Package 'collatz'

September 3, 2022

Version 1.0.0

License Apache License (== 2.0)

Title Functions Related to the Collatz/Syracuse/3n+1 Problem

Description Provides the basic functionality to interact with the Collatz conjecture.
The parameterisation uses the same (P,a,b) notation as Conway's generalisations.
Besides the function and reverse function, there is also functionality to retrieve
the hailstone sequence, the ``stopping time"/``total stopping time", or tree-graph.
The only restriction placed on parameters is that both P and a can't be 0.
For further reading, see <https://en.wikipedia.org/wiki/Collatz_conjecture>.

Author Nathan Levett [aut, cre]

Maintainer Nathan Levett <nathan.a.z.levett@gmail.com>

Date 2022-08-08

URL https://github.com/Skenvy/Collatz, https://github.com/Skenvy/Collatz/tree/main/R, https://skenvy.github.io/Collatz/R/, https://skenvy.github.io/Collatz/R/pdf/

BugReports https://github.com/Skenvy/Collatz/issues/new/choose

Encoding UTF-8

Depends R (>= 3.5.0),
gmp

Collate 'utils.R'
'collatz_function.R'
'hailstone_sequence.R'
'reverse_function.R'
'stopping_time.R'
'tree_graph.R'

Suggests roxygen2,
testthat (>= 3.0.0),
devtools,
covr,
DT,
pkgdown,
servr,

tinytex,
knitr,
rmarkdown

Config/testthat/edition 3

RoxygenNote 7.2.1

VignetteBuilder knitr

R topics documented:

`assert_sane_parameterication` *Sane Parameter Check*

Description

Handles the sanity check for the parameterisation (P,a,b)

Usage

```
assert_sane_parameterication(P, a, b)
```

Arguments

P	Modulus used to devide n, iff n is equivalent to (0 mod P).
a	Factor by which to multiply n.
b	Value to add to the scaled value of n.

Details

Required by both the function and reverse function, to assert that they have sane parameters, otherwise will force stop the execution.

collatz *Collatz*

Description

Functions related to the Collatz/Syracuse/3N+1 problem.

Details

Provides the basic functionality to interact with the Collatz conjecture. The parameterisation uses the same (P,a,b) notation as Conway's generalisations. Besides the function and reverse function, there is also functionality to retrieve the hailstone sequence, the "stopping time"/"total stopping time", or tree-graph. The only restriction placed on parameters is that both P and a can't be 0.

collatz_function *The Collatz function*

Description

Returns the output of a single application of a Collatz-esque function.

Usage

```
collatz_function(n, P = 2, a = 3, b = 1)
```

Arguments

n	(numeric|bigz) The value on which to perform the Collatz-esque function
P	(numeric|bigz): Modulus used to divide n, iff n is equivalent to (0 mod P). Default is 2.
a	(numeric|bigz) Factor by which to multiply n. Default is 3.
b	(numeric|bigz) Value to add to the scaled value of n. Default is 1.

Details

This function will compute and return the result of applying one iteration of a parameterised Collatz-esque function. Although it will operate with integer inputs, for overflow safety, provide a gmp Big Integer ('bigz').

Value

a numeric, either in-built or a bigz from the gmp library.

Examples

```
# Returns the output of a single application of a Collatz-esque function.
# Without `gmp` or parameterisation, we can try something simple like
collatz_function(5)
collatz_function(16)
# If we want change the default parameterisation we can;
collatz_function(4, 5, 2, 3)
# Or if we only want to change one of them
collatz_function(3, a=-2)
# All the above work fine, but the function doesn't offer protection against
# overflowing integers by default. To venture into the world of arbitrary
# integer inputs we can use an `as.bigz` from `gmp`. Compare the two;
collatz_function(99999999999999999999)
collatz_function(as.bigz("99999999999999999999"))
```

---

hailstone_sequence *Hailstone Sequencer*

Description

Calculates the values of a hailstone sequence, from an initial value.

Usage

```
hailstone_sequence(
  initial_value,
  P = 2,
  a = 3,
  b = 1,
  max_total_stopping_time = 1000,
  total_stopping_time = TRUE,
  verbose = TRUE
)
```

Arguments

initial_value	(numeric|bigz) The value to begin the hailstone sequence from.
P	(numeric|bigz): Modulus used to divide n, iff n is equivalent to (0 mod P). Default is 2.
a	(numeric|bigz) Factor by which to multiply n. Default is 3.
b	(numeric|bigz) Value to add to the scaled value of n. Default is 1.
max_total_stopping_time	(int) Maximum amount of times to iterate the function, if 1 is not reached. Default is 1000.
total_stopping_time	(bool) Whether or not to execute until the "total" stopping time (number of iterations to obtain 1) rather than the regular stopping time (number of iterations to reach a value less than the initial value). Default is TRUE.

verbose (bool) If set to verbose, the hailstone sequence will include control string sequences to provide information about how the sequence terminated, whether by reaching a stopping time or entering a cycle. Default is TRUE.

Details

Returns a list of successive values obtained by iterating a Collatz-esque function, until either 1 is reached, or the total amount of iterations exceeds max_total_stopping_time, unless total_stopping_time is FALSE, which will terminate the hailstone at the "stopping time" value, i.e. the first value less than the initial value. While the sequence has the capability to determine that it has encountered a cycle, the cycle from "1" wont be attempted or reported as part of a cycle, regardless of default or custom parameterisation, as "1" is considered a "total stop".

Value

A keyed list consisting of a $values list of numeric | bigz along with a $terminalCondition and $terminalStatus

Examples

```
# Compute a hailstone sequence, which defaults to the total stopping time;
hailstone_sequence(5)
# Or only compute down to the regular stopping time;
hailstone_sequence(5, total_stopping_time=FALSE)
# Remove verbose messaging;
hailstone_sequence(5, verbose=FALSE)
# It will also stop on finding a cycle;
hailstone_sequence(-56)
# And can be parameterised;
hailstone_sequence(3, -1, 3, 1)
# The hailstone sequence can run on `bigz`;
hailstone_sequence(27+as.bigz("576460752303423488"))
```

---

reverse_function *The "inverse"/"reverse" Collatz function*

Description

Calculates the values that would return the input under the Collatz function.

Usage

```
reverse_function(n, P = 2, a = 3, b = 1)
```

Arguments

n	(numeric|bigz) The value on which to perform the reverse Collatz function
P	(numeric|bigz) Modulus used to divide n, iff n is equivalent to (0 mod P) Default is 2.
a	(numeric|bigz) Factor by which to multiply n. Default is 3.
b	(numeric|bigz) Value to add to the scaled value of n. Default is 1.

Details

Returns the output of a single application of a Collatz-esque reverse function. If only one value is returned, it is the value that would be divided by P. If two values are returned, the first is the value that would be divided by P, and the second value is that which would undergo the multiply and add step, regardless of which is larger.

Value

A list of either numeric or bigz type

Examples

```
# Calculates the values that would return the input under the Collatz
# function. Without `gmp` or parameterisation, we can try something
# simple like
reverse_function(1)
reverse_function(2)
reverse_function(4)
# If we want change the default parameterisation we can;
reverse_function(3, -3, -2, -5)
# Or if we only want to change one of them
reverse_function(16, a=5)
# All the above work fine, but the function doesn't offer protection against
# overflowing integers by default. To venture into the world of arbitrary
# integer inputs we can use an `as.bigz` from `gmp`. Compare the two;
reverse_function(99999999999999999999)
reverse_function(as.bigz("99999999999999999999"))
```

---

stopping_time *Stopping Time*

Description

Determine the stopping time, or "total" stopping time, for an initial value.

Usage

```
stopping_time(
  initial_value,
  P = 2,
  a = 3,
  b = 1,
  max_stopping_time = 1000,
  total_stopping_time = FALSE
)
```

Arguments

initial_value	(int): The value for which to find the stopping time.
P	(numeric|bigz): Modulus used to divide n, iff n is equivalent to (0 mod P). Default is 2.
a	(numeric|bigz) Factor by which to multiply n. Default is 3.
b	(numeric|bigz) Value to add to the scaled value of n. Default is 1.
max_stopping_time	(int) Maximum amount of times to iterate the function, if the stopping time is not reached. IF the max_stopping_time is reached, the function will return NaN. Default is 1000.
total_stopping_time	(bool) Whether or not to execute until the "total" stopping time (number of iterations to obtain 1) rather than the regular stopping time (number of iterations to reach a value less than the initial value). Default is FALSE.

Details

Returns the stopping time, the amount of iterations required to reach a value less than the initial value, or NaN if max_stopping_time is exceeded. Alternatively, if total_stopping_time is TRUE, then it will instead count the amount of iterations to reach 1. If the sequence does not stop, but instead ends in a cycle, the result will be (Inf). If (P,a,b) are such that it is possible to get stuck on zero, the result will be the negative of what would otherwise be the "total stopping time" to reach 1, where 0 is considered a "total stop" that should not occur as it does form a cycle of length 1.

Value

An integer numeral if stopped, Inf if a cycle, NaN if OOB, else NA.

Examples

```
# Calculates the "stopping time", or optionally the "total" stopping time.
# Without `gmp` or parameterisation, we can try something simple like
stopping_time(27)
stopping_time(27, total_stopping_time=TRUE)
# If we want change the default parameterisation we can;
stopping_time(3, 5, 2, 1)
# Or if we only want to change one of them
```

```
stopping_time(17, a=5)
# All the above work fine, but the function doesn't offer protection against
# overflowing integers by default. To venture into the world of arbitrary
# integer inputs we can use an `as.bigz` from `gmp`. Compare the two;
stopping_time(99999999999999999)
stopping_time(as.bigz("99999999999999999"))
# As an extra note, the original motivation for creating a range of Collatz
# themed packages came from some earlier scripts for calculating the stopping
# distances under certain parameterisations. An inconsequential result of
# which was observing that all of the following, for however high `k` goes,
# should equal `96`!
stopping_time(27)
stopping_time(27+as.bigz("576460752303423488"))
stopping_time(27+(2*as.bigz("576460752303423488")))
stopping_time(27+(3*as.bigz("576460752303423488")))
stopping_time(27+(4*as.bigz("576460752303423488")))
```

`stopping_time_terminus` *Stopping Time Terminus*

Description

Provides the appropriate lambda to use to check if iterations on an initial value have reached either the stopping time, or total stopping time.

Usage

```
stopping_time_terminus(n, total_stop)
```

Arguments

`n`	The initial value to confirm against a stopping time check.
`total_stop`	If false, the lambda will confirm that iterations of n have reached the oriented stopping time to reach a value closer to 0. If true, the lambda will simply check equality to 1.

Value

An anonymous function to check for the stopping time.

tree_graph *Tree Graph*

Description

Determine the Tree Graph to some depth by iteratively reversing values.

Usage

```
tree_graph(
  initial_value,
  max_orbit_distance,
  P = 2,
  a = 3,
  b = 1,
  cycle_prevention = list()
)
```

Arguments

initial_value	(int) The root value of the directed tree graph.
max_orbit_distance	(int) Maximum amount of times to iterate the reverse function. There is no natural termination to populating the tree graph, equivalent to the termination of hailstone sequences or stopping time attempts, so this is not an optional argument like max_stopping_time or max_total_stopping_time, as it is the intended target of orbits to obtain, rather than a limit to avoid uncapped computation.
P	(numeric|bigz): Modulus used to divide n, iff n is equivalent to (0 mod P). Default is 2.
a	(numeric|bigz) Factor by which to multiply n. Default is 3.
b	(numeric|bigz) Value to add to the scaled value of n. Default is 1.
cycle_prevention	(set[int]) Used to prevent cycles from precipitating by keeping track of all values added across previous nest depths. Only to be used internally by the function recursing. Does not expect input.

Details

Returns nested dictionaries that model the directed tree graph up to a maximum nesting of max_orbit_distance, with the initial_value as the root.

Value

A set of nested dictionaries.

Examples

```
#Compute a tree graph, which takes both a value to initialise the tree from,
# and an "orbit distance" for how many layers deep in the tree to compute;
tree_graph(16, 3)
# It will also stop on finding a cycle;
tree_graph(4, 3)
# And can be parameterised;
tree_graph(1, 1, -3, -2, -5)
# If b is a multiple of a, but not of Pa, then 0 can have a reverse;
tree_graph(0, 1, 17, 2, -6)
# The tree graph can run on `bigz`;
tree_graph((27+as.bigz("576460752303423488")), 3)
```