/issues

NeedsCompilation no

Author Derrick Turk [aut, cre, cph]

Maintainer Derrick Turk <dwt@terminusdatascience.com>

Repository CRAN

Date/Publication 2017-07-23 19:39:44 UTC

R topics documented:

aRpsDCA-package 2
arps 3
arps.eur 4
arps.with.buildup 5
as.effective 6
as.nominal 7
bestfit 8
curtailed 17
exponential 18
format.arps 19
harmonic 20

hyp2exp 21
hyperbolic 22
print.arps 23
rescale.by.time 24

Index **25**

`aRpsDCA-package` *Arps Decline Curve Analysis in R*

Description

Functions for Arps decline-curve analysis. Includes exponential, hyperbolic, harmonic, and hyperbolic-to-exponential models.

Details

Index:

`arps`	Generic functions for Arps decline curves
`arps.eur`	EUR and time to economic limit for Arps decline curves
`curtail`	Arps decline curves with initial rate curtailment
`as.effective`	Arps decline conversion from nominal to effective
`as.nominal`	Arps decline conversion from effective to nominal
`bestfit`	Best-fitting for Arps decline curves
`exponential`	Arps exponential declines
`harmonic`	Arps harmonic declines
`hyp2exp`	Arps hyperbolic-to-exponential declines
`hyperbolic`	Arps hyperbolic declines
`print.arps`	Print representations of Arps decline curves
`rescale.by.time`	Time unit conversion for DCA

Author(s)

Derrick W. Turk | **terminus data science, LLC** <`<dwt@terminusdatascience.com>`>

References

SPEE REP#6 (`https://secure.spee.org/sites/default/files/wp-files/pdf/ReferencesResources/REP06-DeclineCurves.pdf`)

Examples

```
## Plot semi-log rate-time and Cartesian rate-cumulative
## for a hyperbolic-to-exponential decline
t <- seq(0, 10, 0.5)
q <- hyp2exp.q(5000, as.nominal(0.90), 1.5, as.nominal(0.10), t)
Np <- hyp2exp.Np(5000, as.nominal(0.90), 1.5, as.nominal(0.10), t)
```

```
old.par <- par(ask=TRUE)
plot(log(q) ~ t)
plot(q ~ Np)
par(old.par)
```

---

arps *Arps decline classes and S3 methods*

Description

Create Arps decline curve objects and compute rates, cumulative production, and nominal declines.

Usage

```
arps.decline(qi, Di, b=NA, Df=NA)
## S3 method for class 'arps'
arps.q(decl, t)
## S3 method for class 'arps'
arps.Np(decl, t)
## S3 method for class 'arps'
arps.D(decl, t)
```

Arguments

`qi`	initial rate [volume / time], i.e. q(t = 0).
`Di`	nominal Arps decline exponent [1 / time].
`b`	Arps hyperbolic exponent.
`Df`	nominal Arps decline exponent [1 / time].
`t`	time at which to evaluate rate, cumulative, or nominal decline [time].
`decl`	an Arps decline object as returned by `arps.decline`.

Details

Depending on whether arguments `b` and `Df` are supplied, `arps.decline` will select an exponential, hyperbolic, or hyperbolic-to-exponential decline and return an object appropriately. The returned object will have class `"exponential"`, `"hyperbolic"`, or `"hyp2exp"` in addition to class `"arps"`.

Assumes consistent units of time between `qi`, `Di`, `Df`, and `t`. To convert, see the decline-rate conversion functions referenced below.

Value

`arps.decline` returns an object having class `"arps"`, suitable for use as an argument to S3 methods discussed here.

`q.arps` returns the rate for each element of `t` applying decline `decl`, in the same units as the value of `qi` for `decl`.

`Np.arps` returns the cumulative production for each element of `t` applying decline `decl`, in the same units as the value of `qi * t` for `decl`.

`D.arps` returns the nominal decline for each element of `t` applying decline `decl`, in the same units as the value of `Di` for `decl`.

See Also

`print.arps`, `exponential`, `hyperbolic`, `hyp2exp`, `as.effective`, `as.nominal`, `rescale.by.time`.

Examples

```
## exponential decline with
## qi = 1000 Mscf/d, Di = 95% effective / year
## rate for t from 0 to 25 days
decline <- arps.decline(1000,
    as.nominal(0.95, from.period="year", to.period="day"))
arps.q(decline, seq(0, 25))

## hyperbolic decline with
## qi = 500 bopd, Di = 3.91 nominal / year, b = 1.5,
## cumulative production at t = 5 years
decline <- arps.decline(
    rescale.by.time(500, from="day", to="year", method="rate"),
    3.91, 1.5)
arps.Np(decline, 5)
```

`arps.eur` *EUR and time-to-limit for Arps decline curves*

Description

Evaluate estimated ultimate recovery and time to economic limit for Arps decline curve objects.

Usage

```
arps.eur(decl, q.limit)
arps.t.el(decl, q.limit)
```

Arguments

`decl`	an Arps decline object as returned by `arps.decline` or `curtail`.
`q.limit`	economic limit rate [volume / time] in same units as `decl`.

Value

`arps.eur` returns the total production for `decl` at the point in time when the economic limit `q.limit` is reached; that is, `arps.Np(decl, arps.t.el(decl, q.limit))`, in the same units as `q.limit`.

`arps.t.el` returns the time until the economic limit `q.limit` is reached for decline `decl`.

See Also

`arps`, `curtail`.

Examples

```
## exponential decline with
## qi = 1000 Mscf/d, Di = 95% effective / year
## EUR with economic limit 10 Mscf/d
decline <- arps.decline(1000,
    as.nominal(0.95, from.period="year", to.period="day"))
arps.eur(decline, 10)

## hyperbolic decline with
## qi = 500 bopd, Di = 3.91 nominal / year, b = 1.5,
## time to reach economic limit of 3 bopd
decline <- arps.decline(rescale.by.time(500, from="day", to="year"),
    3.91, 1.5)
arps.t.el(decline, rescale.by.time(3, from="day", to="year"))
```

`arps.with.buildup` *Arps declines with linear buildup period*

Description

Extend Arps decline curve objects by replacing early-time declines with a buildup period in which rate is a linear function of time.

Usage

```
arps.with.buildup(decl, initial.rate, time.to.peak)
```

Arguments

`decl`	an Arps decline object as produced by `arps.decline`.
`initial.rate`	initial rate [volume / time] (at time = 0) for buildup period.
`time.to.peak`	time to peak rate (i.e.~length of buildup period).

Value

`arps.with.buildup` returns an object having class `"arps"`, which may be used as an argument to methods such as `arps.q`, `arps.Np`, `arps.D`, or `print.arps`.

This object implements a decline curve which behaves as `decl` for all time greater than `time.to.peak`, but implements a linear buildup of rate interpolated between `initial.rate` at time zero and `arps.q(decl, time.to.peak)` at `time.to.peak`.

See Also

`arps.decline`

Examples

```
## hyperbolic decline with
## qi = 500 bopd, Di = 3.91 nominal / year, b = 1.5,
## cumulative production at t = 5 years
decline <- arps.decline(
    rescale.by.time(500, from="day", to="year", method="rate"),
    3.91, 1.5)
# add buildup from initial rate of 50 bopd, over 30 days
decline.with.buildup <- arps.with.buildup(decline,
    rescale.by.time(50, from="day", to="year", method="rate"),
    rescale.by.time(30, from="day", to="year", method="time"))
# forecast 5 years and compare
forecast.time <- seq(0, 5, 0.1)
plot(arps.q(decline, forecast.time) ~ forecast.time, log="y", type="l",
    lty="dashed", col="red")
lines(arps.q(decline.with.buildup, forecast.time) ~ forecast.time,
    lty="dotted", col="blue")
```

---

as.effective — *Arps decline conversion from nominal to effective*

Description

Convert nominal to effective Arps decline.

Usage

```
as.effective(D.nom,
    from.period=c("year", "month", "day"),
    to.period=c("year", "month", "day"))
```

Arguments

`D.nom`	nominal Arps decline [1 / time].
`from.period`	time period for `D.nom` (default `"year"`).
`to.period`	time period for result (default `"year"`).

Details

The result should be applied as a tangent effective decline (see SPEE REP#6 [https://secure.spee.org/sites/default/files/wp-files/pdf/ReferencesResources/REP06-DeclineCurves.pdf]) specified in fractional (i.e. 95% = 0.95) units.

When appropriate, internally uses `rescale.by.time` to perform time unit conversion.

Value

Returns the tangent effective Arps decline rate in units of [1 / `to.period`].

See Also

`as.nominal`, `rescale.by.time`.

Examples

```
## 0.008 nominal daily decline to tangent effective yearly decline
as.effective(0.008, from.period="day", to.period="year")
```

`as.nominal` *Arps decline conversion from effective to nominal*

Description

Convert effective to nominal Arps decline.

Usage

```
as.nominal(D.eff,
    from.period=c("year", "month", "day"),
    to.period=c("year", "month", "day"))
```

Arguments

`D.eff`	tangent effective Arps decline [1 / time].
`from.period`	time period for `D.eff` (default "year").
`to.period`	time period for result (default "year").

Details

`D.eff` should be specified in fractional (i.e. 95% = 0.95) units as a tangent effective decline (see SPEE REP#6 [https://secure.spee.org/sites/default/files/wp-files/pdf/ReferencesResources/REP06-DeclineCurves.pdf]).

When appropriate, internally uses `rescale.by.time` to perform time unit conversion.

Value

Returns the Arps nominal decline rate in units of [1 / `to.period`].

See Also

`as.effective`, `rescale.by.time`.

Examples

```
## 95% / year effective decline to nominal daily decline
as.nominal(0.95, from.period="year", to.period="day")
```

`bestfit` *Best-fitting of Arps decline curves*

Description

Perform best-fits of Arps decline curves to rate or cumulative data.

Usage

```
best.exponential(q, t,
  lower=c( # lower bounds
    0, # qi > 0
    0), # D > 0
  upper=c( # upper bounds
    max(q) * 5, # qi < qmax * 5
    10) # = 0.99995 / [time] effective
  )

best.hyperbolic(q, t,
  lower=c( # lower bounds
    0,  # qi > 0
    0,  # Di > 0
    0), # b > 0
  upper=c( # upper bounds
    max(q) * 5, # qi < qmax * 5
    10, # = 0.99995 / [time] effective
    2)  # b <= 2.0
  )

best.hyp2exp(q, t,
  lower=c( # lower bounds
    0,  # qi > 0
    0.35,  # Di > 0
    0,  # b > 0
    0), # Df > 0
```

```
    upper=c( # upper bounds
      max(q) * 5, # qi < qmax * 5
      10, # = 0.99995 / [time] effective
      2,  # b <= 2.0
      0.35) # Df <= 0.35
    )

  best.exponential.curtailed(q, t,
    lower=c( # lower bounds
      0, # qi > 0
      0, # D > 0
      0  # t.curtail > 0
    ),
    upper=c( # upper bounds
      max(q) * 5, # qi < qmax * 5
      10, # = 0.99995 / [time] effective
      t[length(t)])
    )

  best.hyperbolic.curtailed(q, t,
    lower=c( # lower bounds
      0,  # qi > 0
      0,  # Di > 0
      0,  # b > 0
      0   # t.curtail > 0
    ),
    upper=c( # upper bounds
      max(q) * 5, # qi < qmax * 5
      10, # = 0.99995 / [time] effective
      2,  # b <= 2.0
      t[length(t)])
    )

  best.hyp2exp.curtailed(q, t,
    lower=c( # lower bounds
      0,  # qi > 0
      0.35,  # Di > 0
      0,  # b > 0
      0,  # Df > 0
      0   # t.curtail > 0
    ),
    upper=c( # upper bounds
      max(q) * 5, # qi < qmax * 5
      10, # = 0.99995 / [time] effective
      2,  # b <= 2.0
      0.35, # Df <= 0.35
      t[length(t)])
    )
```

```
best.fit(q, t)

best.curtailed.fit(q, t)

best.exponential.from.Np(Np, t,
  lower=c( # lower bounds
    0, # qi > 0
    0), # D > 0
  upper=c( # upper bounds
    max(c(Np[1], diff(Np)) / diff(c(0, t))) * 5, # qi < max(rate) * 5
    10) # = 0.99995 / [time] effective)
  )

best.exponential.from.interval(volume, t, t.begin=0.0,
  lower=c( # lower bounds
    0, # qi > 0
    0), # D > 0
  upper=c( # upper bounds
    max(volume / diff(c(t.begin, t))) * 5, # qi < max(rate) * 5
    10) # = 0.99995 / [time] effective)
  )

best.hyperbolic.from.Np(Np, t,
  lower=c( # lower bounds
    0,  # qi > 0
    0,  # Di > 0
    0), # b > 0
  upper=c( # upper bounds
    max(c(Np[1], diff(Np)) / diff(c(0, t))) * 5, # qi < max(rate) * 5
    10, # = 0.99995 / [time] effective
    2)  # b <= 2.0
  )

best.hyperbolic.from.interval(volume, t, t.begin=0.0,
  lower=c( # lower bounds
    0,  # qi > 0
    0,  # Di > 0
    0), # b > 0
  upper=c( # upper bounds
    max(volume / diff(c(t.begin, t))) * 5, # qi < max(rate) * 5
    10, # = 0.99995 / [time] effective
    2)  # b <= 2.0
  )

best.hyp2exp.from.Np(Np, t,
  lower=c( # lower bounds
    0,  # qi > 0
```

```
      0.35,  # Di > 0
      0,  # b > 0
      0), # Df > 0
    upper=c( # upper bounds
      max(c(Np[1], diff(Np)) / diff(c(0, t))) * 5, # qi < max(rate) * 5
      10, # = 0.99995 / [time] effective
      5,  # b <= 2.0
      0.35) # Df <= 0.35
    )

  best.hyp2exp.from.interval(volume, t, t.begin=0.0,
    lower=c( # lower bounds
      0,  # qi > 0
      0.35,  # Di > 0
      0,  # b > 0
      0), # Df > 0
    upper=c( # upper bounds
      max(volume / diff(c(t.begin, t))) * 5, # qi < max(rate) * 5
      10, # = 0.99995 / [time] effective
      5,  # b <= 2.0
      0.35) # Df <= 0.35
    )

  best.exponential.curtailed.from.Np(Np, t,
    lower=c( # lower bounds
      0, # qi > 0
      0, # D > 0
      0  # t.curtail > 0
    ),
    upper=c( # upper bounds
      max(c(Np[1], diff(Np)) / diff(c(0, t))) * 5, # qi < max(rate) * 5
      10, # = 0.99995 / [time] effective
      t[length(t)])
    )

  best.exponential.curtailed.from.interval(volume, t, t.begin=0.0,
    lower=c( # lower bounds
      0, # qi > 0
      0, # D > 0
      0  # t.curtail > 0
    ),
    upper=c( # upper bounds
      max(volume / diff(c(t.begin, t))) * 5, # qi < max(rate) * 5
      10, # = 0.99995 / [time] effective
      t[length(t)])
    )

  best.hyperbolic.curtailed.from.Np(Np, t,
```

```
  lower=c( # lower bounds
    0,  # qi > 0
    0,  # Di > 0
    0,  # b > 0
    0   # t.curtail > 0
  ),
  upper=c( # upper bounds
    max(c(Np[1], diff(Np)) / diff(c(0, t))) * 5, # qi < max(rate) * 5
    10, # = 0.99995 / [time] effective
    5,  # b <= 2.0
    t[length(t)])
  )

best.hyperbolic.curtailed.from.interval(volume, t, t.begin=0.0,
  lower=c( # lower bounds
    0,  # qi > 0
    0,  # Di > 0
    0,  # b > 0
    0   # t.curtail > 0
  ),
  upper=c( # upper bounds
    max(volume / diff(c(t.begin, t))) * 5, # qi < max(rate) * 5
    10, # = 0.99995 / [time] effective
    5,  # b <= 2.0
    t[length(t)])
  )

best.hyp2exp.curtailed.from.Np(Np, t,
  lower=c( # lower bounds
    0,  # qi > 0
    0.35,  # Di > 0
    0,  # b > 0
    0,  # Df > 0
    0
  ),
  upper=c( # upper bounds
    max(c(Np[1], diff(Np)) / diff(c(0, t))) * 5, # qi < max(rate) * 5
    10, # = 0.99995 / [time] effective
    5,  # b <= 2.0
    0.35, # Df <= 0.35
    t[length(t)])
  )

best.hyp2exp.curtailed.from.interval(volume, t, t.begin=0.0,
  lower=c( # lower bounds
    0,  # qi > 0
    0.35,  # Di > 0
    0,  # b > 0
```

```
    0,  # Df > 0
    0
  ),
  upper=c( # upper bounds
    max(volume / diff(c(t.begin, t))) * 5, # qi < max(rate) * 5
    10, # = 0.99995 / [time] effective
    5,  # b <= 2.0
    0.35, # Df <= 0.35
    t[length(t)])
  )

best.fit.from.Np(Np, t)

best.fit.from.interval(volume, t, t.begin=0.0)

best.curtailed.fit.from.Np(Np, t)

best.curtailed.fit.from.interval(volume, t, t.begin=0.0)

best.exponential.with.buildup(q, t,
  lower=c( # lower bounds
    0,  # qi > 0
    0), # D > 0
  upper=c( # upper bounds
    max(q) * 5, # qi < qmax * 5
    10),        # = 0.99995 / [time] effective
  initial.rate=q[1], time.to.peak=t[which.max(q)])

best.hyperbolic.with.buildup(q, t,
  lower=c( # lower bounds
    0,  # qi > 0
    0,  # Di > 0
    0), # b > 0
  upper=c( # upper bounds
    max(q) * 5, # qi < qmax * 5
    10, # = 0.99995 / [time] effective
    2), # b <= 2.0
  initial.rate=q[1], time.to.peak=t[which.max(q)])

best.hyp2exp.with.buildup(q, t,
  lower=c( # lower bounds
    0,  # qi > 0
    0.35,  # Di > 0
    0,  # b > 0
    0), # Df > 0
  upper=c( # upper bounds
    max(q) * 5, # qi < qmax * 5
    10, # = 0.99995 / [time] effective
```

```
    2,  # b <= 2.0
    0.35), # Df <= 0.35
  initial.rate=q[1], time.to.peak=t[which.max(q)])

best.fit.with.buildup(q, t)

best.exponential.from.Np.with.buildup(Np, t,
  lower=c( # lower bounds
    0, # qi > 0
    0), # D > 0
  upper=c( # upper bounds
    max(c(Np[1], diff(Np)) / diff(c(0, t))) * 5, # qi < max(rate) * 5
    10), # = 0.99995 / [time] effective
  initial.rate=Np[1] / t[1],
  time.to.peak=(t[which.max(diff(Np))] + t[which.max(diff(Np)) + 1]) / 2.0)

best.exponential.from.interval.with.buildup(volume, t, t.begin=0.0,
  lower=c( # lower bounds
    0, # qi > 0
    0), # D > 0
  upper=c( # upper bounds
    max(volume / diff(c(t.begin, t))) * 5, # qi < max(rate) * 5
    10), # = 0.99995 / [time] effective
  initial.rate=volume[1] / (t[1] - t.begin),
  time.to.peak=(t - diff(c(t.begin, t)) / 2)[which.max(volume)])

best.hyperbolic.from.Np.with.buildup(Np, t,
  lower=c( # lower bounds
    0,  # qi > 0
    0,  # Di > 0
    0), # b > 0
  upper=c( # upper bounds
    max(c(Np[1], diff(Np)) / diff(c(0, t))) * 5, # qi < max(rate) * 5
    10, # = 0.99995 / [time] effective
    2), # b <= 2.0
  initial.rate=Np[1] / t[1],
  time.to.peak=(t[which.max(diff(Np))] + t[which.max(diff(Np)) + 1]) / 2.0)

best.hyperbolic.from.interval.with.buildup(volume, t, t.begin=0.0,
  lower=c( # lower bounds
    0,  # qi > 0
    0,  # Di > 0
    0), # b > 0
  upper=c( # upper bounds
    max(volume / diff(c(t.begin, t))) * 5, # qi < max(rate) * 5
    10, # = 0.99995 / [time] effective
    2), # b <= 2.0
  initial.rate=volume[1] / (t[1] - t.begin),
```

```
    time.to.peak=(t - diff(c(t.begin, t)) / 2)[which.max(volume)])

  best.hyp2exp.from.Np.with.buildup(Np, t,
    lower=c( # lower bounds
      0,  # qi > 0
      0.35,  # Di > 0
      0,  # b > 0
      0), # Df > 0
    upper=c( # upper bounds
      max(c(Np[1], diff(Np)) / diff(c(0, t))) * 5, # qi < max(rate) * 5
      10, # = 0.99995 / [time] effective
      5,  # b <= 2.0
      0.35), # Df <= 0.35
    initial.rate=Np[1] / t[1],
    time.to.peak=(t[which.max(diff(Np))] + t[which.max(diff(Np)) + 1]) / 2.0)

  best.hyp2exp.from.interval.with.buildup(volume, t, t.begin=0.0,
    lower=c( # lower bounds
      0,  # qi > 0
      0.35,  # Di > 0
      0,  # b > 0
      0), # Df > 0
    upper=c( # upper bounds
      max(volume / diff(c(t.begin, t))) * 5, # qi < max(rate) * 5
      10, # = 0.99995 / [time] effective
      5,  # b <= 2.0
      0.35), # Df <= 0.35
    initial.rate=volume[1] / (t[1] - t.begin),
    time.to.peak=(t - diff(c(t.begin, t)) / 2)[which.max(volume)])

  best.fit.from.Np.with.buildup(Np, t)

  best.fit.from.interval.with.buildup(volume, t, t.begin=0.0)
```

Arguments

q	vector of rate data.
Np	vector of cumulative production data.
volume	vector of interval volume data.
t	vector of times at which q, Np, or volume is measured.
t.begin	initial time for interval volume data, if non-zero.
lower	lower bounds for decline parameters (sane defaults are provided).
upper	upper bounds for decline parameters (sane defaults are provided).
initial.rate	initial rate, for declines with buildup.
time.to.peak	time to peak rate, for declines with buildup.

Details

Best-fitting is carried out by minimizing the sum of squared error in the rate or cumulative forecast, using `nlminb` as the optimizer.

Appropriate bounds are applied to decline-curve parameters by default, but may be altered using the `lower` and `upper` arguments to each specific function.

Value

`best.exponential`, `best.hyperbolic`, and `best.hyp2exp` return objects of the appropriate class (as from `arps.decline`) representing best fits of the appropriate type against `q` and `t`, in the same units as `q` and `t`.

`best.fit` returns the best overall fit, considering results from each function above.

`best.exponential.from.Np`, `best.hyperbolic.from.Np`, and `best.hyp2exp.from.Np` return objects of the appropriate class (as from `arps.decline`) representing best fits of the appropriate type against `Np` and `t`, in the same units as `Np` and `t`.

`best.fit.from.Np` returns the best overall fit, considering results from each function above.

`best.exponential.from.interval`, `best.hyperbolic.from.interval`, and `best.hyp2exp.from.interval` return objects of the appropriate class (as from `arps.decline`) representing best fits of the appropriate type against `volume` and `t`, in the same units as `volume` and `t`.

For these functions, `t` is taken to represent the time at the end of each producing interval; the beginning time for the first interval may be specified as `t.begin` if it is non-zero.

`best.fit.from.interval` returns the best overall fit, considering results from each function above.

`best.exponential.curtailed`, `best.hyperbolic.curtailed`, `best.hyp2exp.curtailed`, `best.curtailed.fit`, `best.exponential.curtailed.from.Np`, `best.hyperbolic.curtailed.from.Np`, `best.hyp2exp.curtailed.from.Np`, `best.curtailed.fit.from.Np`, `best.exponential.curtailed.from.interval`, `best.hyperbolic.curtailed.from.i` `best.hyp2exp.curtailed.from.interval`, and `best.curtailed.fit.from.interval` work as the corresponding functions above, but may return curtailed declines (as from `curtail`).

`best.exponential.with.buildup`, `best.hyperbolic.with.buildup`, `best.hyp2exp.with.buildup`, `best.fit.with.buildup`, `best.exponential.from.Np.with.buildup`, `best.hyperbolic.from.Np.with.buildup`, `best.hyp2exp.from.Np.with.buildup`, `best.fit.from.Np.with.buildup`, `best.exponential.from.interval.with` `best.hyperbolic.from.interval.with.buildup`, `best.hyp2exp.from.interval.with.buildup`, and `best.fit.from.interval.with.buildup` work as the corresponding functions above, but will return a fit including a linear buildup portion (as from `arps.with.buildup`).

See Also

`arps`, `curtailed`, `arps.with.buildup`, `nlminb`

Examples

```
fitme.hyp2exp.t <- seq(0, 5, 1 / 12) # 5 years
fitme.hyp2exp.q <- hyp2exp.q(
    1000, # Bbl/d
    as.nominal(0.70), # / year
    1.9,
    as.nominal(0.15), # / year
```

```
    fitme.hyp2exp.t
) * rnorm(n=length(fitme.hyp2exp.t), mean=1, sd=0.1) # perturb

hyp2exp.fit <- best.hyp2exp(fitme.hyp2exp.q, fitme.hyp2exp.t)
cat(paste("SSE:", hyp2exp.fit$sse))
dev.new()
plot(fitme.hyp2exp.q ~ fitme.hyp2exp.t, main="Hyperbolic-to-Exponential Fit",
     col="blue", log="y", xlab="Time", ylab="Rate")
lines(arps.q(hyp2exp.fit$decline, fitme.hyp2exp.t) ~ fitme.hyp2exp.t,
      col="red")
legend("topright", pch=c(1, NA), lty=c(NA, 1), col=c("blue", "red"), legend=c("Actual", "Fit"))
```

`curtailed` *Arps decline curves with initial curtailment*

Description

Create decline curve objects and compute rates, cumulative production, and nominal declines for Arps decline curves with an initial period of curtailment to constant rate. The resulting decline curve models will impose a constant rate of production equal to the initial rate of the underlying model until the specified end-of-curtailment time.

Usage

```
curtail(decl, t.curtail)
curtailed.q(decl, t.curtail, t)
curtailed.Np(decl, t.curtail, t)
curtailed.D(decl, t.curtail, t)
```

Arguments

`decl`	an Arps decline object as returned by `arps.decline`.
`t.curtail`	time to end of curtailment, in same units as `t` [time].
`t`	time at which to evaluate rate, cumulative, or nominal decline [time].

Details

Assumes consistent units as described for underlying Arps decline types.

Value

`curtail` returns an object having class `"arps"`, suitable for use as an argument to S3 methods listed in `arps`.

`curtailed.q` returns the rate for each element of `t`, under a decline described by `decl` and curtailed until time `t.curtail`.

`curtailed.Np` returns the cumulative production for each element of `t`, under a decline described by `decl` and curtailed until time `t.curtail`.

`curtailed.D` returns the nominal instantaneous decline for each element of `t`, under a decline described by `decl` and curtailed until time `t.curtail`.

See Also

arps, print.arps, exponential, hyperbolic, hyp2exp.

Examples

```
## qi = 1000 Mscf/d, Di = 95% effective / year, b = 1.2, t from 0 to 25 days,
## curtailed until 5 days
curtailed.q(arps.decline(
        1000,
        as.nominal(0.95, from.period="year", to.period="day"),
        1.2
    ),
    5, seq(0, 25))

## hyperbolic decline with
## qi = 500 bopd, Di = 3.91 nominal / year, b = 1.5,
## curtailed for 1 month
## cumulative production at t = 5 years
decline <- curtail(
    arps.decline(rescale.by.time(500, from="day", to="year"),
                 3.91, 1.5),
    (1 / 12)
)
arps.Np(decline, 5)
```

---

exponential *Arps exponential declines*

Description

Compute rates and cumulative production values for Arps exponential decline curves.

Usage

```
exponential.q(qi, D, t)
exponential.Np(qi, D, t)
```

Arguments

qi	initial rate [volume / time], i.e. q(t = 0).
D	nominal Arps decline exponent [1 / time].
t	time at which to evaluate rate or cumulative [time].

Details

Assumes consistent units of time between `qi`, `D`, and `t`. To convert, see the decline-rate conversion functions referenced below.

Value

`exponential.q` returns the rate for each element of `t`, in the same units as `qi`.

`exponential.Np` returns the cumulative production for each element of `t`, in the same units as `qi * t`.

See Also

`as.effective`, `as.nominal`, `rescale.by.time`.

Examples

```
## qi = 1000 Mscf/d, Di = 95% effective / year, t from 0 to 25 days
exponential.q(1000, as.nominal(0.95, from.period="year", to.period="day"), seq(0, 25))

## qi = 500 bopd, Di = 3.91 nominal / year, t = 5 years
exponential.Np(rescale.by.time(500, from.period="day", to.period="year"), 3.91, 5)
```

`format.arps` *Format methods for Arps decline objects*

Description

Get human-readable representation of Arps decline-curve objects.

Usage

```
## S3 method for class 'arps'
format(x, ...)
```

Arguments

`x`	Arps decline curve object as returned from `arps.decline`.
`...`	Arguments to additional `format` methods.

Value

A character representation of `x`.

See Also

`format`, `print.arps`, `arps.decline`.

Examples

```
## exponential decline with
## qi = 1000 Mscf/d, Di = 95% effective / year
## rate for t from 0 to 25 days
decline <- arps.decline(1000,
    as.nominal(0.95, from.period="year", to.period="day"))
format(decline, digits=2)
```

---

harmonic *Arps harmonic declines*

Description

Compute rates, cumulative production values, and instantaneous nominal declines for Arps harmonic decline curves (i.e. hyperbolic with b = 1).

Usage

```
harmonic.q(qi, Di, t)
harmonic.Np(qi, Di, t)
harmonic.D(Di, t)
```

Arguments

`qi`	initial rate [volume / time], i.e. q(t = 0).
`Di`	initial nominal Arps decline exponent [1 / time].
`t`	time at which to evaluate rate or cumulative [time].

Details

Assumes consistent units of time between `qi`, `D`, and `t`. To convert, see the decline-rate conversion functions referenced below.

Value

`harmonic.q` returns the rate for each element of `t`, in the same units as `qi`.

`harmonic.Np` returns the cumulative production for each element of `t`, in the same units as `qi * t`.

`harmonic.D` returns the nominal instantaneous decline for each element of `t`. This can be converted to effective decline and rescaled in time by use of `as.effective` and `rescale.by.time`.

See Also

`as.effective`, `as.nominal`, `rescale.by.time`.

Examples

```
## qi = 1000 Mscf/d, Di = 95% effective / year, t from 0 to 25 days
harmonic.q(1000, as.nominal(0.95, from.period="year", to.period="day"), seq(0, 25))

## qi = 500 bopd, Di = 3.91 nominal / year, t = 5 years
harmonic.Np(rescale.by.time(500, from.period="day", to.period="year"), 3.91, 5)

## Di = 85% effective / year, t = 6 months
harmonic.D(as.nominal(0.85), 0.5)
```

`hyp2exp` *Arps hyperbolic-to-exponential declines*

Description

Compute rates, cumulative production values, instantaneous nominal declines, and transition times for Arps hyperbolic-to-exponential decline curves.

Usage

```
hyp2exp.q(qi, Di, b, Df, t)
hyp2exp.Np(qi, Di, b, Df, t)
hyp2exp.D(Di, b, Df, t)
hyp2exp.transition(Di, b, Df)
```

Arguments

`qi`	initial rate [volume / time], i.e. q(t = 0).
`Di`	initial nominal Arps decline exponent [1 / time].
`b`	Arps hyperbolic exponent.
`Df`	final nominal Arps decline exponent [1 / time].
`t`	time at which to evaluate rate or cumulative [time].

Details

Assumes consistent units of time between `qi`, `Di`, `Df`, and `t`. To convert, see the decline-rate conversion functions referenced below.

When appropriate, internally uses `harmonic.q` and `harmonic.Np` to avoid singularities in calculations for `b` near 1.

Value

`hyp2exp.q` returns the rate for each element of `t`, in the same units as `qi`.

`hyp2exp.Np` returns the cumulative production for each element of `t`, in the same units as `qi * t`.

`hyp2exp.D` returns the nominal instantaneous decline for each element of `t`. This can be converted to effective decline and rescaled in time by use of `as.effective` and `rescale.by.time`.

`hyp2exp.transition` returns the transition time (from hyperbolic to exponential decline), in the same units as `1 / Di`.

See Also

as.effective, as.nominal, rescale.by.time.

Examples

```
## qi = 1000 Mscf/d, Di = 95% effective / year, b = 1.2,
## Df = 15% effective / year, t from 0 to 25 years
## result in Mscf/d
hyp2exp.q(1000, as.nominal(0.95), 1.2, as.nominal(0.15), seq(0, 25))

## qi = 500 bopd, Di = 3.91 nominal / year, b = 0.5,
## Df = 0.3 nominal / year, t = 20 years
hyp2exp.Np(rescale.by.time(500, from.period="day", to.period="year"),
    3.91, 0.5, 0.3, 20)

## Di = 85% effective / year, b = 1.5, Df = 15% effective / year,
## t = 6 and 48 months
hyp2exp.D(as.nominal(0.85), 1.5, as.nominal(0.15), c(0.5, 4))

## Di = 85% effective / year, b = 1.5, Df = 5% effective / year
hyp2exp.transition(as.nominal(0.85), 1.5, as.nominal(0.05))
```

---

hyperbolic *Arps hyperbolic declines*

Description

Compute rates, cumulative production values, and instantaneous nominal declines for Arps hyperbolic decline curves.

Usage

```
hyperbolic.q(qi, Di, b, t)
hyperbolic.Np(qi, Di, b, t)
hyperbolic.D(Di, b, t)
```

Arguments

qi	initial rate [volume / time], i.e. $q(t = 0)$.
Di	initial nominal Arps decline exponent [1 / time].
b	Arps hyperbolic exponent.
t	time at which to evaluate rate or cumulative [time].

Details

Assumes consistent units of time between `qi`, `D`, and `t`. To convert, see the decline-rate conversion functions referenced below.

When appropriate, internally uses `harmonic.q` and `harmonic.Np` to avoid singularities in calculations for b near 1.

Value

hyperbolic.q returns the rate for each element of t, in the same units as qi.

hyperbolic.Np returns the cumulative production for each element of t, in the same units as qi * t.

hyperbolic.D returns the nominal instantaneous decline for each element of t. This can be converted to effective decline and rescaled in time by use of as.effective and rescale.by.time.

See Also

as.effective, as.nominal, rescale.by.time.

Examples

```
## qi = 1000 Mscf/d, Di = 95% effective / year, b = 1.2, t from 0 to 25 days
hyperbolic.q(1000, as.nominal(0.95, from.period="year", to.period="day"),
    1.2, seq(0, 25))

## qi = 500 bopd, Di = 3.91 nominal / year, b = 0.5, t = 5 years
hyperbolic.Np(rescale.by.time(500, from.period="day", to.period="year"),
    3.91, 0.5, 5)

## Di = 85% effective / year, b = 1.5, t = 6 months
hyperbolic.D(as.nominal(0.85), 1.5, 0.5)
```

---

print.arps — *Print methods for Arps decline objects*

Description

Print human-readable representation of Arps decline-curve objects using format.arps.

Usage

```
## S3 method for class 'arps'
print(x, ...)
```

Arguments

x	Arps decline curve object as returned from arps.decline.
...	Arguments to format.arps.

Value

Invisibly (see invisible) returns x.

See Also

print, format.arps, arps.decline.

Examples

```
## exponential decline with
## qi = 1000 Mscf/d, Di = 95% effective / year
## rate for t from 0 to 25 days
decline <- arps.decline(1000,
    as.nominal(0.95, from.period="year", to.period="day"))
print(decline)
```

---

rescale.by.time *Time unit conversion for DCA*

Description

Scales rates, declines, and time periods from one time unit to another.

Usage

```
rescale.by.time(value,
    from.period=c("year", "month", "day"),
    to.period=c("year", "month", "day"),
    method=c("decline", "rate", "time"))
```

Arguments

`value`	rate [volume / time], Arps nominal decline [1 / time], or time to be rescaled.
`from.period`	time period for `value` (default `"year"`).
`to.period`	time period for result (default `"year"`).
`method`	scaling method to be applied, depending upon the type of `value` (default "decline").

Value

Returns `value` scaled from `from.period` to `to.period` according to its type as specified by `method`.

See Also

`as.nominal`, `as.effective`.

Examples

```
## 3 MMscf/D to MMscf/year
rescale.by.time(3, from.period="day", to.period="year", method="rate")

## Nominal decline of 3.2/year to nominal decline per month
rescale.by.time(3.2, from.period="year", to.period="month", method="decline")

## 5 years in days
rescale.by.time(5, from.period="year", to.period="month", method="time")
```

Index

∗ **package**
aRpsDCA-package, 2

arps, 2, 3, 5, 16–18
arps.D, 6
arps.D.curtailed (curtailed), 17
arps.decline, 6, 16, 19, 23
arps.eur, 2, 4
arps.Np, 6
arps.Np.curtailed (curtailed), 17
arps.q, 6
arps.q.curtailed (curtailed), 17
arps.t.el (arps.eur), 4
arps.with.buildup, 5, 16
aRpsDCA (aRpsDCA-package), 2
aRpsDCA-package, 2
as.effective, 2, 4, 6, 8, 19–24
as.nominal, 2, 4, 7, 7, 19, 20, 22–24

best.curtailed.fit (bestfit), 8
best.exponential (bestfit), 8
best.fit (bestfit), 8
best.hyp2exp (bestfit), 8
best.hyperbolic (bestfit), 8
bestfit, 2, 8

curtail, 2, 5, 16
curtail (curtailed), 17
curtailed, 16, 17

exponential, 2, 4, 18, 18

format, 19
format.arps, 19, 23

harmonic, 2, 20
harmonic.Np, 21, 22
harmonic.q, 21, 22
hyp2exp, 2, 4, 18, 21
hyperbolic, 2, 4, 18, 22

invisible, 23

nlminb, 16

print, 23
print.arps, 2, 4, 6, 18, 19, 23

rescale.by.time, 2, 4, 7, 8, 19–23, 24