

## 1. Funciones de redondeo

Existe cierta confusión sobre el funcionamiento de las distintas operaciones de conversión de números no enteros a enteros disponibles en algunos lenguajes de programación. En este documento se trata de clarificar la situación e introducir algunas subrutinas útiles para la programación.

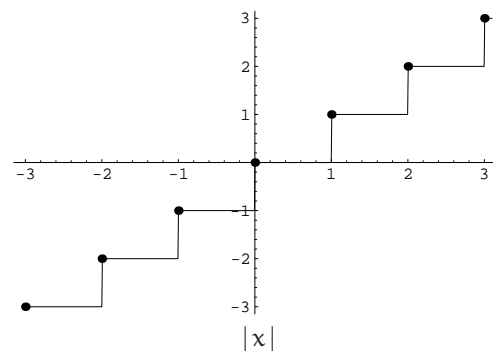
Veremos en primer lugar las funciones matemáticas fundamentales que convierten números reales en enteros, y las distintas funciones disponibles para implementarlas en una selección de lenguajes de programación. Los lenguajes que consideraremos son:

<i>C/C++:</i>	Funciones de las bibliotecas estándar
<i>Ruby:</i>	Módulos de la biblioteca estándar
<i>Python:</i>	Módulos de la biblioteca estándar
<i>ECMAScript:</i>	Objetos nativos del estándar de JavaScript
<i>Java:</i>	Clases del módulo <code>java.lang</code>
<i>C#:</i>	Bibliotecas del sistema .NET
<i>Mathematica:</i>	Funciones incorporadas del sistema
<i>RPL:</i>	Lenguaje User-RPL de las calculadoras HP (28, 48, 49)
<i>Visual Basic:</i>	Funciones de Microsoft VB 6, VBA, VBScript
<i>Fortran:</i>	Funciones intrínsecas de Fortran77
<i>Pascal:</i>	Rutinas estándar de Object Pascal (Delphi/Kylix)

### 1.1. Entero inferior

La función *floor*,  $\lfloor x \rfloor$ , a veces denotada  $[x]$ , es el mayor entero menor o igual que  $x$ . También es conocida por el nombre francés *entier*.

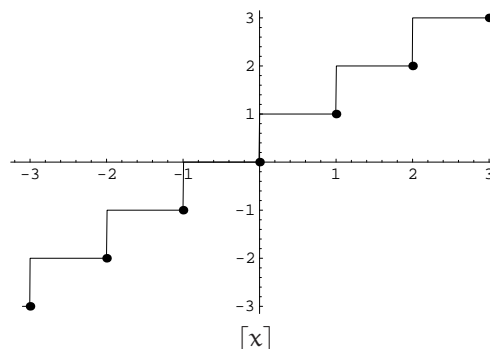
<i>C/C++:</i>	<code>floor(x)</code>
<i>Ruby:</i>	<code>x.floor</code>
<i>Python:</i>	<code>math.floor(x)</code>
<i>ECMAScript:</i>	<code>Math.floor(x)</code>
<i>Java:</i>	<code>java.lang.Math.floor(x)</code>
<i>C# (.NET):</i>	<code>System.Math.Floor(x)</code>
<i>Mathematica:</i>	<code>Floor[x]</code>
<i>RPL:</i>	<code>x FLOOR</code>
<i>Visual Basic:</i>	<code>Int(x)</code>



### 1.2. Entero superior

La función *ceiling*,  $\lceil x \rceil = -\lfloor -x \rfloor$ , es el menor entero mayor o igual que  $x$ .

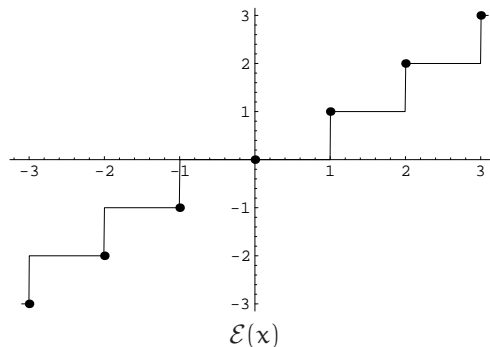
<i>C/C++:</i>	<code>ceil(x)</code>
<i>Ruby:</i>	<code>x.ceil</code>
<i>Python:</i>	<code>math.ceil(x)</code>
<i>ECMAScript:</i>	<code>Math.ceil(x)</code>
<i>Java:</i>	<code>java.lang.Math.ceil(x)</code>
<i>C# (.NET):</i>	<code>System.Math.Ceiling(x)</code>
<i>Mathematica:</i>	<code>Ceiling[x]</code>
<i>RPL:</i>	<code>x CEIL</code>
<i>Visual Basic:</i>	<code>-Int(-x)</code>



### 1.3. Parte entera

Ésta es la función que devuelve la parte entera de un número:  $\mathcal{E}(x) = \begin{cases} \lfloor x \rfloor & \text{si } x \geq 0 \\ \lceil x \rceil & \text{si } x < 0 \end{cases}$

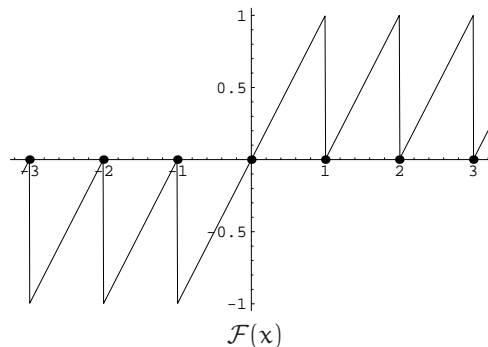
*Ruby:* `x.truncate = x.to_i`  
*Mathematica:* `IntegerPart[x]`  
*RPL:* `x IP`  
*Visual Basic:* `Fix(x)`  
*Pascal:* `Trunc(x)`  
*Fortran:* `AINT(x)`



#### 1.3.1. Parte fraccionaria

La parte fraccionaria es  $\mathcal{F}(x) = x - \mathcal{E}(x)$ .

*Mathematica:* `FractionalPart[x]`  
*RPL:* `x FP`  
*Visual Basic:* `x-Fix(x)`  
*Pascal:* `Frac(x)`



### 1.4. Redondeo

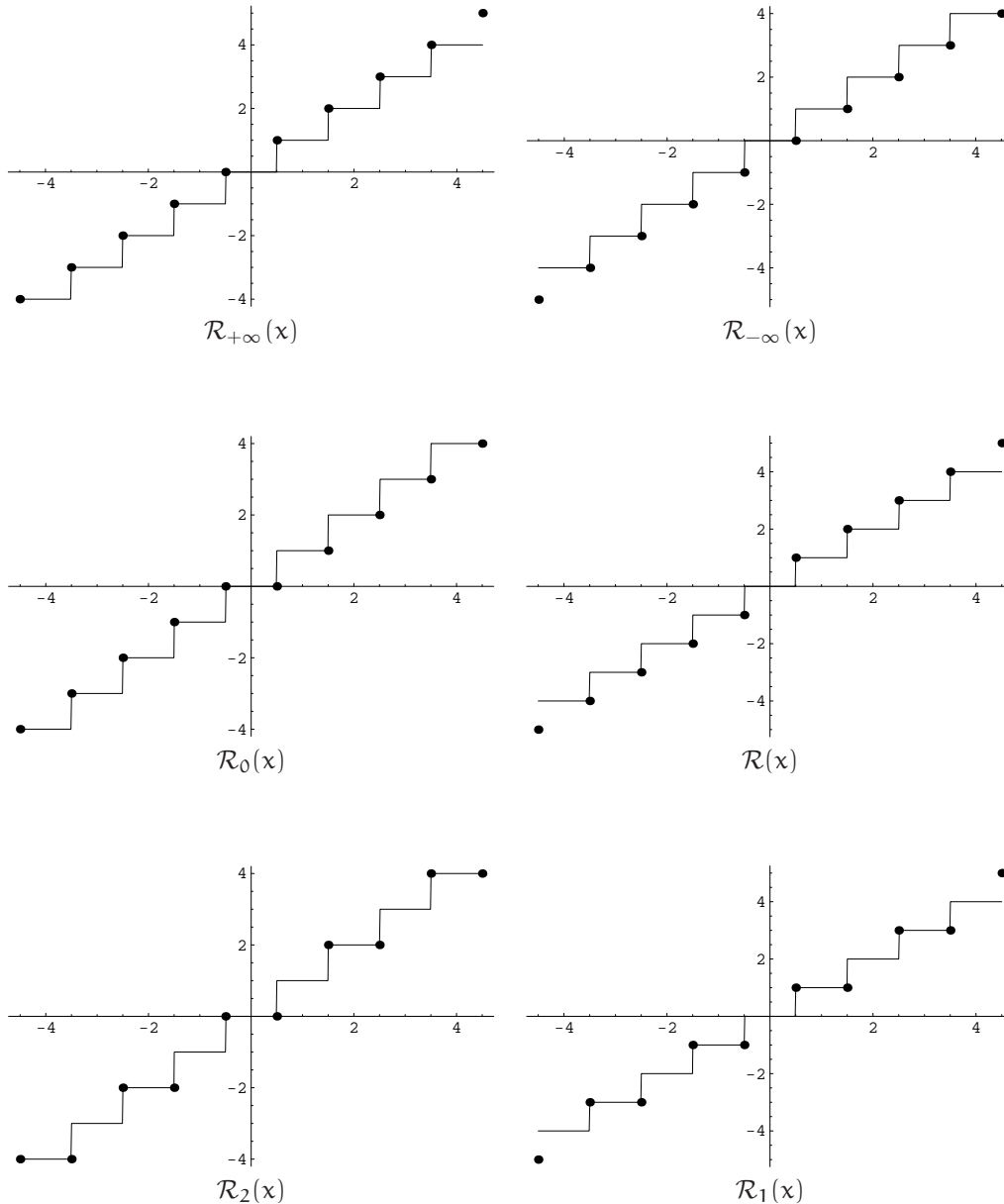
Podemos definir diferentes funciones para calcular el entero más cercano a un número según cuál sea el tratamiento de los números el conjunto  $\mathcal{M} = \{x \mid |\mathcal{F}(x)| = 1/2\} = \{x \mid x + 1/2 \in \mathbb{Z}\}$ , ya que éstos número son equidistantes de los enteros inferior y superior más cercanos.

$$\begin{aligned} \mathcal{R}_{+\infty}(x) &= \lfloor x + 1/2 \rfloor \\ \mathcal{R}_{-\infty}(x) &= \lceil x - 1/2 \rceil \\ \mathcal{R}_0(x) &= \begin{cases} \lfloor x - 1/2 \rfloor & \text{si } x \geq 0 \\ \lceil x + 1/2 \rceil & \text{si } x < 0 \end{cases} \\ \mathcal{R}(x) \equiv \mathcal{R}_\infty(x) &= \begin{cases} \lfloor x + 1/2 \rfloor = \mathcal{E}(x + 1/2) & \text{si } x \geq 0 \\ \lceil x - 1/2 \rceil = \mathcal{E}(x - 1/2) & \text{si } x < 0 \end{cases} \end{aligned}$$

Para evitar sesgos en el tratamiento de los números de  $\mathcal{M}$ , se suele usar en la práctica la regla de redondeo conocida como "bancaria" o "gaussiana",  $\mathcal{R}_2$ , según la cual se redondean siempre los valores de  $\mathcal{M}$  a números pares. De forma simétrica podemos definir una regla  $\mathcal{R}_1$  que redondee  $\mathcal{M}$  a números impares:

$$\begin{aligned} \mathcal{R}_2(x) &= \begin{cases} 2 \lfloor \frac{x+1/2}{2} \rfloor & \text{si } x \in \mathcal{M} \\ \mathcal{R}(x) = \mathcal{R}_0(x) = \mathcal{R}_{+\infty} = \mathcal{R}_{-\infty} & \text{si } x \notin \mathcal{M} \end{cases} \\ \mathcal{R}_1(x) &= \begin{cases} 2 \lfloor \frac{x-1/2}{2} \rfloor + 1 & \text{si } x \in \mathcal{M} \\ \mathcal{R}(x) = \mathcal{R}_0(x) = \mathcal{R}_{+\infty} = \mathcal{R}_{-\infty} & \text{si } x \notin \mathcal{M} \end{cases} \end{aligned}$$

*Ruby:*  $x.\text{round} = \mathcal{R}(x)$   
*Python:*  $\text{round}(x) = \mathcal{R}(x)$   
*ECMAScript:*  $\text{Math.round}(x) = \mathcal{R}(x)$   
*Java:*  $\text{java.lang.Math.round}(x) = \mathcal{R}_{+\infty}(x)$   
*C# (.NET):*  $\text{System.Math.Round}(x) = \mathcal{R}_2(x)$   
*Mathematica:*  $\text{Round}[x] = \mathcal{R}_2(x)$   
*RPL (HP48,HP49):*  $x \text{ 0 RND} = \mathcal{R}(x)$   
*Visual Basic:*  $\text{Round}(x) = \mathcal{R}_2(x)$   
*Fortran:*  $\text{ANINT}(x) = \mathcal{R}(x)$   
*Pascal:*  $\text{Round}(x) = \mathcal{R}_2(x)$



#### 1.4.1. Redondeo en un dígito determinado

Para redondear con  $n$  dígitos decimales podemos calcular  $\mathcal{R}_n(x) = \mathcal{R}(10^n x)/10^n$ , y análogamente,  $\mathcal{R}_{2,n}(x)$ ,  $\mathcal{R}_{0,n}(x)$ ,  $\mathcal{R}_{-\infty,n}(x)$  y  $\mathcal{R}_{+\infty,n}(x)$ .

*Python:*  $\text{round}(x, n) = \mathcal{R}_n(x)$   
*C# (.NET):*  $\text{System.Math.Round}(x, n) = \mathcal{R}_{2,n}(x)$   
*Mathematica:*  $\text{Round}[x*10^n]/10^n = \mathcal{R}_{2,n}(x)$   
*RPL (HP48,HP49):*  $x \ n \ \text{RND} = \mathcal{R}_n(x)$   
*Visual Basic:*  $\text{Round}(x, n) = \mathcal{R}_{2,n}(x)$

También puede ser necesario redondear con  $m$  dígitos de *precisión*. En este caso se deben contar estos dígitos entre los más significativos de la cifra, en lugar de entre los decimales. Esto se puede lograr sencillamente aplicando el redondeo con  $n$  dígitos decimales, siendo  $n = m - 1 - \lfloor \lg |x| \rfloor$ . En algunos lenguajes, como RPL, esta característica está disponible empleando un número de dígitos negativos,  $-m$ .

Igualmente podríamos redondear en un dígito determinado de la expresión del valor en una base arbitraria  $b$ :  $\mathcal{R}_n^b(x) = \mathcal{R}(b^n x)/b^n$ , y análogamente,  $\mathcal{R}_{2,n}^b(x)$ ,  $\mathcal{R}_{0,n}^b(x)$ ,  $\mathcal{R}_{-\infty,n}^b(x)$  y  $\mathcal{R}_{+\infty,n}^b(x)$ .

## 2. Programación

Definiremos aquí algunas funciones de redondeo no presentes en algunos lenguajes. Podemos calcular cualquiera de las funciones más interesantes de redondeo disponiendo únicamente de la función  $\lfloor x \rfloor$ :

- $\lceil x \rceil = -\lfloor -x \rfloor$
- $\mathcal{E}(x) = \begin{cases} \lfloor x \rfloor & \text{si } x \geq 0 \\ \lceil x \rceil & \text{si } x < 0 \end{cases}$
- $\mathcal{F}(x) = x - \mathcal{E}(x)$
- $\mathcal{R}_{+\infty}(x) = \lfloor x + 1/2 \rfloor$
- $\mathcal{R}_{-\infty}(x) = \lceil x - 1/2 \rceil$
- $\mathcal{R}_0(x) = \begin{cases} \lceil x - 1/2 \rceil & \text{si } x \geq 0 \\ \lfloor x + 1/2 \rfloor & \text{si } x < 0 \end{cases}$
- $\mathcal{R}(x) = \begin{cases} \lfloor x + 1/2 \rfloor & \text{si } x \geq 0 \\ \lceil x - 1/2 \rceil & \text{si } x < 0 \end{cases}$
- $\mathcal{M} = \{x \mid \lfloor x \rfloor + 1/2 = x\}$
- $\mathcal{R}_2(x) = \begin{cases} 2\lfloor \frac{x+1/2}{2} \rfloor & \text{si } x \in \mathcal{M} \\ \lfloor x + 1/2 \rfloor & \text{si } x \notin \mathcal{M} \end{cases}$

Si no se dispone de la función  $\lfloor x \rfloor$ , se puede calcular esta, (y por tanto todas las demás), empleando  $\mathcal{E}(x)$  o en general cualquier función  $f(x)$  tal que  $\forall x \geq 0, f(x) = \lfloor x \rfloor$

$$\lfloor x \rfloor = \begin{cases} f(x) & \text{si } x \geq 0 \\ f(f(-x) + 1 + x) - f(-x) - 1 & \text{si } x < 0 \end{cases}$$

Para cualquiera de las funciones  $f$  de redondeo podemos calcular una función  $f_n$  que aplique el redondeo al dígito decimal  $n$ , y, con más generalidad, una función  $f_{\langle k \rangle}$  que redondee a un múltiplo de  $k$ . (tendremos que  $f_n(x) = f_{\langle 10^{-n} \rangle}(x)$ ).

- $f_n(x) = f(10^n x)/10^n$
- $f_{\langle k \rangle}(x) = kf(x/k)$

También podemos definir una función  $f^m$  que aplique el redondeo  $f$  al dígito significativo  $m$  así:

- $f^m(x) = f_{m-1-\lfloor \lg |x| \rfloor}(x)$

## 2.1. Visual Basic

El lenguaje Visual Basic carece de una función de redondeo hacia infinito  $\mathcal{R}(x)$ . Definiremos una función `RoundI` para ello.

El valor que deseamos calcular es  $\text{Fix}(x \cdot 10^d + 0.5 \cdot \text{Sgn}(x)) / 10^d$ , pero trataremos de lograr una mayor eficiencia:

```
"vb/roundi.bas" 5a ≡
Public Function RoundI(ByVal x As Double, Optional ByVal d As Integer = 0) As Double
    Dim m As Double
    m = 10^d
    If x < 0 Then
        RoundI = Fix(x*m - 0.5) / m
    Else
        RoundI = Fix(x*m + 0.5) / m
    End If
End Function
◇
```

Nota: Muchos programas emplean la expresión `Int(x+0.5)` para realizar redondeos en Visual Basic; esto equivale a la función  $\mathcal{R}_{+\infty}(x)$ .

## 2.2. C++

Las siguientes funciones están escritas en C++, pero es muy sencillo traducirlas a C.

```
"c/round.h" 5b ≡
#ifndef ROUNDING_H
#define ROUNDING_H
namespace rounding {
    <Declaraciones de funciones de redondeo 5d, ... >
};
#endif
◇
```

```
"c/round.cpp" 5c ≡
#include "round.h"
namespace rounding {
    <Definiciones de funciones de redondeo 5e, ... >
};
◇
```

Definiremos dos funciones en C++ para calcular  $\mathcal{E}(x)$  y  $\mathcal{F}(x)$ .

```
<Declaraciones de funciones de redondeo 5d> ≡
double integer_part(double);
double fractional_part(double);
◇
```

Fragmento definido en [5d](#), [6bd](#).

Fragmento usado en [5b](#).

Ésta es la función  $\mathcal{E}(x)$ , que equivale a `(x<0) ? std::ceil(x) : std::floor(x)`.

```
<Definiciones de funciones de redondeo 5e> ≡
double integer_part(double x) // (x<0) ? std::ceil(x) : std::floor(x)
{
    static double y;
    std::modf(x, &y);
    return y;
}
◇
```

Fragmento definido en [5e](#), [6ace](#).

Fragmento usado en [5c](#).

Y ésta,  $\mathcal{F}(x)$ , y equivale a `x-integer_part(x)`.

```

<Definiciones de funciones de redondeo 6a> ≡
double fractional_part(double x) // x-integer_part(x)
{
    static double y;
    return std::modf(x, &y);
}
◇

```

Fragmento definido en [5e](#), [6ace](#).  
Fragmento usado en [5c](#).

Las bibliotecas estándar de C y C++ carecen de funciones de redondeo. Definiremos aquí las más usadas.  
En primer lugar,  $\mathcal{R}(x)$ .

```

<Declaraciones de funciones de redondeo 6b> ≡
double round_inf(double x, int d=0);
◇

```

Fragmento definido en [5d](#), [6bd](#).  
Fragmento usado en [5b](#).

```

<Definiciones de funciones de redondeo 6c> ≡
double round_inf(double x, int d)
{
    if (d==0)
        return x < 0.0 ? std::ceil(x-0.5) : std::floor(x+0.5);
    double m = std::pow(10.0,d);
    return x<0.0 ? std::ceil(x*m-0.5)/m : std::floor(x*m-0.5)/m;
}
◇

```

Fragmento definido en [5e](#), [6ace](#).  
Fragmento usado en [5c](#).

Implementaremos también el redondeo bancario  $\mathcal{R}_2(x) = \begin{cases} 2\lfloor \frac{x+1/2}{2} \rfloor & \text{si } x \in \mathcal{M} \\ \lfloor x + 1/2 \rfloor & \text{si } x \notin \mathcal{M} \end{cases}$

```

<Declaraciones de funciones de redondeo 6d> ≡
double round_unbiased(double x, int d=0);
◇

```

Fragmento definido en [5d](#), [6bd](#).  
Fragmento usado en [5b](#).

```

<Definiciones de funciones de redondeo 6e> ≡
double round_unbiased(double x, int d)
{
    double m = pow(10.0,d);
    x *= m;
    x += 0.5;
    x = (fractional_part(x)==0.0) ? 2.0*std::floor(x*0.5) : std::floor(x);
    return x/m;
}
◇

```

Fragmento definido en [5e](#), [6ace](#).  
Fragmento usado en [5c](#).

También podríamos haber usado únicamente la función `floor` así:

```

⟨Alternativa sin fractional_part 7a⟩ ≡
double round_unbiased(double x, int d)
{
    double m = pow(10.0, d);
    x *= m;
    x += 0.5;
    double y = std::floor(x);
    x = (x==y) ? 2.0*std::floor(x*0.5) : y;
    return y/m;
}
◇

```

Fragmento no usado.

## 2.3. Mathematica

Las funciones  $\mathcal{R}(x)$ ,  $\mathcal{R}_{+\infty}(x)$ ,  $\mathcal{R}_{-\infty}(x)$ ,  $\mathcal{R}_0(x)$  y  $\mathcal{R}_1(x)$  no están presentes en Mathematica, y las definiremos aquí.

La función `RoundUp[x]` implementa  $\mathcal{R}_{+\infty}(x)$ :

```

"math/round.m" 7b ≡
RoundUp[x_] := Floor[x + 0.5]
◇

```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

La función `RoundUp[x]` implementa  $\mathcal{R}_{-\infty}(x)$ :

```

"math/round.m" 7c ≡
RoundDn[x_] := Ceiling[x - 0.5]
◇

```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

La función `RoundZero[x]` implementa  $\mathcal{R}_0(x)$ :

```

"math/round.m" 7d ≡
RoundZero[x_] := If[x < 0, RoundUp[x], RoundDn[x]]
◇

```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

La función `RoundInf[x]` implementa  $\mathcal{R}(x)$ :

```

"math/round.m" 7e ≡
RoundInf[x_] := If[x < 0, RoundDn[x], RoundUp[x]]
◇

```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

La función `RoundOdd[x]` implementa  $\mathcal{R}_1(x)$ :

```

"math/round.m" 7f ≡
RoundOdd[x_] := If[Floor[x]+0.5==x, 2Floor[(x-0.5)*0.5]+1, Floor[x+0.5]]
◇

```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

Definiremos aquí también las subrutinas empleadas para generar las ilustraciones de este documento.

En primer lugar, una función `Fplot` para generar el gráfico de una función entre dos valores de su variable libre:

```

"math/round.m" 7g ≡
Fplot[f_, x1_, x2_] := Plot[f[x], {x, x1, x2}]
◇

```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

Y otra función `Pplot` para marcar una serie de puntos discretos de esa función. Los puntos se tomarán en los valores enteros entre  $x_1$  y  $x_2$ , pero desplazados en una cantidad  $d$ . De esta forma podremos marcar los puntos del conjunto  $\mathcal{M}$ , haciendo  $d=1/2$ .

```
"math/round.m" 8a ≡
  Pplot[f_, x1_, x2_, d_] := ListPlot[Table[{x + d, f[x + d]}, {x, x1, x2}]]
  ◇
```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

Por último, la función `ShowF` nos permitirá combinar ambos gráficos seleccionando un tamaño adecuado para los puntos. Los argumentos de este procedimiento serán la función  $f$  a visualizar, los extremos para los valores de la función y los extremos y desplazamiento para los puntos discretos.

```
"math/round.m" 8b ≡
  ShowF[f_, x1_, x2_, n1_, n2_, d_] :=
    Show[Fplot[f, x1, x2], Pplot[f, n1, n2, d],
    Prolog -> AbsolutePointSize[5]]
  ◇
```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

Los gráficos que aparecen en este documento han sido generados con las siguientes instrucciones:

- Función  $\lfloor x \rfloor$ :

```
"math/round.m" 8c ≡
  ShowF[Floor, -3, 3.1, -3, 3, 0]
  ◇
```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

- Función  $\lceil x \rceil$ :

```
"math/round.m" 8d ≡
  ShowF[Ceiling, -3.1, 3, -3, 3, 0]
  ◇
```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

- Función  $\mathcal{E}(x)$ :

```
"math/round.m" 8e ≡
  ShowF[IntegerPart, -3.1, 3.1, -3, 3, 0]
  ◇
```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

- Función  $\mathcal{F}(x)$ :

```
"math/round.m" 8f ≡
  ShowF[FractionalPart, -3.1, 3.2, -3, 3, 0]
  ◇
```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

- Función  $\mathcal{R}_2(x)$ :

```
"math/round.m" 8g ≡
  ShowF[Round, -4.5, 4.5, -5, 4, 0.5]
  ◇
```

Archivo definido en [7bcdefg](#), [8abcdefg](#), [9abcde](#).

- Función  $\mathcal{R}_1(x)$ :

```
"math/round.m" 9a ≡
  ShowF[RoundOdd, -4.5, 4.5, -5, 4, 0.5]
  ◇
Archivo definido en 7bcdefg, 8abcdefg, 9abcde.
```

- Función  $\mathcal{R}_{+\infty}(x)$ :

```
"math/round.m" 9b ≡
  ShowF[RoundUp, -4.5, 4.5, -5, 4, 0.5]
  ◇
Archivo definido en 7bcdefg, 8abcdefg, 9abcde.
```

- Función  $\mathcal{R}_{-\infty}(x)$ :

```
"math/round.m" 9c ≡
  ShowF[RoundDn, -4.5, 4.5, -5, 4, 0.5]
  ◇
Archivo definido en 7bcdefg, 8abcdefg, 9abcde.
```

- Función  $\mathcal{R}_0(x)$ :

```
"math/round.m" 9d ≡
  ShowF[RoundZero, -4.5, 4.5, -5, 4, 0.5]
  ◇
Archivo definido en 7bcdefg, 8abcdefg, 9abcde.
```

- Función  $\mathcal{R}(x)$ :

```
"math/round.m" 9e ≡
  ShowF[RoundInf, -4.5, 4.5, -5, 4, 0.5]
  ◇
Archivo definido en 7bcdefg, 8abcdefg, 9abcde.
```

## 2.4. RPL

La versión de RPL referida aquí es la de las calculadoras de las series 48 y 49, que tienen una función RND con dos argumentos: el número a redondear y la precisión. La serie 28 tenía una instrucción RND con un único argumento (el número a redondear) que redondeaba en base al modo numérico actual (STD, FIX, ENG).

Incluiremos los programas de RPL descritos aquí en un directorio:

```
"round.rpl" 9f ≡
  %%HP: T(3)A(D)F(.);
  DIR
  <Objetos RPL 10a,...>
  END
  ◇
```

Este primer programa calcula el redondeo  $\mathcal{R}_2(x) = \begin{cases} 2\lfloor \frac{x+1/2}{2} \rfloor & \text{si } x \in \mathcal{M} \\ \lfloor x + 1/2 \rfloor & \text{si } x \notin \mathcal{M} \end{cases}$

```

⟨ Objetos RPL 10a ⟩ ≡
  RNDI0
  \<<
  .5 +
  IF DUP FP 0 == THEN
    .5 * FLOOR 2 *
  ELSE
    FLOOR
  END
  \>>
  ◇

```

Fragmento definido en [10abc](#).

Fragmento usado en [9f](#).

Si queremos redondear a un número de dígitos dado, podemos emplear el siguiente programa, introduciendo en la pila en primer lugar el número a redondear y en segundo lugar el número de decimales.

```

⟨ Objetos RPL 10b ⟩ ≡
  RNDIN
  \<<
  ALOG SWAP OVER *
  RNDI0
  SWAP /
  \>>
  ◇

```

Fragmento definido en [10abc](#).

Fragmento usado en [9f](#).

La instrucción RND de RPL puede redondear además con un número dado de dígitos de *precisión*, en lugar de dígitos *decimales*. Esto lo hace cuando el número de dígitos establecido es negativo. Realizaremos un programa RNDI que haga lo mismo, pero con el redondeo  $\mathcal{R}_2$ .

```

⟨ Objetos RPL 10c ⟩ ≡
  RNDI
  \<<
  IF DUP 0 < THEN
    NEG OVER ABS LOG FLOOR 1 + -
  END
  ALOG SWAP OVER *
  RNDI0
  SWAP /
  \>>
  ◇

```

Fragmento definido en [10abc](#).

Fragmento usado en [9f](#).

Las calculadoras RPL (HP28, HP48 y HP49) redondean internamente los resultados de las operaciones aritméticas en coma flotante mediante  $\mathcal{R}_2^{12}(x)$  (a 12 dígitos significativos). El siguiente programa usa esta característica para implementar  $\mathcal{R}_{2,n}(x)$

```

\<<
11 SWAP - ALOG      OVER SIGN *      DUP ROT + SWAP -
\>>

```

Por último, este programa reproduce la función RND de HP48 y HP49, y puede ser útil como comando unificado que también funciona en HP28.

```
"round28.rpl" 11 ≡
%%HP: T(3)A(D)F(.);
ROUND
\<<
  IF DUP 0 < THEN
    NEG OVER ABS LOG FLOOR 1 + -
  END
  ALOG SWAP OVER *
  IF DUP 0 < THEN
    .5 - CEIL
  ELSE
    .5 + FLOOR
  END
  SWAP /
\>>
◇
```

#### 2.4.1. Redondeo interno en calculadoras HP

Las calculadoras de 10 dígitos como la HP-35, HP-15c, HP-16c usan un redondeo entero del tipo  $\mathcal{R}(x)$ . En este ejemplo el número 1000000000,5 se redondea a 1000000001:

```
1 EEX 9 0.5 + -> 1000000001
```

La representación en pantalla en los modos FIX, SCI, ENG cuando requiere redondeo utiliza también una función del tipo  $\mathcal{R}_n(x)$ . En el caso de la función RND de la 15c se utiliza el mismo redondeo usado en la visualización.

Las calculadoras HP de 12 dígitos, como las series 28, 48, 49, 50 (RPL), las 32s, 42s (RPL internamente) o las 33s, 35s, usan un redondeo interno  $\mathcal{R}_2(x)$  de forma que 10000000000,5 se redondea a 10000000000 y 10000000001,5 a 10000000002:

```
1 EEX 11 0.5 + -> 100000000000
1 EEX 11 1 + 0.5 + -> 100000000002
```

Pero en estas mismas calculadoras el redondeo usado para los modos de visualización FIX, SCI, ENG y para la función RND es del tipo  $\mathcal{R}_n(x)$ .

Las calculadoras de las series 48 y 49 usan internamente números de 15 dígitos (*reales extendidos*) que no son redondeados sino truncados (además el resultado de algunas funciones internas es inexacto hasta en los dos últimos dígitos). Los resultados finales se convierten al formato normal (*real*) de 12 dígitos mediante redondeo  $\mathcal{R}_2$ .

Los números de precisión extendida pueden emplearse programando en SysRPL o mediante la biblioteca XREAL (1005). Estos números son útiles para realizar cálculos y almacenar los resultados intermedios, de forma similar a los registros de 80 bits de la implementación de coma flotante IEEE de los procesadores INTEL. Por ejemplo podríamos programar una función para logaritmos en base arbitraria usando XREAL así:

```
\<< R~X SWAP R~X XLN SWAP XLN XDIV R~X \>>
```

Obteniendo una mayor precisión que mediante su equivalente LN SWAP LN SWAP /.

La calculadora aHP-15C cuenta con las operaciones:

- $x \text{ INT} = \mathcal{E}(x)$
- $x \text{ FRAC} = \mathcal{F}(x)$
- $x \text{ RND} = \mathcal{R}_n(x)$  con  $n$  según el modo de visualización actual.

La calculadora HP-35s tiene estas operaciones:

- $x \text{ IP} = \mathcal{E}(x)$
- $x \text{ FP} = \mathcal{F}(x)$
- $x \text{ RND} = \mathcal{R}_n(x)$  con  $n$  según el modo de visualización actual (notar que internamente en cambio se usa el redondeo  $\mathcal{R}_{2,15}(x)$ )

### 2.4.2. LONGFLOAT

También hay disponible una biblioteca para la serie 49, LONGFLOAT (902), que trabaja con números de precisión arbitraria (fijada por una variable `DIGITS`). Un número  $x$  se representa mediante un exponente decimal y una *mantisa* entera de  $d = \text{DIGITS}$  dígitos cuyo valor  $y$  es:

$$y = 10^d \frac{x}{10^{\lfloor \log |x| \rfloor + 1}}$$

Se cuenta con cuatro modos de “redondeo” que se aplican a  $y$  en función del estado de los indicadores 37 y 38 del sistema y son similares a las especificadas en IEEE 754:

- “al más cercano” ( $\mathcal{R}(y)$ ): -37 CF -38 CF
- “hacia abajo” ( $\lfloor y \rfloor$ ): -37 SF -38 CF
- “hacia arriba” ( $\lceil y \rceil$ ): -37 CF -38 SF
- “truncado” hacia cero  $\mathcal{E}(y)$ : -37 SF -38 SF

Pero este “redondeo” se hace teniendo únicamente en cuenta el dígito siguiente al dígito redondeado, por lo que sólo se realiza correctamente en el primer caso ( $\mathcal{R}(y)$ ).

La versión 3.5 cambia el redondeo “al más cercano” por redondeo a “par”  $\mathcal{R}_2(x)$ ; se sigue usando únicamente un dígito adicional. La versión 3.93 se distribuye en dos opciones,  $\mathcal{R}(y)$  y  $\mathcal{R}_2(x)$ , esta última con el defecto mencionado.

### 3. Otros lenguajes y sistemas

#### 3.1. SQL

El lenguaje Transact-SQL, o T-SQL es la versión de SQL implementada por Microsoft SQL Server (originalmente Sybase). Estas funciones no están en el estándar de SQL.

- $\text{FLOOR}(x) = \lfloor x \rfloor$
- $\text{CEILING}(x) = \lceil x \rceil$
- $\text{ROUND}(x, n) = \text{ROUND}(x, n, 0) = \mathcal{R}(10^n x)/10^n$
- $\text{ROUND}(x, n, 1) = \mathcal{E}(10^n x)/10^n$  (el tercer parámetro indica truncamiento)

Los dialectos SQL de PostgreSQL, MySQL y Oracle tienen estas mismas funciones, excepto la versión de `ROUND` de tres parámetros. Además la función `CEILING` anterior se puede usar con la denominación `CEIL`.

#### 3.2. MapBasic

El lenguaje MapBasic es el lenguaje del sistema MapInfo.

- $\text{Int}(x) = \lfloor x \rfloor$
- $\text{Fix}(x) = \mathcal{E}(x)$
- $\text{Round}(x, y) = \begin{cases} y\mathcal{R}_{+\infty}(x/y) & \text{si } y \neq 0 \\ x & \text{si } y = 0 \end{cases}$

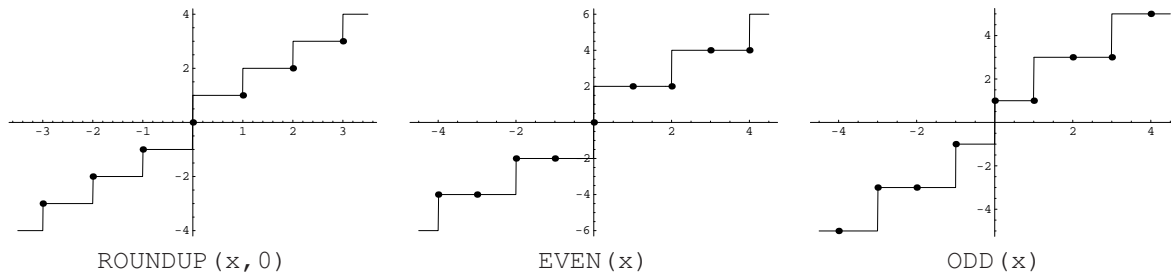
#### 3.3. Excel

Aquí nos referimos al lenguaje de fórmulas de Microsoft Excel, no al lenguaje de macros que incorpora, VBA (Visual Basic for Applications).

Los nombres de las funciones son distintos en versiones de Excel para diferentes idiomas. Se muestran los nombres en inglés y castellano.

- $\text{ROUND}(x, n) \equiv \text{REDONDEAR}(x, n) = \mathcal{R}_n(x)$
- $\text{ROUNDUP}(x, n) \equiv \text{REDONDEAR.MAS}(x, n) = \begin{cases} \lceil (10^n x) \rceil / 10^n & \text{si } x \geq 0 \\ \lfloor (10^n x) \rfloor / 10^n & \text{si } x < 0 \end{cases}$
- $\text{ROUNDDOWN}(x, n) \equiv \text{REDONDEAR.MENOS}(x, n) = \mathcal{E}(10^n x)/10^n = \begin{cases} \lfloor (10^n x) \rfloor / 10^n & \text{si } x \geq 0 \\ \lceil (10^n x) \rceil / 10^n & \text{si } x < 0 \end{cases}$
- $\text{EVEN}(x) \equiv \text{REDONDEA.PAR}(x) = 2 * \text{ROUNDUP}(x/2, 0) = \begin{cases} 2\lceil x/2 \rceil & \text{si } x \geq 0 \\ 2\lfloor x/2 \rfloor & \text{si } x < 0 \end{cases}$
- $\text{ODD}(x) \equiv \text{REDONDEA.IMPARG}(x) = \begin{cases} 2\lceil (x+1)/2 \rceil - 1 & \text{si } x \geq 0 \\ 2\lfloor (x-1)/2 \rfloor + 1 & \text{si } x < 0 \end{cases}$
- $\text{INT}(x) \equiv \text{ENTERO}(x) = \lfloor x \rfloor$
- $\text{TRUNC}(x, n) \equiv \text{TRUNCAR}(x, n) = \text{ROUNDDOWN}(x, n)$
- $\text{FLOOR}(x, y) \equiv \text{MULTIPLO.INFERIOR}(x, y) = y\lfloor (x/y) \rfloor$
- $\text{CEILING}(x, y) \equiv \text{MULTIPLO.SUPERIOR}(x, y) = y\lceil (x/y) \rceil$
- $\text{MROUND}(x, y) \equiv \text{REDOND.MULT}(x, y) = y * \text{ROUND}(x/y, 0) = y\mathcal{R}(x/y)$

Excel introduce tres funciones cuya forma no ha sido mostrada en la primera sección de este documento:



### 3.4. PHP

- $\text{floor}(x) = \lfloor x \rfloor$
- $\text{ceil}(x) = \lceil x \rceil$
- $\text{round}(x, n) == \mathcal{R}_n(x)$  —a partir de la versión 4
- $\text{round}(x) = \text{round}(x, 0) = \mathcal{R}(x)$

### 3.5. MatLab

- $\text{floor}(x) = \lfloor x \rfloor$
- $\text{ceil}(x) = \lceil x \rceil$
- $\text{fix}(x) = \mathcal{E}(x)$
- $\text{round}(x) = \mathcal{R}(x)$  —sin verificar; podría ser  $\mathcal{R}_2$

### 3.6. MathCAD

- $\text{round}(x, n) = \mathcal{R}_n(x)$  —en algunas versiones, (e.g. 8.00-8-02),  $\mathcal{R}_{2,n}(x)$
- $\text{round}(x) = \text{round}(x, 0)$
- $\text{floor}(x) = \lfloor x \rfloor$
- $\text{ceil}(x) = \lceil x \rceil$
- $\text{trunc}(x) = \mathcal{E}(x)$
- $\text{mantissa}(x) = x - \text{floor}(x)$
- $\text{round}(x) = \text{round}(x, 0) = \mathcal{R}(x)$

### 3.7. Maple

- $\text{floor}(x) = \lfloor x \rfloor$
- $\text{ceil}(x) = \lceil x \rceil$
- $\text{trunc}(x) = \mathcal{E}(x)$
- $\text{frac}(x) = \mathcal{F}(x)$
- $\text{round}(x) = \mathcal{R}(x)$

### 3.8. Scheme

- $(\text{round } x) = \mathcal{R}_2(x)$  —algunas implementaciones no estándar, como KSM, usan  $\mathcal{R}(x)$
- $(\text{floor } x) = \lfloor x \rfloor$
- $(\text{ceiling } x) = \lceil x \rceil$
- $(\text{truncate } x) = \mathcal{E}(x)$

#### 3.8.1. Common Lisp

De forma similar a como ocurre en Fortran con el prefijo *A*, (*AIN*,*ANINT* vs. *INT*,*NINT*), Las siguientes funciones con el prefijo *f* devuelven valores en coma flotante, mientras que las funciones sin este prefijo devuelven valores de tipo entero.

- $(\text{fround } x) = \mathcal{R}(x)$  —sin verificar
- $(\text{ffloor } x) = \lfloor x \rfloor$
- $(\text{fceiling } x) = \lceil x \rceil$
- $(\text{ftruncate } x) = \mathcal{E}(x)$

### 3.9. Perl

- $\text{int}(x) = \mathcal{E}(x)$

Podemos emplear calcular  $\mathcal{R}(x) = \text{int}(x + .5 * (x <=> 0))$ . Por ejemplo:

```
"round.perl" 15 ≡
sub round {
    my($number) = shift;
    return int($number + .5 * ($number <=> 0));
}
◇
```

A partir de la versión 5, se incluye un módulo POSIX que contiene las funciones:

- $\text{floor}(x) = \lfloor x \rfloor$
- $\text{ceil}(x) = \lceil x \rceil$

### 3.10. IEEE 754

El estándar de coma flotante IEEE 754 especifica 4 modos de ‘redondeo’ que son útiles para acotación de errores, comprobación de la estabilidad de los cálculos, implementación de aritmética de intervalos, etc. Estos redondeos (de los cuales sólo el modo por defecto es un redondeo en el sentido definido aquí de ajuste al valor más cercano y los otros se denominan redondeos orientados —*directed*) se aplican al dígito menos significativo de los resultados de operaciones aritméticas que se calculan internamente con mayor precisión.

Los modos definidos por el estándar, para una base *b* arbitraria (en IEEE 754  $b = 2$ ) y *n* como posición del dígito menos significativo, son:

- *round to nearest*:  $\mathcal{R}_{2,n}^b(x)$  (*round to even*)
- *round toward  $+\infty$* :  $\lceil xb^n \rceil / b^n$  (*round up*)
- *round toward  $-\infty$* :  $\lfloor xb^n \rfloor / b^n$  (*round down*)
- *round toward 0*:  $\mathcal{E}xb^n / b^n$  (*truncate*)

### 3.10.1. IEEE 754r

La revisión del estándar especifica otros nombres para los modos de redondeo y se añade un nuevo modo de redondeo.

- *roundTiesToEven*:  $\mathcal{R}_{2,n}^b(x)$
- *roundTiesToAway*:  $\mathcal{R}_n^b(x) = \mathcal{R}_{\infty,n}^b(x)$
- *roundTowardPositive*:  $\lceil xb^n \rceil / b^n$
- *roundTowardNegative*:  $\lfloor xb^n \rfloor / b^n$
- *roundTowardZero*:  $\mathcal{E}xb^n / b^n$

## 3.11. Coprocesadores numéricos Intel

Los procesadores 8087, 80287, i387, i486, Pentium, etc., se ajustan a las normas de coma flotante IEEE y realizan un redondeo en el bit menos significativo en las operaciones aritméticas controlado por los bits 10 y 11 de la *palabra de control* del coprocesador:

- 00 (*round to nearest*):  $\mathcal{R}_{2,n}^b(x)$
- 01 (*round toward  $-\infty$* ):  $\lfloor xb^n \rfloor / b^n$
- 10 (*round toward  $+\infty$* ):  $\lceil xb^n \rceil / b^n$
- 11 (*round toward 0*):  $\mathcal{E}xb^n / b^n$

La instrucción `FRNDINT` convierte un valor a entero en función del modo de redondeo:

- modo 00 (*round to nearest*)  $\rightarrow \mathcal{R}_2(x)$
- modo 01 (*round toward  $-\infty$* )  $\rightarrow \lfloor x \rfloor$
- modo 10 (*round toward  $+\infty$* )  $\rightarrow \lceil x \rceil$
- modo 11 (*round toward 0*)  $\rightarrow \mathcal{E}(x)$

## 3.12. C/C++

### 3.12.1. C99

Las bibliotecas de C99 incluyen control del método de redondeo en `<fenv.h>` acordes al estándar IEEE y funciones de redondeo en `<tgmath.h>` y `<math.h>`.

- `fsetround(FE_DOWNWARD); \Rightarrow \text{nearbyint}(x) = \text{rint}(x) = \lfloor x \rfloor`
- `fsetround(FE_UPWARD); \Rightarrow \text{nearbyint}(x) = \text{rint}(x) = \lceil x \rceil`
- `fsetround(FE_TONEAREST); \Rightarrow \text{nearbyint}(x) = \text{rint}(x) = \mathcal{R}_2(x)`
- `fsetround(FE_TOWARDZERO); \Rightarrow \text{nearbyint}(x) = \text{rint}(x) = \mathcal{E}(x)`
- `round(x) = \mathcal{R}(x)`
- `trunc(x) = \mathcal{E}(x)`

### 3.12.2. C90

En C90 (C89), hay un valor definido en `<float.h>` que indica el modo de redondeo para la suma en coma flotante:

- `FLT_ROUNDS \equiv -1 \rightarrow` (indeterminado)
- `FLT_ROUNDS \equiv 0 \rightarrow \mathcal{E}(x)`
- `FLT_ROUNDS \equiv 1 \rightarrow \mathcal{R}(x)` (o bien otro redondeo al entero más cercano)
- `FLT_ROUNDS \equiv 2 \rightarrow \lceil x \rceil`
- `FLT_ROUNDS \equiv 3 \rightarrow \lfloor x \rfloor`

### 3.12.3. C++

En C++, este mismo valor está definido en `<limits>`, en `std::numeric_limits<...>::round_style`

- `std::round_indeterminable`  $\equiv -1 \rightarrow$  (indeterminado)
- `std::round_toward_zero`  $\equiv 0 \rightarrow \mathcal{E}(x)$
- `std::round_to_nearest`  $\equiv 1 \rightarrow \mathcal{R}(x)$  (u otro redondeo)
- `std::round_toward_infinity`  $\equiv 2 \rightarrow \lceil x \rceil$
- `std::round_toward_neg_infinity`  $\equiv 3 \rightarrow \lfloor x \rfloor$

### 3.13. General Decimal Arithmetic Specification

Existe un *contexto* que contiene parámetros y reglas que afectan a los resultados de las operaciones. Uno de los parámetros es el modo de redondeo (*rounding*) que indica el algoritmo que se usa para ajustar la precisión de los resultados, y que puede tomar los siguiente valores:

- *round-half-even* =  $\mathcal{R}_{2,n}(x)$  (em round to nearest, ties to even)
- *round-half-up* =  $\mathcal{R}_n(x) = \mathcal{R}_{\infty,n}(x)$  (round to nearest, ties away from zero)
- *round-half-down* =  $\mathcal{R}_{0,n}(x)$  ) opcional
- *round-up* =  $\begin{cases} \lceil (10^n x) \rceil / 10^n & \text{si } x \geq 0 \\ \lfloor (10^n x) \rfloor / 10^n & \text{si } x < 0 \end{cases}$  *round away from zero* opcional
- *round-down* =  $\mathcal{E}(10^n x) / 10^n = \begin{cases} \lfloor (10^n x) \rfloor / 10^n & \text{si } x \geq 0 \\ \lceil (10^n x) \rceil / 10^n & \text{si } x < 0 \end{cases}$  (*round toward zero, truncate*)
- *round-ceiling* =  $\lceil 10^n x \rceil / 10^n$  (*round toward  $+\infty$* )
- *round-floor* =  $\lfloor 10^n x \rfloor / 10^n$  (*round toward  $-\infty$* )

La especificación define una función para aplicar el redondeo definido por el contexto; por ejemplo si el redondeo es *round-half-up*:  $\text{quantize}(x, y) = \mathcal{R}_n(x)$  con  $n$  igual al exponente de  $y$ ; si  $y$  es normalizado se tiene  $n = \lfloor \log_{10} y \rfloor$ . Lo habitual es emplear potencias de 10 para  $y$ :  $\mathcal{R}_n(x) = \text{quantize}(10^{-n})$ .

Inicialmente se había definido una función equivalente con otra forma de especificar el número de dígitos:  $\text{rescale}(x, n) = \mathcal{R}_n(x)$

### 3.14. Ruby BigDecimal

La clase `BigDecimal` (incluida con Ruby, versión 1.8 y posteriores) tiene varios métodos y opciones de redondeo.

- `x.round(n, BigDecimal::ROUND_HALF_UP) = \mathcal{R}_n(x)`
- `x.round(n, BigDecimal::ROUND_HALF_EVEN) = \mathcal{R}_{2,n}(x)`

El modo de redondeo `ROUND_HALF_DOWN` es una versión defectuosa de  $\mathcal{R}_0$ , ya que sólo se tiene en cuenta el dígito siguiente al dígito redondeado (entendiendo por tal el dígito menos significativo mostrado después del redondeo, que puede haber sido ajustado). Si el dígito a redondear es 5 (o inferior), se redondea el valor hacia cero, aunque haya otros dígitos no nulos detrás de él. Si el dígito es 6 (o superior) se redondea hacia infinito.

Nota: en las últimas pruebas con Ruby 1.8.6, también `ROUND_HALF_EVEN` presenta este comportamiento anómalo.

Los siguientes modos de redondeo no son propiamente redondeos, corresponden a los modos del estándar IEEE y permiten la implementación de aritmética de intervalos; (las funciones tercera y cuarta corresponden a `ROUNDUP` y `ROUNDDOWN` de Excel.

- `x.round(n, BigDecimal::ROUND_CEILING) = x.ceil(n) = \lceil 10^n x \rceil / 10^n`

- $x.\text{round}(n, \text{BigDecimal}::\text{ROUND\_FLOOR}) = x.\text{floor}(n) = \lfloor 10^n x \rfloor / 10^n$
- $x.\text{round}(n, \text{BigDecimal}::\text{ROUND\_UP}) = \begin{cases} \lceil (10^n x) \rceil / 10^n & \text{si } x \geq 0 \\ \lfloor (10^n x) \rfloor / 10^n & \text{si } x < 0 \end{cases}$
- $x.\text{round}(n, \text{BigDecimal}::\text{ROUND\_DOWN}) = \mathcal{E}(10^n x) / 10^n = \begin{cases} \lfloor (10^n x) \rfloor / 10^n & \text{si } x \geq 0 \\ \lceil (10^n x) \rceil / 10^n & \text{si } x < 0 \end{cases}$

Como hemos visto, las funciones `floor`, `ceil` también admiten un parámetro opcional (con valor 0 por omisión) para especificar el dígito sobre el que se opera.

El modo de redondeo por omisión se puede establecer con `BigDecimal::mode`; el valor inicial por omisión es `ROUND_HALF_UP` ( $\mathcal{R}$ ).

```
BigDecimal::mode(BigDecimal::ROUND_MODE, BigDecimal::ROUND_UP)
BigDecimal::mode(BigDecimal::ROUND_MODE, BigDecimal::ROUND_DOWN)
BigDecimal::mode(BigDecimal::ROUND_MODE, BigDecimal::ROUND_HALF_UP)
BigDecimal::mode(BigDecimal::ROUND_MODE, BigDecimal::ROUND_HALF_DOWN)
BigDecimal::mode(BigDecimal::ROUND_MODE, BigDecimal::ROUND_HALF_EVEN)
BigDecimal::mode(BigDecimal::ROUND_MODE, BigDecimal::ROUND_CEILING)
BigDecimal::mode(BigDecimal::ROUND_MODE, BigDecimal::ROUND_FLOOR)
```

Este modo de redondeo se usa también para el redondeo interno, el que ocurre por ejemplo en la expresión:

```
BigDecimal(' .1E15' ).add(BigDecimal(' 0.5' ), 15)
```

La posición decimal a redondear es también opcional; el valor por omisión es 0. Si el modo de redondeo por omisión inicial no se ha modificado, tendremos que:

- $x.\text{round} = \mathcal{R}(x)$

Otros métodos:

- $x.\text{fix} = \mathcal{E}(x)$
- $x.\text{frac} = \mathcal{F}(x)$
- $x.\text{truncate} = \mathcal{E}(x)$
- $x.\text{truncate}(n) = \mathcal{E}(10^n x) / 10^n$

Nota: `to_i` da el mismo resultado que `fix`, pero en forma de número entero (`Integer`) en lugar de `BigDecimal`.

### 3.15. Python Decimal

El tipo `Decimal` de Python está regido por la *General Decimal Arithmetic Specification*.

Los modos de redondeo (asignados por ejemplo mediante `getcontext().rounding=ROUND_HALF_EVEN`) son los siguientes:

- $\text{ROUND\_CEILING} = x.\text{ceil}(n) = \lceil 10^n x \rceil / 10^n$
- $\text{ROUND\_DOWN} = \mathcal{E}(10^n x) / 10^n = \begin{cases} \lfloor (10^n x) \rfloor / 10^n & \text{si } x \geq 0 \\ \lceil (10^n x) \rceil / 10^n & \text{si } x < 0 \end{cases}$
- $\text{ROUND\_FLOOR} = x.\text{floor}(n) = \lfloor 10^n x \rfloor / 10^n$
- $\text{ROUND\_HALF\_UP} = \mathcal{R}_n(x)$
- $\text{ROUND\_HALF\_EVEN} = \mathcal{R}_{2,n}(x)$
- $\text{ROUND\_HALF\_DOWN} = \mathcal{R}_{0,n}(x)$
- $\text{ROUND\_UP} = \begin{cases} \lceil (10^n x) \rceil / 10^n & \text{si } x \geq 0 \\ \lfloor (10^n x) \rfloor / 10^n & \text{si } x < 0 \end{cases}$

El contexto aritmético contiene además del modo de redondeo y otros parámetros la precisión (número de decimales) de las operaciones y conversiones. Los modos de redondeo se utilizan en todas las operaciones, de forma que el resultado equivale a realizar la operación de forma exacta y después redondearlo mediante el modo y precisión establecidos en el contexto. El modo de redondeo no se emplean también en la conversión de literales de texto a números.

El modo de redondeo inicial por omisión es `ROUND_HALF_EVEN` ( $\mathcal{R}_2$ ) y la precisión 28.

Hay una función genérica de redondeo, `quantize` que se puede usar con cualquiera de los modos:

`x.quantize(y, rounding=ROUND_HALF_EVEN) = \mathcal{R}_{2,n}` con `y=Decimal(10)**(-n)` ( $y = 10^{-n}$ ) o, con más generalidad `n=y.as_tuple()[2]` ( $n$  es el exponente de  $y$ ). Si  $y$  es un valor normalizado, entonces  $n = \lfloor \log_{10} y \rfloor$ .