Funciones para crear o comprobar vectores de tipo doble

integer {base}: enteros
Documentación de R, 4.0.0

Contenido

1 Descripción

La función `double()` creará un vector de tipo *doble* (`"double"`) que podrá almacenar un subconjunto muy amplio de los números reales hasta cierto nivel de precisión.

La función `as.double()` intentará convertir en un vector de tipo *doble* a cualquier tipo de objeto. Por su parte, la función `is.double()` verificará si un objeto es de tipo *doble*.

La función `single()` creará también un vector de tipo *doble* para almacenamiento de números reales, pero añadirá un atributo que permitirá identificar al vector como uno de tipo *simple* a nivel interno del código de **R**. Para más información, puedes ver más adelante la sección *Valor devuelto*.

La función `as.single()` intentará coaccionar cualquier objeto hacia un vector de tipo *doble* y añadirá el atributo mencionado.

2 Forma de uso o sintaxis

```
double(length = 0)
as.double(x, ...)
is.double(x)

single(length = 0)
as.single(x, ...)
```

3 Argumentos

Tabla 1: Argumentos para las funciones de creación y verificación de vectores *dobles*

Argumento	Valor esperado	Propósito
`length=` *longitud*	Un valor entero mayor o igual a cero	Determina la longitud deseada del vector, es decir, el número de elementos que almacenará. El argumento de longitud aceptará números enteros no negativos. Los valores continuos o con decimales (tipo doble) serán convertidos a enteros y la aportación de más un valor devolverá un mensaje de error.
`x=`	Un objeto	Un objeto para ser coaccionado o verificado como vector de tipo *doble*.
`...`	Otros argumentos	Otros argumentos que serán pasados desde o hacia otras funciones.

4 Detalles

Un vector *doble* es una estructura fundamental de **R** destinada al almacenamiento exclusivo de datos de tipo *doble* en celdas contiguas, así como a la realización de operaciones individuales o en paralelo con los elementos de estas celdas. Los datos de tipo *doble* consisten en un formato de representación de números reales basado en la aritmética de *punto flotante*.

La aritmética de *punto flotante* es el formato más extendido entre las computadoras actuales para aproximarse a la aritmética de los números reales y realizar cálculos con ellos. En esencia, es un formato de almacenamiento que permite simular un conjunto infinito de números continuos —los reales— con los elementos de un conjunto finito de números

discretos —los dígitos de la máquina— (Mullller et al. 2018, 3). El nombre del formato proviene de la posibilidad de colocar el separador decimal —el punto— en cualquiera de los dígitos disponibles para almacenar una cifra, lo que permite ajustar el nivel de precisión del número representado. Si deseas obtener más información al respecto, puedes consultar más adelante la sección *La aritmética de punto flotante*.

Específicamente, **R** define al tipo *doble* como una aplicación de la aritmética de *punto flotante* para representar a un subconjunto de los números reales con signo, separador de decimales, y precisión limitada, en un tamaño de memoria de 64 dígitos binarios (*bits*) por cifra completa. Esto equivale a la posibilidad de almacenar valores reales en el intervalo aproximado de -1.8×10^{308} a 1.8×10^{308}, o, en valor absoluto, entre $|\mp 2 \times 10^{-308}|$ y $|\mp 1.8 \times 10^{308}|$. Así, los vectores *dobles* son el único tipo de dato en **R** que permite guardar números reales y realizar operaciones aritméticas con ellos.

El primer tamaño de memoria estandarizado para guardar números reales fue de 32 *bits* y se denominó de *precisión simple* en el estándar 754-1985 del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE por sus siglas en inglés) de Estados Unidos. El siguiente rango de almacenamiento para números reales fue de 64 *bits*, por lo que se le denominó de *precisión doble*. En la actualidad la denominación en dicho estándar (754-2019) para la *precisión simple* es *binaria32* (*binary32*) y para la *precisión doble, binaria64* (*binary64*) .

La precisión doble implica que un número real podrá almacenarse con entre 15 y 17 cifras significativas exactas. A partir del dígito 16, algunos valores podrán ser almacenados de forma aproximada, debido a las características propias del almacenamiento de *punto flotante*. A partir del dígito 18, las cifras almacenadas tendrán una precisión inexacta. Finalmente, el límite máximo de cifras significativas representables para cualquier número real es de 22 dígitos.

Si deseas trabajar con números de más de 15 cifras significativas es recomendable el uso de paquetes especializados. Por ejemplo, el paquete `Rmpfr` permite realizar operaciones aritméticas con una precisión arbitraria. Con todo, los instrumentos físicos más avanzados de la actualidad no suelen medir magnitudes menores a 10^{-15}, y muy pocos campos de la ciencia necesitan precisiones mayores a este umbral, por lo que un amplio número de aplicaciones estadísticas pueden realizarse apropiadamente con la *precisión doble* (Mullller *et al*. 2018, 3).

Como es evidente, los límites del almacenamiento para números reales en **R** ($\mp 1.8 \times 10^{308}$) comprenden cifras mucho más precisas que el despliegue, en el mejor de los casos, de 17 cifras significativas exactas. Para resolver esta aparente contradicción, **R** utiliza la notación cienftífica E, que es una forma abreviada de la notación científica tradicional. Debido a que el despliegue de exponentes con superíndices (como en 10^n) no puede realizarse en el codigo fuente, **R** utiliza la notación `Me+N` para denotar "un número *M* multiplicado por 10 y elevado a la potencia de un número *N*", lo que en notación científica tradicional sería escrito como: $m \times 10^n$.

De este modo, las funciones básicas para crear vectores *dobles* son `double()`, `as.double()` y `c()`. Por su parte, la función `is.double()` se utilizará para comprobar si un objeto es de tipo *doble* (`"double"`). En particular, `as.double()`, `is.double()` y `c()` son funciones primitivas, por lo que su código fuente está implementado de manera interna y no será visible directamente por la usuaria. Para más información sobre la forma de utilización de las funciones mencionadas, puedes consultar la sección *Valor devuelto* en esta misma página de ayuda.

Las funciones `single()` y `as.single()` crearán también vectores de tipo *doble*, pero tendrán asignados el atributo adicional `"Csingle"` que indicará a nivel interno que los objetos creados deberán ser interpretados como un vectores de precisión *simple*. Los vectores *simples* pueden almacenar números reales con una precisión de siete cifras significativas exactas en el intervalo aproximado de $\mp 3.4 \times 10^{38}$.

Como objetos, los vectores *dobles* son un tipo de vector *atómico* o fundamental de **R**, por lo que no podrán convertirse en objetos más simples ni contener elementos que no sean del mismo tipo. Los vectores atómicos están definidos a nivel interno por su tipo de almacenamiento. El tipo de almacenamiento determina el tipo de vector y las operaciones que le son aplicables.

El tipo interno de almacenamiento de un vector *doble* tendrá asignada como atributo la etiqueta inglesa *doble* (`"double"`) y su modo de almacenamiento tendrá asociada como atributo la etiqueta inglesa *numérico* (`"numeric"`). Además, los vectores *dobles* pertenecen a la clase `"numeric"`, por lo que también podrán extender sus propiedades a nuevos objetos.

Los vectores de tipo *doble* (`"double"`) podrán almacenar valores enteros en forma exacta más allá del rango de almacenamiento del tipo *entero* (`"integer"`), el cual va, aproximadamente, de $\mp 2 \times 10^9$.

Para cada tipo de vector atómico, salvo para los vectores crudos (`"raw"`), existe un tipo propio de valor no disponible `NA`. Así, al tipo *doble* le corresponderá el valor no disponible `NA_real_`. Sin embargo, los valores no disponibles de los vectores *dobles* serán mostrados en pantalla solamente con los caracteres `NA`.

Si deseas asegurarte de que los vectores *dobles* recibirán solamente valores no disponibles de tipo *doble*, puedes utilizar el valor `NA_real_` (en vez de la forma más simple `NA`) en las operaciones de asignación. Para más información puedes consultar más adelante la sección *Ejemplos*, así como la página de ayuda de los valores no disponibles.

5 Valor devuelto

Cualquier cifra tecleada sin comillas (`""`) en la consola de **R** será devuelta como un número de tipo *doble* por el lenguaje. Es decir, **R** podrá identificar *literales* numéricos en el código fuente y los devolverá como valores numéricos constantes de tipo *doble*.

R reconocerá literales numéricos en el sistema de numeración decimal y hexadecimal. Los valores en base decimal se escribirán tal cual, mientras que los valores en base hexadecimal deberán estar acompañados del prefijo `0x` o `0X` para indicar que se trata de números en esa base. Por ejemplo, la cifra `0x10` devolverá el valor *doble* `16` a partir de la representación hexadecimal ingresada. Por otro lado, la cifra en notación científica `1e3` devolverá el valor entero `1000` y será equivalente a haber ingresado el literal numérico `1000`.

El acceso a las celdas o los elementos de un vector *doble* se realizará por medio de las operaciones de indización (`[ ]`) de sus elementos tal como, por ejemplo: `un_vector[i]`; en donde `i` será el número índice del elemento que se desea extraer o almacenar.

La función `double()` creará un vector *doble* con el número de elementos especificado en el argumento de longitud, `length=`. Al momento de su creación, cada elemento del vector será igual a cero (`0`). Luego, se podrán asignar valores reales positivos o negativos, así como valores no disponibles (`NA`), al vector recientemente creado.

La función `double()` creará un vector doble con el número de elementos especificado en el argumento de longitud, `length=`. Al momento de su creación, cada elemento del vector será igual a cero (`0`). Luego, se podrán asignar valores de números reales, positivos o negativos, así como valores no disponibles (`NA_real_`), y valores numéricos especiales como el cero con signo (`+0`, `-0`).

Además, los vectores *dobles* también podrán almacenar valores no numéricos (`NaN`) derivados de operaciones matemáticas no definidas —como la división entre cero— , el infinito positivo (`+Inf`, `Inf`) y el infinito negativo (`-Inf`).

La función `as.double()` intentará coaccionar al objeto referido en el argumento `x=` al tipo *doble* y, en caso de tener éxito, devolverá al objeto como un vector de este tipo (`"double"`). Si la coacción no ha sido exitosa, el resultado será un valor no disponible (`NA`).

Las cadenas de caracteres que contengan representaciones de números del sistema decimal o hexadecimal (las cuales comienzan con `0x` o `0X`) entre espacios en blanco iniciales o finales se podrán convertir a valores *dobles*. No obstante, cada cadena de caracteres deberá contener una sola representación numérica sin espacios intercalados, de lo contrario esos elementos serán coaccionados como valores no disponibles (`NA`).

Además, las cadenas de caracteres que contengan el nombre de valores numéricos especiales de **R** como, por ejemplo, `"NA"`, `"NaN"`, `"Inf"` e incluso, `"infinity"`, serán convertidas a sus respectivos valores especiales.

`as.double()` eliminará los atributos, incluidos los nombres, de los objetos coaccionados, tal como lo hace la función `as.vector()`. Para asegurarte de que un objeto `x` permanezca con el tipo *doble* sin perder sus atributos, podrás asignar a un vector la etiqueta del tipo *doble* (`"double"`) con la función `storage.mode()`, por ejemplo, como en: `storage.mode(x) <- "integer"`. Esta forma de coacción hacia el tipo *doble* tiene la ventaja de modificar el tipo de almacenamiento sin eliminar los atributos del objeto.

La función `as.integer()` intentará coaccionar al objeto referido en el argumento `x=` al tipo *entero* y lo devolverá como un vector de este tipo. Si la coacción no ha sido exitosa, el resultado será un valor no disponible (`NA`).

Cuando se intente coaccionar un valor al tipo *entero*, o se ingrese un literal entero en la consola, y éste sea mayor o menor a los límites del intervalo de almacenamiento, se devolverá un valor no disponible (`NA`). Este comportamiento es diferente al de *S*, que devolvía un valor del mismo signo igual al límite del intervalo de almacenamiento.

Cuando `as.integer()` reciba valores con números reales, las fracciones decimales de estos valores serán truncados hacia el cero siempre y cuando se encuentren dentro del intervalo de almacenamiento entero. Esto significa que `as.integer(x)` funcionará igual que `trunc(x)` en estos casos. Del mismo modo, las partes imaginarias de los valores de números complejos se descartarán con una advertencia.

`as.integer()` podrá convertir en valores enteros a aquellas cadenas de caracteres (es decir, literales entre comillas) que contengan representaciones de números del sistema decimal o hexadecimal entre espacios en blanco iniciales o finales. No obstante, cada cadena de caracteres deberá contener una sola cifra, sin espacios intercalados, de lo contrario esos elementos serán coaccionados en valores no disponibles (`NA`). Algunas plataformas de sistema operativo podrían aceptar la coacción de cadenas de caracteres que representen números en otros sistemas de numeración diferentes al decimal o al hexadecimal, como el binario o el octal.

`as.integer()` eliminará los atributos, incluidos los nombres, de los objetos coaccionados, tal como lo hace la función `as.vector()`. Para asegurarte de que un objeto `x` permanezca con el tipo *entero* sin perder sus atributos, podrás asignar a un vector la etiqueta del tipo *entero* (`"integer"`) con la función `storage.mode()`, por ejemplo, como en: `storage.mode(x) <- "integer"`. Esta forma de coacción hacia el tipo *entero* tiene la ventaja de modificar el tipo de almacenamiento sin eliminar los atributos del objeto.

La función `c()` devolverá un vector *entero* si se utiliza para combinar valores enteros que estén separados por comas, siempre que al final de cada valor numérico se añada, sin mediar espacio, la letra ele mayúscula (`L`), por ejemplo: `c(1L, 2L, 3L)`. El resultado de la combinación de elementos enteros creará un vector de tipo *entero* si ninguno de los valores combinados tiene valores decimales. Para mayor información sobre la combinación de elementos para crear vectores de un determinado tipo puedes ver la página de ayuda de la función `c()`.

La función `is.integer()` devolverá el valor lógico verdadero (`TRUE`) o falso (`FALSE`) dependiendo de si el objeto referido en el argumento `x=` es de tipo *entero*, es decir, de si el vector tiene asociada internamente la etiqueta `"integer"`. Toma en cuenta que esta función sólo verificará el tipo de almacenamiento específico del vector, no si los objetos contenidos en él son, matemáticamente, números enteros; para más información puedes ver las secciones *Nota* y *Ejemplos* en esta página de ayuda. En el caso de los factores, que asocian números enteros a valores categóricos, `is.integer()` devolverá el valor lógico falso (`FALSE`) al momento de verificar el tipo del objeto.

6 <Sección adicional>

En computación, la *precisión* de una cifra alude al número de dígitos significativos que pueden ser desplegados en pantalla, independientemente de su exactitud. Por su parte, la *exactitud* de un formato de almacenamiento se refiere a la correspondencia entre la cifra almacenada internamente y la desplegada en la pantalla del ordenador. Para más información al respecto, puedes ver más adelante la sección *El tipo de almacenamiento doble*.

7 Nota

8 Referencias

9 También véase

10 Ejemplos

11 Código fuente