Fixed-Point Toolbox 2
Reference

MATLAB®

The MathWorks
Accelerating the pace of engineering and science

How to Contact The MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Fixed-Point Toolbox Reference

Revision History

June 2004	First printing	New for Version 1.0 (Release 14)
October 2004	Online only	Version 1.1 (Release 14SP1)
March 2005	Online only	Version 1.2 (Release 14SP2)
September 2005	Online only	Version 1.3 (Release 14SP3)
October 2005	Second printing	Version 1.3
March 2006	Online only	Version 1.4 (R2006a)
September 2006	Online only	Version 1.5 (R2006b)
March 2007	Online only	Version 2.0 (R2007a)

Contents

1 Property Reference

2 Functions — By Category

1

Property Reference

fi Object Properties

The properties associated with fi objects are described in the following sections in alphabetical order.

Note The fimath properties and numerictype properties are also properties of the fi object. Refer to "fimath Object Properties" on page 1-6 and "numerictype Object Properties" on page 1-17 for more information.

---

bin

Stored integer value of a fi object in binary.

data

Numerical real-world value of a fi object.

dec

Stored integer value of a fi object in decimal.

double

Real-world value of a fi object stored as a MATLAB double.

fimath

fimath object associated with a fi object. The default fimath object has the following settings:

```
            RoundMode: nearest
         OverflowMode: saturate
          ProductMode: FullPrecision
 MaxProductWordLength: 128
              SumMode: FullPrecision
     MaxSumWordLength: 128
        CastBeforeSum: true
```

To learn more about `fimath` properties, refer to "fimath Object Properties" on page 1-6.

hex

Stored integer value of a `fi` object in hexadecimal.

int

Stored integer value of a `fi` object, stored in a built-in MATLAB integer data type. You can also use `int8`, `int16`, `int32`, `uint8`, `uint16`, and `uint32` to get the stored integer value of a fi object in these formats.

NumericType

Structure containing all the data type and scaling attributes of a `fi` object. The `numerictype` object acts the same way as any MATLAB structure, except that it only lets you set valid values for defined fields. The following table shows the possible settings of each field of the structure that are valid for `fi` objects.

DataTypeMode	DataType	Scaling	Signed	WordLength	FractionLength	Slope	Bias
Fully specified fixed-point data types							
`Fixed-point: binary point scaling`	`Fixed`	`BinaryPoint`	`1/0` `true/false`	positive integer from 1 to 65,536	positive or negative integer	1	0
`Fixed-point: slope and bias scaling`	`Fixed`	`SlopeBias`	`1/0` `true/false`	positive integer from 1 to 65,536	`N/A`	any floating-point number	any floating-point number
Partially specified fixed-point data type							

DataTypeMode	Data-Type	Scaling	Signed	Word-Length	Fraction-Length	Slope	Bias

`Fixed-point: unspecified scaling`	`Fixed`	`Unspecified`	`1/0`
			
			`true/ false`
				positive integer from 1 to 65,536	`N/A`	`N/A`	`N/A`

Fully specified scaled double data types							

`Scaled double: binary point scaling`	`ScaledDouble`	`BinaryPoint`	`1/0`
			
			`true/ false`
				positive integer from 1 to 65,536	positive or negative integer	`1`	`0`

`Scaled double: slope and bias scaling`	`ScaledDouble`	`SlopeBias`	`1/0`
			
			`true/ false`
				positive integer from 1 to 65,536	`N/A`	any floating-point number	any floating-point number

Partially specified scaled double data type							

`Scaled double: unspecified scaling`	`ScaledDouble`	`Unspecified`	`1/0`
			
			`true/ false`
				positive integer from 1 to 65,536	`N/A`	`N/A`	`N/A`

Built-in data types							
`double`	`double`	`N/A`	`1` `true`	`64`	`0`	`1`	`0`
`single`	`single`	`N/A`	`1` `true`	`32`	`0`	`1`	`0`
`boolean`	`boolean`	`N/A`	`0` `false`	`1`	`0`	`1`	`0`

You cannot change the `numerictype` properties of a `fi` object after `fi` object creation.

oct

Stored integer value of a `fi` object in octal.

fimath Object Properties

The properties associated with `fimath` objects are described in the following sections in alphabetical order.

CastBeforeSum

Whether both operands are cast to the sum data type before addition. Possible values of this property are `1` (cast before sum) and `0` (do not cast before sum).

The default value of this property is `1` (true).

MaxProductWordLength

Maximum allowable word length for the product data type.

The default value of this property is `128`.

MaxSumWordLength

Maximum allowable word length for the sum data type.

The default value of this property is `128`.

OverflowMode

Overflow-handling mode. The value of the `OverflowMode` property can be one of the following strings:

- `saturate` — Saturate to maximum or minimum value of the fixed-point range on overflow.
- `wrap` — Wrap on overflow. This mode is also known as two's complement overflow.

The default value of this property is `saturate`.

ProductBias

Bias of the product data type. This value can be any floating-point number. The product data type defines the data type of the result of a multiplication of two `fi` objects.

The default value of this property is `0`.

ProductFixedExponent

Fixed exponent of the product data type. This value can be any positive or negative integer. The product data type defines the data type of the result of a multiplication of two `fi` objects.

$ProductSlope = ProductSlopeAdjustmentFactor * 2 \wedge ProductFixedExponent$. Changing one of these properties changes the others.

The `ProductFixedExponent` is the negative of the `ProductFractionLength`. Changing one property changes the other.

The default value of this property is `-30`.

ProductFractionLength

Fraction length, in bits, of the product data type. This value can be any positive or negative integer. The product data type defines the data type of the result of a multiplication of two `fi` objects.

The `ProductFractionLength` is the negative of the `ProductFixedExponent`. Changing one property changes the other.

The default value of this property is `30`.

ProductMode

Defines how the product data type is determined. In the following descriptions, let A and B be real operands, with [word length, fraction length] pairs [W_a F_a] and [W_b F_b], respectively. W_p is the product data type word length and F_p is the product data type fraction length.

- `FullPrecision` — The full precision of the result is kept. An error is generated if the calculated word length is greater than `MaxProductWordLength`.

  $$W_p = W_a + W_b$$
  $$F_p = F_a + F_b$$

- `KeepLSB` — Keep least significant bits. You specify the product data type word length, while the fraction length is set to maintain the least significant bits of the product. In this mode, full precision is kept, but overflow is possible. This behavior models the C language integer operations.

  $$W_p = \text{specified in the } \texttt{ProductWordLength} \text{ property}$$
  $$F_p = F_a + F_b$$

- `KeepMSB` — Keep most significant bits. You specify the product data type word length, while the fraction length is set to maintain the most significant bits of the product. In this mode, overflow is prevented, but precision may be lost.

  $$W_p = \text{specified in the } \texttt{ProductWordLength} \text{ property}$$
  $$F_p = W_p - \text{integer length}$$

  where

  $$\text{integer length} = (W_a + W_b) - (F_a - F_b)$$

- `SpecifyPrecision` — You specify both the word length and fraction length of the product data type.

  $$W_p = \text{specified in the } \texttt{ProductWordLength} \text{ property}$$
  $$F_p = \text{specified in the } \texttt{ProductFractionLength} \text{ Property}$$

  For [Slope Bias] math, you specify both the slope and bias of the product data type.

  $$S_p = \text{specified in the } \texttt{ProductSlope} \text{ property}$$
  $$B_p = \text{specified in the } \texttt{ProductBias} \text{ property}$$

[Slope Bias] math is only defined for products when `ProductMode` is set to `SpecifyPrecision`.

The default value of this property is `FullPrecision`.

ProductSlope

Slope of the product data type. This value can be any floating-point number. The product data type defines the data type of the result of a multiplication of two `fi` objects.

$ProductSlope = ProductSlopeAdjustmentFactor * 2\,\hat{}\; ProductFixedExponent$. Changing one of these properties changes the others.

The default value of this property is `9.3132e-010`.

ProductSlopeAdjustmentFactor

Slope adjustment factor of the product data type. This value can be any floating-point number greater than or equal to 1 and less than 2. The product data type defines the data type of the result of a multiplication of two `fi` objects.

$ProductSlope = ProductSlopeAdjustmentFactor * 2\,\hat{}\; ProductFixedExponent$. Changing one of these properties changes the others.

The default value of this property is `1`.

ProductWordLength

Word length, in bits, of the product data type. This value must be a positive integer. The product data type defines the data type of the result of a multiplication of two `fi` objects.

The default value of this property is `32`.

RoundMode

The rounding mode. The value of the `RoundMode` property can be one of the following strings:

- `ceil` — Round toward positive infinity.
- `convergent` — Round to the closest representable integer. Ties round to the nearest even stored integer. This is the least biased rounding method provided by Fixed-Point Toolbox.
- `fix` — Round toward zero.
- `floor` — Round toward negative infinity.
- `nearest` — Round toward nearest. Ties round toward positive infinity.
- `round` — Round toward nearest. Ties round toward negative infinity for negative numbers, and toward positive infinity for positive numbers.

The default value of this property is `nearest`.

SumBias

The bias of the sum data type. This value can be any floating-point number. The sum data type defines the data type of the result of a sum of two `fi` objects.

The default value of this property is `0`.

SumFixedExponent

The fixed exponent of the sum data type. This value can be any positive or negative integer. The sum data type defines the data type of the result of a sum of two `fi` objects

$SumSlope = SumSlopeAdjustmentFactor * 2 \wedge SumFixedExponent$. Changing one of these properties changes the others.

The `SumFixedExponent` is the negative of the `SumFractionLength`. Changing one property changes the other.

The default value of this property is `-30`.

SumFractionLength

The fraction length, in bits, of the sum data type. This value can be any positive or negative integer. The sum data type defines the data type of the result of a sum of two `fi` objects.

The `SumFractionLength` is the negative of the `SumFixedExponent`. Changing one property changes the other.

The default value of this property is`30` .

SumMode

Defines how the sum data type is determined. In the following descriptions, let *A* and *B* be real operands, with [word length, fraction length] pairs [W_a F_a] and [W_b F_b], respectively. W_s is the sum data type word length and F_s is the sum data type fraction length.

Note In the case where there are two operands, as in *A* + *B*, *NumberOfSummands* is `2`, and `ceil(log2(NumberOfSummands)) = 1`. In `sum(A)` where *A* is a matrix, the *NumberOfSummands* is `size(A,1)`. In `sum(A)` where *A* is a vector, the *NumberOfSummands* is `length(A)`.

- `FullPrecision` — The full precision of the result is kept. An error is generated if the calculated word length is greater than `MaxSumWordLength`.

  $$W_s = \text{integer length} + F_s$$

  where

  $$\text{integer length} = \max\left(W_a - F_a, W_b - F_b\right) + \text{ceil}\left(\log 2\left(NumberOfSummands\right)\right)$$

  $$F_s = \max(F_a, F_b)$$

- `KeepLSB` — Keep least significant bits. You specify the sum data type word length, while the fraction length is set to maintain the least significant bits of the sum. In this mode, full precision is kept, but overflow is possible. This behavior models the C language integer operations.

W_s = specified in the `SumWordLength` property

$F_s = \max(F_a, F_b)$

- `KeepMSB` — Keep most significant bits. You specify the sum data type word length, while the fraction length is set to maintain the most significant bits of the sum and no more fractional bits than necessary. In this mode, overflow is prevented, but precision may be lost.

  W_s = specified in the `SumWordLength` property

  $F_s = W_s - \text{integer length}$

  where

  $$\text{integer length} = \max(W_a - F_a, W_b - F_b) + \text{ceil}(\log 2(NumberOfSummands))$$

- `SpecifyPrecision` — You specify both the word length and fraction length of the sum data type.

  W_s = specified in the `SumWordLength` property

  F_s = specified in the `SumFractionLength` property

  For [Slope Bias] math, you specify both the slope and bias of the sum data type.

  S_s = specified in the `SumSlope` property

  B_s = specified in the `SumBias` property

  [Slope Bias] math is only defined for sums when `SumMode` is set to `SpecifyPrecision`.

The default value of this property is `FullPrecision`.

SumSlope

The slope of the sum data type. This value can be any floating-point number. The sum data type defines the data type of the result of a sum of two `fi` objects.

$SumSlope = SumSlopeAdjustmentFactor * 2\text{^}SumFixedExponent$. Changing one of these properties changes the others.

The default value of this property is `9.3132e-010`.

SumSlopeAdjustmentFactor

The slope adjustment factor of the sum data type. This value can be any floating-point number greater than or equal to 1 and less than 2. The sum data type defines the data type of the result of a sum of two `fi` objects.

$SumSlope = SumSlopeAdjustmentFactor * 2^{SumFixedExponent}$. Changing one of these properties changes the others.

The default value of this property is `1`.

SumWordLength

The word length, in bits, of the sum data type. This value must be a positive integer. The sum data type defines the data type of the result of a sum of two `fi` objects.

The default value of this property is `32`.

fipref Object Properties

The properties associated with fipref objects are described in the following sections in alphabetical order.

DataTypeOverride

Data type override options for fi objects

- ForceOff — No data type override
- ScaledDoubles — Override with scaled doubles
- TrueDoubles — Override with doubles
- True Singles — Override with singles

Data type override only occurs when the fi constructor function is called.

The default value of this property is ForceOff.

FimathDisplay

Display options for the fimath attributes of a fi object

- full — Displays all of the fimath attributes of a fixed-point object
- none — None of the fimath attributes are displayed

The default value of this property is full.

LoggingMode

Logging options for operations performed on fi objects

- off — No logging
- on — Information is logged for future operations

Overflows and underflows for assignment, plus, minus, and multiplication operations are logged as warnings when LoggingMode is set to on.

When `LoggingMode` is `on`, you can also use the following functions to return logged information about assignment and creation operations to the MATLAB command line:

- `maxlog` — Returns the maximum real-world value
- `minlog` — Returns the minimum value
- `noverflows` — Returns the number of overflows
- `nunderflows` — Returns the number of underflows

`LoggingMode` must be set to `on` before you perform any operation in order to log information about it. To clear the log, use the function `resetlog`.

The default value of this property of `off`.

NumericTypeDisplay

Display options for the `numerictype` attributes of a `fi` object

- `full` — Displays all the `numerictype` attributes of a fixed-point object
- `none` — None of the `numerictype` attributes are displayed.
- `short` — Displays an abbreviated notation of the fixed-point data type and scaling of a fixed-point object in the format `xWL,FL` where
  - `x` is `s` for signed and `u` for unsigned.
  - `WL` is the word length.
  - `FL` is the fraction length.

The default value of this property is `full`.

NumberDisplay

Display options for the value of a `fi` object

- `bin` — Displays the stored integer value in binary format
- `dec` — Displays the stored integer value in unsigned decimal format

- `RealWorldValue` — Displays the stored integer value in the format specified by the MATLAB `format` function
- `hex` — Displays the stored integer value in hexadecimal format
- `int` — Displays the stored integer value in signed decimal format
- `none` — No value is displayed.

The default value of this property is `RealWorldValue`. In this mode, the value of a `fi` object is displayed in the format specified by the MATLAB `format` function: `+`, `bank`, `compact`, `hex`, `long`, `long e`, `long g`, `loose`, `rat`, `short`, `short e`, or `short g`. `fi` objects in `rat` format are displayed according to

$$1/\left(2^{\mathit{fixed\text{-}point\ exponent}}\right) \times \mathit{stored\ integer}$$

numerictype Object Properties

The properties associated with numerictype objects are described in the following sections in alphabetical order.

Bias

Bias associated with a fi object. The bias is part of the numerical representation used to interpret a fixed-point number. Along with the slope, the bias forms the scaling of the number. Fixed-point numbers can be represented as

$$\textit{real-world value} = (\textit{slope} \times \textit{integer}) + \textit{bias}$$

where the slope can be expressed as

$$\textit{slope} = \textit{fractionalslope} \times 2^{\textit{fixed exponent}}$$

DataType

Data type associated with a fi object. The possible value of this property are

- boolean — Built-in MATLAB boolean data type
- double — Built-in MATLAB double data type
- Fixed — Fixed-point or integer data type
- ScaledDouble — Scaled double data type
- single — Built-in MATLAB single data type

The default value of this property is fixed.

DataTypeMode

Data type and scaling associated with a fi object. The possible values of this property are

- boolean — Built-in boolean

- `double` — Built-in `double`
- `Fixed-point:  binary point scaling` — Fixed-point data type and scaling defined by the word length and fraction length
- `Fixed-point:  slope and bias scaling` — Fixed-point data type and scaling defined by the slope and bias
- `Fixed-point:  unspecified scaling` —- Fixed-point data type with unspecified scaling
- `Scaled double:  binary point scaling` — Double data type with fixed-point word length and fraction length information retained
- `Scaled double:  slope and bias scaling` — Double data type with fixed-point slope and bias information retained
- `Scaled double:  unspecified scaling` —- Double data type with unspecified fixed-point scaling
- `single` — Built-in `single`

The default value of this property is `Fixed-point:  binary point scaling`.

FixedExponent

Fixed-point exponent associated with a `fi` object. The exponent is part of the numerical representation used to express a fixed-point number. Fixed-point numbers can be represented as

$$\text{real-world value} = (slope \times integer) + bias$$

where the slope can be expressed as

$$slope = fractionalslope \times 2^{fixed\ exponent}$$

The exponent of a fixed-point number is equal to the negative of the fraction length:

$$fixed\ exponent = -fraction\ length$$

FractionLength

Value of the `FractionLength` property is the fraction length of the stored integer value of a `fi` object, in bits. The fraction length can be any integer value. If you do not specify the fraction length of a `fi` object, it is set to the best possible precision.

This property is automatically set by default to the best precision possible based on the value of the word length.

Scaling

Fixed-point scaling mode of a `fi` object. The possible values of this property are

- `BinaryPoint` — Scaling for the `fi` object is defined by the fraction length.
- `SlopeBias` — Scaling for the `fi` object is defined by the slope and bias.
- `Unspecified` — A temporary setting that is only allowed at `fi` object creation, in order to allow for the automatic assignment of a binary point best precision scaling.
- `Integer` — The `fi` object is an integer; the binary point is understood to be at the far right of the word, making the fraction length zero.

The default value of this property is `BinaryPoint`.

Signed

Whether a `fi` object is signed. The possible values of this property are

- `1` — signed
- `0` — unsigned
- `true` — signed
- `false` — unsigned

The default value of this property is `true` .

Slope

Slope associated with a `fi` object. The slope is part of the numerical representation used to express a fixed-point number. Along with the bias, the slope forms the scaling of a fixed-point number. Fixed-point numbers can be represented as

$$real\text{-}world\ value = (slope \times integer) + bias$$

where the slope can be expressed as

$$slope = fractional\,slope \times 2^{fixed\ exponent}$$

SlopeAdjustmentFactor

Slope adjustment associated with a `fi` object. The slope adjustment is equivalent to the fractional slope of a fixed-point number. The fractional slope is part of the numerical representation used to express a fixed-point number. Fixed-point numbers can be represented as

$$real\text{-}world\ value = (slope \times integer) + bias$$

where the slope can be expressed as

$$slope = fractional\,slope \times 2^{fixed\ exponent}$$

WordLength

Value of the `WordLength` property is the word length of the stored integer value of a fixed-point object, in bits. The word length can be any positive integer value.

The default value of this property is `16`.

quantizer Object Properties

The properties associated with `quantizer` objects are described in the following sections in alphabetical order.

DataMode

Type of arithmetic used in quantization. This property can have the following values:

- `fixed` — Signed fixed-point calculations
- `float` — User-specified floating-point calculations
- `double` — Double-precision floating-point calculations
- `single` — Single-precision floating-point calculations
- `ufixed` — Unsigned fixed-point calculations

The default value of this property is `fixed`.

When you set the `DataMode` property value to `double` or `single`, the `Format` property value becomes read only.

Format

Data format of a `quantizer` object. The interpretation of this property value depends on the value of the `DataMode` property.

For example, whether you specify the `DataMode` property with fixed- or floating-point arithmetic affects the interpretation of the data format property. For some `DataMode` property values, the data format property is read only.

The following table shows you how to interpret the values for the `Format` property value when you specify it, or how it is specified in read-only cases.

DataMode Property Value	Interpreting the Format Property Values
`fixed` or `ufixed`	You specify the `Format` property value as a vector. The number of bits for the `quantizer` object word length is the first entry of this vector, and the number of bits for the `quantizer` object fraction length is the second entry. The word length can range from 2 to the limits of memory on your PC. The fraction length can range from 0 to one less than the word length.
`float`	You specify the `Format` property value as a vector. The number of bits you want for the `quantizer` object word length is the first entry of this vector, and the number of bits you want for the `quantizer` object exponent length is the second entry. The word length can range from 2 to the limits of memory on your PC. The exponent length can range from 0 to 11.
`double`	The `Format` property value is specified automatically (is read only) when you set the `DataMode` property to `double`. The value is [64 11], specifying the word length and exponent length, respectively.
`single`	The `Format` property value is specified automatically (is read only) when you set the `DataMode` property to `single`. The value is [32 8], specifying the word length and exponent length, respectively.

OverflowMode

Overflow-handling mode. The value of the `OverflowMode` property can be one of the following strings:

- `saturate` — Overflows saturate.

  When the values of data to be quantized lie outside the range of the largest and smallest representable numbers (as specified by the data format properties), these values are quantized to the value of either the largest or smallest representable value, depending on which is closest.

- `wrap` — Overflows wrap to the range of representable values.

  When the values of data to be quantized lie outside the range of the largest and smallest representable numbers (as specified by the data format properties), these values are wrapped back into that range using modular arithmetic relative to the smallest representable number.

The default value of this property is `saturate`.

Note Floating-point numbers that extend beyond the dynamic range overflow to `±inf`.

The `OverflowMode` property value is set to `saturate` and becomes a read-only property when you set the value of the `DataMode` property to `float`, `double`, or `single`.

RoundMode

Rounding mode. The value of the `RoundMode` property can be one of the following strings:

- `ceil` — Round up to the next allowable quantized value.
- `convergent` — Round to the nearest allowable quantized value. Numbers that are exactly halfway between the two nearest allowable quantized values are rounded up only if the least significant bit (after rounding) would be set to 0.
- `fix` — Round negative numbers up and positive numbers down to the next allowable quantized value.
- `floor` — Round down to the next allowable quantized value.
- `nearest` — Round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.

The default value of this property is `floor`.

Functions — By Category

Bitwise Functions

`bitand`	Bitwise `AND` of two `fi` objects
`bitcmp`	Bitwise complement of `fi` object
`bitget`	Bit at certain position
`bitor`	Bitwise `OR` of two `fi` objects
`bitset`	Set bit at certain position
`bitshift`	Shift bits specified number of places
`bitxor`	Bitwise exclusive `OR` of two `fi` objects

Constructor and Property Functions

`copyobj`	Make independent copy of `quantizer` object
`fi`	Construct `fi` object
`fimath`	Construct `fimath` object
`fipref`	Construct `fipref` object
`get`	Property values of object

`numerictype`	Construct `numerictype` object
`quantizer`	Construct `quantizer` object
`reset`	Reset objects to initial conditions
`savefipref`	Save `fi` preferences for next MATLAB session
`set`	Set or display property values for `quantizer` objects
`stripscaling`	Stored integer of `fi` object
`tostring`	Convert `quantizer` object to string

Data Manipulation Functions

`denormalmax`	Largest denormalized quantized number for `quantizer` object
`denormalmin`	Smallest denormalized quantized number for `quantizer` object
`eps`	Quantized relative accuracy for `fi` or `quantizer` objects
`exponentbias`	Exponent bias for `quantizer` object
`exponentlength`	Exponent length of `quantizer` object
`exponentmax`	Maximum exponent for `quantizer` object
`exponentmin`	Minimum exponent for `quantizer` object
`fractionlength`	Fraction length of `quantizer` object
`intmax`	Largest positive stored integer value representable by `numerictype` of `fi` object

`intmin`	Smallest stored integer value representable by `numerictype` of `fi` object
`isequal`	Determine whether real-world values of two `fi` objects are equal, or determine whether properties of two `fimath`, `numerictype`, or `quantizer` objects are equal
`isfi`	Determine whether variable is `fi` object
`isfimath`	Determine whether variable is `fimath` object
`isnumerictype`	Determine whether variable is `numerictype` object
`ispropequal`	Determine whether properties of two `fi` objects are equal
`issigned`	Determine whether `fi` object is signed
`lowerbound`	Lower bound of range of `fi` object
`lsb`	Scaling of least significant bit of `fi` object
`range`	Numerical range of `fi` or `quantizer` object
`realmax`	Largest positive fixed-point value or quantized number
`realmin`	Smallest positive normalized fixed-point value or quantized number
`rescale`	Change scaling of `fi` object
`upperbound`	Upper bound of range of `fi` object
`wordlength`	Word length of `quantizer` object

Data Type Functions

`double`	Double-precision floating-point real-world value of `fi` object
`int`	Smallest built-in integer in which stored integer value of `fi` object will fit
`int16`	Stored integer value of `fi` object as built-in `int16`
`int32`	Stored integer value of `fi` object as built-in `int32`
`int8`	Stored integer value of `fi` object as built-in `int8`
`logical`	Convert numeric values to logical
`single`	Single-precision floating-point real-world value of `fi` object
`uint16`	Stored integer value of `fi` object as built-in `uint16`
`uint32`	Stored integer value of `fi` object as built-in `uint32`
`uint8`	Stored integer value of `fi` object as built-in `uint8`

Data Quantizing Functions

`convergent`	Apply convergent rounding
`quantize`	Apply `quantizer` object to data

`randquant`	Generate uniformly distributed, quantized random number using `quantizer` object
`round`	Round input data using `quantizer` object without checking for overflow

Element-Wise Logical Operator Functions

`all`	Determine whether all array elements are nonzero
`and`	Find logical AND of array or scalar inputs
`any`	Determine whether any array elements are nonzero
`not`	Find logical NOT of array or scalar input
`or`	Find logical OR of array or scalar inputs

Math Operation Functions

`abs`	Absolute value of `fi` object
`add`	Add two objects using `fimath` object
`complex`	Construct complex `fi` object from real and imaginary parts
`conj`	Complex conjugate of `fi` object
`divide`	Divide two objects
`imag`	Imaginary part of complex number

`innerprodintbits`	Number of integer bits needed for fixed-point inner product
`minus`	Matrix difference between `fi` objects
`mpy`	Multiply two objects using `fimath` object
`mtimes`	Matrix product of `fi` objects
`plus`	Matrix sum of `fi` objects
`pow2`	Multiply by 2^K
`real`	Real part of complex number
`sign`	Perform signum function on array
`sqrt`	Square root of `fi` object
`sub`	Subtract two objects using `fimath` object
`sum`	Sum of array elements
`times`	Element-by-element multiplication of `fi` objects
`uminus`	Negate elements of `fi` object array
`uplus`	Unary plus

Matrix Manipulation Functions

`buffer`	Buffer signal vector into matrix of data frames
`ctranspose`	Complex conjugate transpose of `fi` object
`diag`	Diagonal matrices or diagonals of matrix
`disp`	Display object
`end`	Last index of array

`flipdim`	Flip array along specified dimension
`fliplr`	Flip matrix left to right
`flipud`	Flip matrix up to down
`hankel`	Hankel matrix
`horzcat`	Horizontally concatenate multiple `fi` objects
`ipermute`	Inverse permute dimensions of multidimensional array
`iscolumn`	Determine whether `fi` object is column vector
`isempty`	Determine whether array is empty
`isfinite`	Determine whether array elements are finite
`isinf`	Determine whether array elements are infinite
`isnan`	Determine whether array elements are NaN
`isnumeric`	Determine whether input is numeric array
`isobject`	Determine whether input is MATLAB OOPS object
`isreal`	Determine whether array elements are real
`isrow`	Determine whether `fi` object is row vector
`isscalar`	Determine whether input is scalar
`isvector`	Determine whether input is vector
`length`	Vector length
`ndims`	Number of array dimensions
`permute`	Rearrange dimensions of multidimensional array

`repmat`	Replicate and tile array
`reshape`	Reshape array
`shiftdim`	Shift dimensions
`size`	Array dimensions
`squeeze`	Remove singleton dimensions
`toeplitz`	Create Toeplitz matrix
`transpose`	Transpose operation
`tril`	Lower triangular part of matrix
`vertcat`	Vertically concatenate multiple `fi` objects

Plotting Functions

`area`	Create filled area 2-D plot
`bar`	Create vertical bar graph
`barh`	Create horizontal bar graph
`clabel`	Create contour plot elevation labels
`comet`	Create 2-D comet plot
`comet3`	Create 3-D comet plot
`compass`	Plot arrows emanating from origin
`coneplot`	Plot velocity vectors as cones in 3-D vector field
`contour`	Create contour graph of matrix
`contour3`	Create 3-D contour plot
`contourc`	Create two-level contour plot computation
`contourf`	Create filled 2-D contour plot

`errorbar`	Plot error bars along curve
`etreeplot`	Plot elimination tree
`ezcontour`	Easy-to-use contour plotter
`ezcontourf`	Easy-to-use filled contour plotter
`ezmesh`	Easy-to-use 3-D mesh plotter
`ezplot`	Easy-to-use function plotter
`ezplot3`	Easy-to-use 3-D parametric curve plotter
`ezpolar`	Easy-to-use polar coordinate plotter
`ezsurf`	Easy-to-use 3-D colored surface plotter
`ezsurfc`	Easy-to-use combination surface/contour plotter
`feather`	Plot velocity vectors
`fplot`	Plot function between specified limits
`gplot`	Plot set of nodes using adjacency matrix
`hist`	Create histogram plot
`histc`	Histogram count
`line`	Create line object
`loglog`	Create log-log scale plot
`mesh`	Create mesh plot
`meshc`	Create mesh plot with contour plot
`meshz`	Create mesh plot with curtain plot
`patch`	Create patch graphics object
`pcolor`	Create pseudocolor plot
`plot`	Create linear 2-D plot
`plot3`	Create 3-D line plot

`plotmatrix`	Draw scatter plots
`plotyy`	Create graph with y-axes on right and left sides
`polar`	Plot polar coordinates
`quiver`	Create quiver or velocity plot
`quiver3`	Create 3-D quiver or velocity plot
`rgbplot`	Plot colormap
`ribbon`	Create ribbon plot
`rose`	Create angle histogram
`scatter`	Create scatter or bubble plot
`scatter3`	Create 3-D scatter or bubble plot
`semilogx`	Create semilogarithmic plot with logarithmic x-axis
`semilogy`	Create semilogarithmic plot with logarithmic y-axis
`slice`	Create volumetric slice plot
`spy`	Visualize sparsity pattern
`stairs`	Create stairstep graph
`stem`	Plot discrete sequence data
`stem3`	Plot 3-D discrete sequence data
`streamribbon`	Create 3-D stream ribbon plot
`streamslice`	Draw streamlines in slice planes
`streamtube`	Create 3-D stream tube plot
`surf`	Create 3-D shaded surface plot
`surfc`	Create 3-D shaded surface plot with contour plot
`surfl`	Create surface plot with colormap-based lighting

`surfnorm`	Compute and display 3-D surface normals
`text`	Create text object in current axes
`treeplot`	Plot picture of tree
`trimesh`	Create triangular mesh plot
`triplot`	Create 2-D triangular plot
`trisurf`	Create triangular surface plot
`triu`	Upper triangular part of matrix
`voronoi`	Create Voronoi diagram
`voronoin`	Create n-D Voronoi diagram
`waterfall`	Create waterfall plot
`xlim`	Set or query x-axis limits
`ylim`	Set or query y-axis limits
`zlim`	Set or query z-axis limits

Radix Conversion Functions

`bin`	Binary representation of stored integer of `fi` object
`bin2num`	Convert two's complement binary string to number using `quantizer` object
`dec`	Unsigned decimal representation of stored integer of `fi` object
`hex`	Hexadecimal representation of stored integer of `fi` object
`hex2num`	Convert hexadecimal string to number using `quantizer` object

`num2bin`	Convert number to binary string using `quantizer` object
`num2hex`	Convert number to hexadecimal equivalent using `quantizer` object
`num2int`	Convert number to signed integer
`oct`	Octal representation of stored integer of `fi` object
`sdec`	Signed decimal representation of stored integer of `fi` object

Relational Operator Functions

`eq`	Determine whether real-world values of two `fi` objects are equal
`ge`	Determine whether real-world value of one `fi` object is greater than or equal to another
`gt`	Determine whether real-world value of one `fi` object is greater than another
`le`	Determine whether real-world value of `fi` object is less than or equal to another
`lt`	Determine whether real-world value of one `fi` object is less than another
`ne`	Determine whether real-world values of two `fi` objects are not equal

Statistics Functions

`max`	Largest element in array of `fi` objects
`maxlog`	Largest real-world value of `fi` object or maximum value of `quantizer` object before quantization
`min`	Smallest element in array of `fi` objects
`minlog`	Smallest real-world value of `fi` object or minimum value of `quantizer` object before quantization
`noperations`	Number of operations
`noverflows`	Number of overflows
`numberofelements`	Number of data elements in `fi` array
`nunderflows`	Number of underflows
`resetlog`	Clear log for `fi` or `quantizer` object

Subscripted Assignment and Reference Functions

`subsasgn`	Subscripted assignment
`subsref`	Subscripted reference

fi Object Functions

`abs`	Absolute value of `fi` object
`all`	Determine whether all array elements are nonzero
`and`	Find logical AND of array or scalar inputs
`any`	Determine whether any array elements are nonzero
`area`	Create filled area 2-D plot
`bar`	Create vertical bar graph
`barh`	Create horizontal bar graph
`bin`	Binary representation of stored integer of `fi` object
`bitand`	Bitwise `AND` of two `fi` objects
`bitcmp`	Bitwise complement of `fi` object
`bitget`	Bit at certain position
`bitor`	Bitwise `OR` of two `fi` objects
`bitshift`	Shift bits specified number of places
`bitxor`	Bitwise exclusive `OR` of two `fi` objects
`buffer`	Buffer signal vector into matrix of data frames
`clabel`	Create contour plot elevation labels
`comet`	Create 2-D comet plot
`comet3`	Create 3-D comet plot
`compass`	Plot arrows emanating from origin
`complex`	Construct complex `fi` object from real and imaginary parts
`coneplot`	Plot velocity vectors as cones in 3-D vector field

`conj`	Complex conjugate of `fi` object
`contour`	Create contour graph of matrix
`contour3`	Create 3-D contour plot
`contourc`	Create two-level contour plot computation
`contourf`	Create filled 2-D contour plot
`ctranspose`	Complex conjugate transpose of `fi` object
`dec`	Unsigned decimal representation of stored integer of `fi` object
`diag`	Diagonal matrices or diagonals of matrix
`disp`	Display object
`double`	Double-precision floating-point real-world value of `fi` object
`end`	Last index of array
`eps`	Quantized relative accuracy for `fi` or `quantizer` objects
`eq`	Determine whether real-world values of two `fi` objects are equal
`errorbar`	Plot error bars along curve
`etreeplot`	Plot elimination tree
`ezcontour`	Easy-to-use contour plotter
`ezcontourf`	Easy-to-use filled contour plotter
`ezmesh`	Easy-to-use 3-D mesh plotter
`ezplot`	Easy-to-use function plotter
`ezplot3`	Easy-to-use 3-D parametric curve plotter
`ezpolar`	Easy-to-use polar coordinate plotter

ezsurf	Easy-to-use 3-D colored surface plotter
ezsurfc	Easy-to-use combination surface/contour plotter
feather	Plot velocity vectors
fi	Construct fi object
fimath	Construct fimath object
flipdim	Flip array along specified dimension
fliplr	Flip matrix left to right
flipud	Flip matrix up to down
fplot	Plot function between specified limits
ge	Determine whether real-world value of one fi object is greater than or equal to another
get	Property values of object
gplot	Plot set of nodes using adjacency matrix
gt	Determine whether real-world value of one fi object is greater than another
hankel	Hankel matrix
hex	Hexadecimal representation of stored integer of fi object
hist	Create histogram plot
histc	Histogram count
horzcat	Horizontally concatenate multiple fi objects
imag	Imaginary part of complex number
innerprodintbits	Number of integer bits needed for fixed-point inner product

`int`	Smallest built-in integer in which stored integer value of `fi` object will fit
`int16`	Stored integer value of `fi` object as built-in `int16`
`int32`	Stored integer value of `fi` object as built-in `int32`
`int8`	Stored integer value of `fi` object as built-in `int8`
`intmax`	Largest positive stored integer value representable by `numerictype` of `fi` object
`intmin`	Smallest stored integer value representable by `numerictype` of `fi` object
`ipermute`	Inverse permute dimensions of multidimensional array
`iscolumn`	Determine whether `fi` object is column vector
`isempty`	Determine whether array is empty
`isequal`	Determine whether real-world values of two `fi` objects are equal, or determine whether properties of two `fimath`, `numerictype`, or `quantizer` objects are equal
`isfi`	Determine whether variable is `fi` object
`isfinite`	Determine whether array elements are finite
`isinf`	Determine whether array elements are infinite
`isnan`	Determine whether array elements are NaN

`isnumeric`	Determine whether input is numeric array
`isobject`	Determine whether input is MATLAB OOPS object
`ispropequal`	Determine whether properties of two `fi` objects are equal
`isreal`	Determine whether array elements are real
`isrow`	Determine whether `fi` object is row vector
`isscalar`	Determine whether input is scalar
`issigned`	Determine whether `fi` object is signed
`isvector`	Determine whether input is vector
`le`	Determine whether real-world value of `fi` object is less than or equal to another
`length`	Vector length
`line`	Create line object
`logical`	Convert numeric values to logical
`lowerbound`	Lower bound of range of `fi` object
`lsb`	Scaling of least significant bit of `fi` object
`lt`	Determine whether real-world value of one `fi` object is less than another
`max`	Largest element in array of `fi` objects
`mesh`	Create mesh plot
`meshc`	Create mesh plot with contour plot
`meshz`	Create mesh plot with curtain plot

`min`	Smallest element in array of `fi` objects
`minus`	Matrix difference between `fi` objects
`mtimes`	Matrix product of `fi` objects
`ndims`	Number of array dimensions
`ne`	Determine whether real-world values of two `fi` objects are not equal
`not`	Find logical NOT of array or scalar input
`numberofelements`	Number of data elements in `fi` array
`numerictype`	Construct `numerictype` object
`oct`	Octal representation of stored integer of `fi` object
`or`	Find logical OR of array or scalar inputs
`patch`	Create patch graphics object
`pcolor`	Create pseudocolor plot
`permute`	Rearrange dimensions of multidimensional array
`plot`	Create linear 2-D plot
`plot3`	Create 3-D line plot
`plotmatrix`	Draw scatter plots
`plotyy`	Create graph with y-axes on right and left sides
`plus`	Matrix sum of `fi` objects
`polar`	Plot polar coordinates
`pow2`	Multiply by 2^K
`quantizer`	Construct `quantizer` object
`quiver`	Create quiver or velocity plot

`quiver3`	Create 3-D quiver or velocity plot
`range`	Numerical range of `fi` or `quantizer` object
`real`	Real part of complex number
`realmax`	Largest positive fixed-point value or quantized number
`realmin`	Smallest positive normalized fixed-point value or quantized number
`repmat`	Replicate and tile array
`rescale`	Change scaling of `fi` object
`reshape`	Reshape array
`rgbplot`	Plot colormap
`ribbon`	Create ribbon plot
`rose`	Create angle histogram
`scatter`	Create scatter or bubble plot
`scatter3`	Create 3-D scatter or bubble plot
`sdec`	Signed decimal representation of stored integer of `fi` object
`shiftdim`	Shift dimensions
`sign`	Perform signum function on array
`single`	Single-precision floating-point real-world value of `fi` object
`size`	Array dimensions
`slice`	Create volumetric slice plot
`spy`	Visualize sparsity pattern
`stairs`	Create stairstep graph
`stem`	Plot discrete sequence data
`stem3`	Plot 3-D discrete sequence data

`streamribbon`	Create 3-D stream ribbon plot
`streamslice`	Draw streamlines in slice planes
`streamtube`	Create 3-D stream tube plot
`stripscaling`	Stored integer of `fi` object
`subsasgn`	Subscripted assignment
`subsref`	Subscripted reference
`sum`	Sum of array elements
`surf`	Create 3-D shaded surface plot
`surfc`	Create 3-D shaded surface plot with contour plot
`surfl`	Create surface plot with colormap-based lighting
`surfnorm`	Compute and display 3-D surface normals
`text`	Create text object in current axes
`times`	Element-by-element multiplication of `fi` objects
`toeplitz`	Create Toeplitz matrix
`transpose`	Transpose operation
`treeplot`	Plot picture of tree
`tril`	Lower triangular part of matrix
`trimesh`	Create triangular mesh plot
`triplot`	Create 2-D triangular plot
`trisurf`	Create triangular surface plot
`triu`	Upper triangular part of matrix
`uint16`	Stored integer value of `fi` object as built-in `uint16`
`uint32`	Stored integer value of `fi` object as built-in `uint32`

`uint8`	Stored integer value of `fi` object as built-in `uint8`
`uminus`	Negate elements of `fi` object array
`uplus`	Unary plus
`upperbound`	Upper bound of range of `fi` object
`vertcat`	Vertically concatenate multiple `fi` objects
`voronoi`	Create Voronoi diagram
`voronoin`	Create n-D Voronoi diagram
`waterfall`	Create waterfall plot
`xlim`	Set or query x-axis limits
`ylim`	Set or query y-axis limits
`zlim`	Set or query z-axis limits

fimath Object Functions

`add`	Add two objects using `fimath` object
`disp`	Display object
`fimath`	Construct `fimath` object
`isequal`	Determine whether real-world values of two `fi` objects are equal, or determine whether properties of two `fimath`, `numerictype`, or `quantizer` objects are equal
`isfimath`	Determine whether variable is `fimath` object
`mpy`	Multiply two objects using `fimath` object
`sub`	Subtract two objects using `fimath` object

fipref Object Functions

`disp`	Display object
`fipref`	Construct `fipref` object
`reset`	Reset objects to initial conditions
`savefipref`	Save `fi` preferences for next MATLAB session

numerictype Object Functions

`disp`	Display object
`divide`	Divide two objects
`isequal`	Determine whether real-world values of two `fi` objects are equal, or determine whether properties of two `fimath`, `numerictype`, or `quantizer` objects are equal
`isnumeric`	Determine whether input is numeric array

quantizer Object Functions

bin2num	Convert two's complement binary string to number using quantizer object
copyobj	Make independent copy of quantizer object
denormalmax	Largest denormalized quantized number for quantizer object
denormalmin	Smallest denormalized quantized number for quantizer object
disp	Display object
eps	Quantized relative accuracy for fi or quantizer objects
exponentbias	Exponent bias for quantizer object
exponentlength	Exponent length of quantizer object
exponentmax	Maximum exponent for quantizer object
exponentmin	Minimum exponent for quantizer object
fractionlength	Fraction length of quantizer object
get	Property values of object
hex2num	Convert hexadecimal string to number using quantizer object
isequal	Determine whether real-world values of two fi objects are equal, or determine whether properties of two fimath, numerictype, or quantizer objects are equal
length	Vector length
max	Largest element in array of fi objects

`min`	Smallest element in array of `fi` objects
`noperations`	Number of operations
`noverflows`	Number of overflows
`num2bin`	Convert number to binary string using `quantizer` object
`num2hex`	Convert number to hexadecimal equivalent using `quantizer` object
`num2int`	Convert number to signed integer
`nunderflows`	Number of underflows
`quantize`	Apply `quantizer` object to data
`quantizer`	Construct `quantizer` object
`randquant`	Generate uniformly distributed, quantized random number using `quantizer` object
`range`	Numerical range of `fi` or `quantizer` object
`realmax`	Largest positive fixed-point value or quantized number
`realmin`	Smallest positive normalized fixed-point value or quantized number
`reset`	Reset objects to initial conditions
`round`	Round input data using `quantizer` object without checking for overflow
`set`	Set or display property values for `quantizer` objects
`tostring`	Convert `quantizer` object to string
`wordlength`	Word length of `quantizer` object

Functions — Alphabetical List

abs

Purpose

Absolute value of `fi` object

Syntax

```
c = abs(a)
```

Description

`c = abs(a)` returns the absolute value of `fi` object `a`.

When the object `a` is real and has a signed data type, the absolute value of the most negative value is problematic since it is not representable. In this case, the absolute value saturates to the most positive value representable by the data type if the `OverflowMode` property is set to `saturate`. If `OverflowMode` is `wrap`, the absolute value of the most negative value has no effect.

`abs` does not support complex inputs.

Examples

The following example shows the difference between the absolute value results for the most negative value representable by a signed data type when `OverflowMode` is `saturate` or `wrap`.

```
P = fipref('NumericTypeDisplay','full',...
           'FimathDisplay','full');
a = fi(-128)

a =

  -128

          DataTypeMode: Fixed-point: binary point scaling
                Signed: true
            WordLength: 16
        FractionLength: 8

             RoundMode: nearest
          OverflowMode: saturate
           ProductMode: FullPrecision
  MaxProductWordLength: 128
               SumMode: FullPrecision
      MaxSumWordLength: 128
```

```
         CastBeforeSum: true

abs(a)

ans =

  127.9961

          DataTypeMode: Fixed-point: binary point scaling
                Signed: true
            WordLength: 16
        FractionLength: 8

             RoundMode: nearest
          OverflowMode: saturate
           ProductMode: FullPrecision
  MaxProductWordLength: 128
               SumMode: FullPrecision
      MaxSumWordLength: 128
         CastBeforeSum: true
a.OverflowMode = 'wrap'

a =

  -128

          DataTypeMode: Fixed-point: binary point scaling
                Signed: true
            WordLength: 16
        FractionLength: 8

             RoundMode: nearest
          OverflowMode: wrap
           ProductMode: FullPrecision
  MaxProductWordLength: 128
               SumMode: FullPrecision
```

```
         MaxSumWordLength: 128
            CastBeforeSum: true

abs(a)

ans =

  -128

             DataTypeMode: Fixed-point: binary point scaling
                   Signed: true
               WordLength: 16
           FractionLength: 8

                RoundMode: nearest
             OverflowMode: wrap
              ProductMode: FullPrecision
     MaxProductWordLength: 128
                  SumMode: FullPrecision
         MaxSumWordLength: 128
            CastBeforeSum: true
```

Purpose Add two objects using `fimath` object

Syntax

```
c = F.add(a,b)
```

Description

`c = F.add(a,b)` adds objects `a` and `b` using `fimath` object `F`. This is helpful in cases when you want to override the `fimath` objects of `a` and `b`, or if the `fimath` objects of `a` and `b` are different.

`a` and `b` must have the same dimensions unless one is a scalar. If either `a` or `b` is scalar, then `c` has the dimensions of the nonscalar object.

If either `a` or `b` is a `fi` object, and the other is a MATLAB built-in numeric type, then the built-in object is cast to the word length of the `fi` object, preserving best-precision fraction length.

Examples

In this example, `c` is the 32-bit sum of `a` and `b` with fraction length 16:

```
a = fi(pi);
b = fi(exp(1));
F = fimath('SumMode','SpecifyPrecision','SumWordLength',
    32,'SumFractionLength',16);
c = F.add(a,b)

c =

    5.8599


          DataTypeMode: Fixed-point: binary point scaling
                Signed: true
            WordLength: 32
        FractionLength: 16

             RoundMode: nearest
          OverflowMode: saturate
           ProductMode: FullPrecision
  MaxProductWordLength: 128
               SumMode: SpecifyPrecision
```

```
         SumWordLength: 32
     SumFractionLength: 16
         CastBeforeSum: true
```

Algorithm

`c = F.add(a,b)` is equivalent to

```
a.fimath = F;
b.fimath = F;
c = a + b;
```

except that the `fimath` properties of `a` and `b` are not modified when you use the functional form.

See Also

`divide`, `fi`, `fimath`, `mpy`, `numerictype`, `sub`, `sum`

Purpose Determine whether all array elements are nonzero

Description Refer to the MATLAB `all` reference page for more information.

Purpose Find logical AND of array or scalar inputs

Description Refer to the MATLAB `and` reference page for more information.

Purpose Determine whether any array elements are nonzero

Description Refer to the MATLAB `any` reference page for more information.

Purpose Create filled area 2-D plot

Description Refer to the MATLAB `area` reference page for more information.

Purpose Create vertical bar graph

Description Refer to the MATLAB `bar` reference page for more information.

barh

Purpose Create horizontal bar graph

Description Refer to the MATLAB `barh` reference page for more information.

Purpose

Binary representation of stored integer of `fi` object

Syntax

`bin(a)`

Description

Fixed-point numbers can be represented as

$$\textit{real-world value} = 2^{-\textit{fraction length}} \times \textit{stored integer}$$

or, equivalently,

$$\textit{real-world value} = (\textit{slope} \times \textit{stored integer}) + \textit{bias}$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

`bin(a)` returns the stored integer of `fi` object `a` in unsigned binary format as a string.

Examples

The following code

```
a = fi([-1 1],1,8,7);
bin(a)
```

returns

```
10000000   01111111
```

See Also

`dec`, `hex`, `int`, `oct`

bin2num

Purpose

Convert two's complement binary string to number using `quantizer` object

Syntax

`y = bin2num(q,b)`

Description

`y = bin2num(q,b)` uses the properties of `quantizer` object `q` to convert binary string `b` to numeric array `y`. When `b` is a cell array containing binary strings, `y` is a cell array of the same dimension containing numeric arrays. The fixed-point binary representation is two's complement. The floating-point binary representation is in IEEE Standard 754 style.

`bin2num` and `num2bin` are inverses of one another. Note that `num2bin` always returns the strings in a column.

Examples

Create a `quantizer` object and an array of numeric strings. Convert the numeric strings to binary strings, then use `bin2num` to convert them back to numeric strings.

```
q=quantizer([4 3]);
[a,b]=range(q);
x=(b:-eps(q):a)';
b = num2bin(q,x)

b =

0111
0110
0101
0100
0011
0010
0001
0000
1111
1110
1101
```

```
1100
1011
1010
1001
1000
```

`bin2num` performs the inverse operation of `num2bin`.

```
y=bin2num(q,b)

y =

    0.8750
    0.7500
    0.6250
    0.5000
    0.3750
    0.2500
    0.1250
         0
   -0.1250
   -0.2500
   -0.3750
   -0.5000
   -0.6250
   -0.7500
   -0.8750
   -1.0000
```

See Also

`hex2num`, `num2bin`, `num2hex`, `num2int`

bitand

Purpose Bitwise AND of two `fi` objects

Syntax

```
c = bitand(a, b)
```

Description `c = bitand(a, b)` returns the bitwise AND of `fi` objects `a` and `b`.

The `fimath` and the `numerictype` objects of `a` and `b` must be identical. If the `numerictype` is signed, then the bit representation of the stored integer is in two's complement representation.

`a` and `b` must have the same dimensions unless one is a scalar.

`bitand` only supports `fi` objects with fixed-point data types.

See Also `bitcmp`, `bitget`, `bitor`, `bitset`, `bitxor`

Purpose

Bitwise complement of `fi` object

Syntax

```
c = bitcmp(a)
```

Description

`c = bitcmp(a)` returns the bitwise complement of `fi` object `a`. If `a` has a signed `numerictype`, then the bit representation of the stored integer is in two's complement representation.

`bitcmp` only supports `fi` objects with fixed-point data types.

See Also

`bitand`, `bitget`, `bitor`, `bitset`, `bitxor`

bitget

Purpose

Bit at certain position

Syntax

```
c = bitget(a, bit)
```

Description

`c = bitget(a, bit)` returns the value of the bit at position `bit` in `a`. `bit` must be a number between `1` and the word length of `a`, inclusive. If `a` has a signed `numerictype`, then the bit representation of the stored integer is in two's complement representation.

`bitget` only supports `fi` objects with fixed-point data types.

See Also

`bitand`, `bitcmp`, `bitor`, `bitset`, `bitxor`

Purpose

Bitwise `OR` of two `fi` objects

Syntax

```
c = bitor(a, b)
```

Description

`c = bitor(a, b)` returns the bitwise `OR` of `fi` objects `a` and `b`.

The `fimath` and the `numerictype` objects of `a` and `b` must be identical. If the `numerictype` is signed, then the bit representation of the stored integer is in two's complement representation.

`a` and `b` must have the same dimensions unless one is a scalar.

`bitor` only supports `fi` objects with fixed-point data types.

See Also

`bitand`, `bitcmp`, `bitget`, `bitset`, `bitxor`

Purpose

Set bit at certain position

Syntax

```
c = bitset(a, bit)
c = bitset(a, bit, v)
```

Description

`c = bitset(a, bit)` sets bit position `bit` in `a` to `1` (on).

`c = bitset(a, bit, v)` sets bit position `bit` in `a` to `v`. `v` must be `0` (off) or `1` (on). Any value `v` other than `0` is automatically set to `1`.

`bit` must be a number between `1` and the word length of `a`, inclusive. If `a` has a signed `numerictype`, then the bit representation of the stored integer is in two's complement representation.

`bitset` only supports `fi` objects with fixed-point data types.

See Also

`bitand`, `bitcmp`, `bitget`, `bitor`, `bitxor`

Purpose

Shift bits specified number of places

Syntax

```
c = bitshift(a, k)
```

Description

`c = bitshift(a, k)` returns the value of `a` shifted by `k` bits.

`fi` object `a` can be any fixed-point numeric type. The `OverflowMode` and `RoundMode` properties are obeyed.

`bitshift` only supports `fi` objects with fixed-point data types.

Example

This example highlights how changing the `OverflowMode` property of the `fimath` object can change the results returned by the `bitshift` function. Consider the following signed fixed-point `fi` object with a value of 3, word length 16, and fraction length 0:

```
a = fi(3,1,16,0);
```

By default, the `OverflowMode` `fimath` property is `saturate`. When `a` is shifted such that it overflows, it is saturated to the maximum possible value:

```
for k=0:16,b=bitshift(a,k);...
disp([num2str(k,'%02d'),'. ',bin(b)]);end

00. 0000000000000011
01. 0000000000000110
02. 0000000000001100
03. 0000000000011000
04. 0000000000110000
05. 0000000001100000
06. 0000000011000000
07. 0000000110000000
08. 0000001100000000
09. 0000011000000000
10. 0000110000000000
11. 0001100000000000
12. 0011000000000000
```

```
13. 0110000000000000
14. 0111111111111111
15. 0111111111111111
16. 0111111111111111
```

Now change `OverflowMode` to `wrap`. In this case, most significant bits shift off the “top” of `a` until the value is zero:

```
a = fi(3,1,16,0,'OverflowMode','wrap');
for k=0:16,b=bitshift(a,k);...
disp([num2str(k,'%02d'),'. ',bin(b)]);end

00. 0000000000000011
01. 0000000000000110
02. 0000000000001100
03. 0000000000011000
04. 0000000000110000
05. 0000000001100000
06. 0000000011000000
07. 0000000110000000
08. 0000001100000000
09. 0000011000000000
10. 0000110000000000
11. 0001100000000000
12. 0011000000000000
13. 0110000000000000
14. 1100000000000000
15. 1000000000000000
16. 0000000000000000
```

See Also

`bitand`, `bitcmp`, `bitget`, `bitor`, `bitset`, `bitxor`

Purpose Bitwise exclusive `OR` of two `fi` objects

Syntax

```
c = bitxor(a, b)
```

Description `c = bitxor(a, b)` returns the bitwise exclusive `OR` of `fi` objects `a` and `b`.

The `fimath` and the `numerictype` objects of `a` and `b` must be identical. If the `numerictype` is signed, then the bit representation of the stored integer is in two's complement representation.

`a` and `b` must have the same dimensions unless one is a scalar.

`bitxor` only supports `fi` objects with fixed-point data types.

See Also `bitand`, `bitcmp`, `bitget`, `bitor`, `bitset`

buffer

Purpose Buffer signal vector into matrix of data frames

Description Refer to Signal Processing Toolbox `buffer` reference page for more information.

Purpose Create contour plot elevation labels

Description Refer to the MATLAB `clabel` reference page for more information.

comet

Purpose Create 2-D comet plot

Description Refer to the MATLAB `comet` reference page for more information.

Purpose Create 3-D comet plot

Description Refer to the MATLAB `comet3` reference page for more information.

Purpose Plot arrows emanating from origin

Description Refer to the MATLAB `compass` reference page for more information.

Purpose

Construct complex `fi` object from real and imaginary parts

Syntax

```
c = complex(a,b)
c = complex(a)
```

Description

The `complex` function constructs a complex `fi` object from real and imaginary parts.

`c = complex(a,b)` returns the complex result `a + bi`, where `a` and `b` are identically sized real N-D arrays, matrices, or scalars of the same data type. When `b` is all zero, `c` is complex with an all-zero imaginary part. This is in contrast to the addition of `a` + 0i, which returns a strictly real result.

`c = complex(a)` for a real `fi` object `a` returns the complex result `a + bi` with real part `a` and an all-zero imaginary part. Even though its imaginary part is all zero, `c` is complex.

The `numerictype` and `fimath` objects of the leftmost input that is a `fi` object are applied to the output `c`.

See Also

`imag`, `real`

coneplot

Purpose Plot velocity vectors as cones in 3-D vector field

Description Refer to the MATLAB `coneplot` reference page for more information.

Purpose Complex conjugate of `fi` object

Syntax `conj(a)`

Description `conj(a)` is the complex conjugate of `fi` object `a`.

When `a` is complex,

$$\mathrm{conj}(a) = \mathrm{real}(a) - i \times \mathrm{imag}(a)$$

The `numerictype` and `fimath` objects of the input `a` are applied to the output.

See Also `complex`, `imag`, `real`

Purpose

Create contour graph of matrix

Description

Refer to the MATLAB `contour` reference page for more information.

Purpose Create 3-D contour plot

Description Refer to the MATLAB `contour3` reference page for more information.

Purpose Create two-level contour plot computation

Description Refer to the MATLAB `contourc` reference page for more information.

Purpose Create filled 2-D contour plot

Description Refer to the MATLAB `contourf` reference page for more information.

convergent

Purpose

Apply convergent rounding

Syntax

```
convergent(x)
```

Description

`convergent(x)` rounds the elements of `x` to the nearest integer, except in a tie, then rounds to the nearest even integer.

Examples

MATLAB `round` and `convergent` differ in the way they treat values whose fractional part is 0.5. In `round`, every tie is rounded up in absolute value. `convergent` rounds ties to the nearest even integer.

```
x=[-3.5:3.5]';
[x convergent(x) round(x)]
ans =

   -3.5000   -4.0000   -4.0000
   -2.5000   -2.0000   -3.0000
   -1.5000   -2.0000   -2.0000
   -0.5000         0   -1.0000
    0.5000         0    1.0000
    1.5000    2.0000    2.0000
    2.5000    2.0000    3.0000
    3.5000    4.0000    4.0000
```

Purpose Make independent copy of quantizer object

Syntax

```
q1 = copyobj(q)
[q1,q2,...] = copyobj(obja,objb,...)
```

Description q1 = copyobj(q) makes a copy of quantizer object q and returns it in q1.

[q1,q2,...] = copyobj(obja,objb,...)copies obja into q1, objb into q2, and so on.

Using copyobj to copy a quantizer object is not the same as using the command syntax q1 = q to copy a quantizer object. quantizer objects have memory (their read-only properties). When you use copyobj, the resulting copy is independent of the original item; it does not share the original object's memory, such as the values of the properties min, max, noverflows, or noperations. Using q1 = q creates a new object that is an alias for the original and shares the original object's memory, and thus its property values.

Examples

```
q = quantizer('CoefficientFormat',[8 7]);
q1 = copyobj(q);
```

See Also quantizer, get, set

ctranspose

Purpose

Complex conjugate transpose of `fi` object

Syntax

```
ctranspose(a)
```

Description

`ctranspose(a)` returns the complex conjugate transpose of `fi` object `a`. It is also called for the syntax `a'`.

See Also

`transpose`

Purpose

Unsigned decimal representation of stored integer of `fi` object

Syntax

`dec(a)`

Description

Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times stored\ integer$$

or, equivalently,

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

`dec(a)` returns the stored integer of `fi` object `a` in unsigned decimal format as a string.

Examples

The code

```
a = fi([-1 1],1,8,7);
dec(a)
```

returns

```
128   127
```

See Also

`bin`, `hex`, `int`, `oct`, `sdec`

denormalmax

Purpose

Largest denormalized quantized number for `quantizer` object

Syntax

```
x = denormalmax(q)
```

Description

`x = denormalmax(q)` is the largest positive denormalized quantized number where `q` is a `quantizer` object. Anything larger than `x` is a normalized number. Denormalized numbers apply only to floating-point format. When `q` represents fixed-point numbers, this function returns `eps(q)`.

Examples

```
q = quantizer('float',[6 3]);
x = denormalmax(q)

x =

    0.1875
```

Algorithm

When `q` is a floating-point `quantizer` object,

```
denormalmax(q) = realmin(q) - denormalmin(q)
```

When `q` is a fixed-point `quantizer` object,

```
denormalmax(q) = eps(q)
```

See Also

`denormalmin`, `eps`, `quantizer`

Purpose

Smallest denormalized quantized number for `quantizer` object

Syntax

```
x = denormalmin(q)
```

Description

`x = denormalmin(q)` is the smallest positive denormalized quantized number where `q` is a `quantizer` object. Anything smaller than `x` underflows to zero with respect to the `quantizer` object `q`. Denormalized numbers apply only to floating-point format. When `q` represents a fixed-point number, `denormalmin` returns `eps(q)`.

Examples

```
q = quantizer('float',[6 3]);
denormalmin(q)

ans =

    0.0625
```

Algorithm

When `q` is a floating-point `quantizer` object,

$$x = 2^{Emin - f}$$

where E_{min} is equal to `exponentmin(q)`.

When `q` is a fixed-point `quantizer` object,

$$x = \mathrm{eps(q)} = 2^{-f}$$

where f is equal to `fractionlength(q)`.

See Also

`denormalmax`, `eps`, `quantizer`

diag

Purpose Diagonal matrices or diagonals of matrix

Description Refer to the MATLAB `diag` reference page for more information.

Purpose Display object

Description Refer to the MATLAB `disp` reference page for more information.

divide

Purpose

Divide two objects

Syntax

```
c = divide(T,a,b)
c = T.divide(a,b)
```

Description

`c = divide(T,a,b)` and `c = T.divide(a,b)` perform division on the elements of `a` by the elements of `b`. The result `c` has the `numerictype` object `T`.

`a` and `b` must have the same dimensions unless one is a scalar. If either `a` or `b` is scalar, then `c` has the dimensions of the nonscalar object.

If either `a` or `b` is a `fi` object, and the other is a MATLAB built-in numeric type, then the built-in object is cast to the word length of the `fi` object, preserving best-precision fraction length.

If `a` and `b` are both MATLAB built-in doubles or singles, then `c` is the floating-point quotient `a./b`, and `numerictype T` is ignored.

Note The `divide` function is not currently supported for [Slope Bias] signals.

Examples

This example highlights the precision of the `fi divide` function.

First, create an unsigned `fi` object with an 80-bit word length and 2^-83 scaling, which puts the leading 1 of the representation into the most significant bit. Initialize the object with double-precision floating-point value 0.1, and examine the binary representation:

```
P =
fipref('NumberDisplay','bin',...
       'NumericTypeDisplay','short',...
       'FimathDisplay','none');
a = fi(0.1, false, 80, 83)

a =
```

```
110011001100110011001100110011001100110011001100110011010000
0000000000000000000000000
(bin)
      u80,83
110011001100110011001100110011001100110011001100110011001100
11001100110011001100110
```

Notice that the infinite repeating representation is truncated after 52 bits, because the mantissa of an IEEE standard double-precision floating-point number has 52 bits.

Contrast the above to calculating 1/10 in fixed-point arithmetic with the quotient set to the same numeric type as before:

```
T = numerictype('Signed',false,'WordLength',80,...
           'FractionLength',83);
a = fi(1);
b = fi(10);
c = T.divide(a,b);
c.bin

ans =

110011001100110011001100110011001100110011001100110011001100
11001100110011001100110
```

Notice that when you use the `divide` function, the quotient is calculated to the full 80 bits, regardless of the precision of `a` and `b`. Thus, the `fi` object `c` represents 1/10 more precisely than IEEE standard double-precision floating-point number can.

With 1000 bits of precision,

```
T = numerictype('Signed',false,'WordLength',1000,...
           'FractionLength',1003);
a = fi(1);
b = fi(10);
c = T.divide(a,b);
```

```
c.bin

ans =

11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100
110011001100110011001100110011001100110011001100
```

See Also

`add`, `fi`, `fimath`, `mpy`, `numerictype`, `sub`, `sum`

Purpose Double-precision floating-point real-world value of `fi` object

Syntax `double(a)`

Description Fixed-point numbers can be represented as

$$\textit{real-world value} = 2^{-\textit{fraction length}} \times \textit{stored integer}$$

or, equivalently,

$$\textit{real-world value} = (\textit{slope} \times \textit{stored integer}) + \textit{bias}$$

`double(a)` returns the real-world value of a `fi` object in double-precision floating point.

See Also `single`

Purpose Last index of array

Description Refer to the MATLAB `end` reference page for more information.

Purpose

Quantized relative accuracy for `fi` or `quantizer` objects

Syntax

```
eps(obj)
```

Description

`eps(obj)` returns the value of the least significant bit of the value of the `fi` object or `quantizer` object `obj`. The result of this function is equivalent to that given by the Fixed-Point Toolbox `lsb` function.

See Also

`intmax`, `intmin`, `lowerbound`, `lsb`, `range`, `realmax`, `realmin`, `upperbound`

Purpose

Determine whether real-world values of two `fi` objects are equal

Syntax

```
c = eq(a,b)
a == b
```

Description

`c = eq(a,b)` is called for the syntax `a == b` when a or `b` is a `fi` object. `a` and `b` must have the same dimensions unless one is a scalar. A scalar can be compared with another object of any size.

`a == b` does an element-by-element comparison between `a` and `b` and returns a matrix of the same size with elements set to `1` where the relation is true, and `0` where the relation is false.

See Also

`ge`, `gt`, `isequal`, `le`, `lt`, `ne`

Purpose Plot error bars along curve

Description Refer to the MATLAB `errorbar` reference page for more information.

etreeplot

Purpose Plot elimination tree

Description Refer to the MATLAB `etreeplot` reference page for more information.

Purpose

Exponent bias for `quantizer` object

Syntax

```
b = exponentbias(q)
```

Description

`b = exponentbias(q)` returns the exponent bias of the `quantizer` object `q`. For fixed-point `quantizer` objects, `exponentbias(q)` returns 0.

Examples

```
q = quantizer('double');
b = exponentbias(q)

b =

        1023
```

Algorithm

For floating-point `quantizer` objects,

$$b = 2^{e-1} - 1$$

where `e` = `eps(q)`, and `exponentbias` is the same as the exponent maximum.

For fixed-point `quantizer` objects, `b = 0` by definition.

See Also

`eps`, `exponentlength`, `exponentmax`, `exponentmin`

exponentlength

Purpose

Exponent length of `quantizer` object

Syntax

```
e = exponentlength(q)
```

Description

`e = exponentlength(q)` returns the exponent length of `quantizer` object `q`. When `q` is a fixed-point `quantizer` object, `exponentlength(q)` returns `0`. This is useful because exponent length is valid whether the `quantizer` object mode is floating point or fixed point.

Examples

```
q = quantizer('double');
e = exponentlength(q)

e =

    11
```

Algorithm

The exponent length is part of the format of a floating-point `quantizer` object `[w e]`. For fixed-point `quantizer` objects, $e = 0$ by definition.

See Also

`eps`, `exponentbias`, `exponentmax`, `exponentmin`

Purpose

Maximum exponent for `quantizer` object

Syntax

```
exponentmax(q)
```

Description

`exponentmax(q)` returns the maximum exponent for `quantizer` object `q`. When `q` is a fixed-point `quantizer` object, it returns `0`.

Examples

```
q = quantizer('double');
exponentmax(q)

ans =

        1023
```

Algorithm

For floating-point `quantizer` objects,

$$E_{max} = 2^{e-1} - 1$$

For fixed-point `quantizer` objects, $E_{max} = 0$ by definition.

See Also

`eps`, `exponentbias`, `exponentlength`, `exponentmin`

exponentmin

Purpose Minimum exponent for `quantizer` object

Syntax

```
emin = exponentmin(q)
```

Description `emin = exponentmin(q)` returns the minimum exponent for `quantizer` object `q`. If `q` is a fixed-point `quantizer` object, `exponentmin` returns 0.

Examples

```
q = quantizer('double');
emin = exponentmin(q)

emin =

      -1022
```

Algorithm For floating-point `quantizer` objects,

$$E_{min} = -2^{e-1} + 2$$

For fixed-point `quantizer` objects, $E_{min} = 0$.

See Also `eps`, `exponentbias`, `exponentlength`, `exponentmax`

Purpose Easy-to-use contour plotter

Description Refer to the MATLAB `ezcontour` reference page for more information.

Purpose Easy-to-use filled contour plotter

Description Refer to the MATLAB `ezcontourf` reference page for more information.

Purpose Easy-to-use 3-D mesh plotter

Description Refer to the MATLAB `ezmesh` reference page for more information.

ezplot

Purpose Easy-to-use function plotter

Description Refer to the MATLAB `ezplot` reference page for more information.

Purpose Easy-to-use 3-D parametric curve plotter

Description Refer to the MATLAB `ezplot3` reference page for more information.

Purpose Easy-to-use polar coordinate plotter

Description Refer to the MATLAB `ezpolar` reference page for more information.

Purpose Easy-to-use 3-D colored surface plotter

Description Refer to the MATLAB `ezsurf` reference page for more information.

ezsurfc

Purpose Easy-to-use combination surface/contour plotter

Description Refer to the MATLAB `ezsurfc` reference page for more information.

Purpose Plot velocity vectors

Description Refer to the MATLAB `feather` reference page for more information.

Purpose

Construct `fi` object

Syntax

```
a = fi
a = fi(v)
a = fi(v,s)
a = fi(v,s,w)
a = fi(v,s,w,f)
a = fi(v,s,w,slope,bias)
a = fi(v,s,w,slopeadjustmentfactor,fixedexponent,bias)
a = fi(v,T)
a = fi(v,F)
b = fi(a,F)
a = fi(v,T,F)
a = fi(v,s,F)
a = fi(v,s,w,F)
a = fi(v,s,w,f,F)
a = fi(v,s,w,slope,bias,F)
a = fi(v,s,w,slopeadjustmentfactor,fixedexponent,bias,F)
a = fi(...'PropertyName',PropertyValue...)
a = fi('PropertyName',PropertyValue...)
```

Description

You can use the `fi` constructor function in the following ways:

- `a = fi` is the default constructor and returns a `fi` object with no value, 16-bit word length, and 15-bit fraction length.
- `a = fi(v)` returns a signed fixed-point object with value `v`, 16-bit word length, and best-precision fraction length.
- `a = fi(v,s)` returns a fixed-point object with value `v`, signedness `s`, 16-bit word length, and best-precision fraction length. `s` can be `0` (false) for unsigned or `1` (true) for signed.
- `a = fi(v,s,w)` returns a fixed-point object with value `v`, signedness `s`, word length `w`, and best-precision fraction length.
- `a = fi(v,s,w,f)` returns a fixed-point object with value `v`, signedness `s`, word length `w`, and fraction length `f`.

- `a = fi(v,s,w,slope,bias)` returns a fixed-point object with value `v`, signedness `s`, word length `w`, `slope`, and `bias`.
- `a = fi(v,s,w,slopeadjustmentfactor,fixedexponent,bias)` returns a fixed-point object with value `v`, signedness `s`, word length `w`, `slopeadjustmentfactor`, `fixedexponent`, and `bias`.
- `a = fi(v,T)` returns a fixed-point object with value `v` and `embedded.numerictype` `T`. Refer to "Working with numerictype Objects" for more information on `numerictype` objects.
- `a = fi(v,F)` returns a fixed-point object with value `v`, `embedded.fimath` `F`, 16-bit word length, and best-precision fraction length. Refer to "Working with fimath Objects" for more information on `fimath` objects.
- `b = fi(a,F)` allows you to maintain the value and `numerictype` object of `fi` object `a`, while changing its `fimath` object to `F`.
- `a = fi(v,T,F)` returns a fixed-point object with value `v`, `embedded.numerictype` `T`, and `embedded.fimath` `F`.
- `a = fi(v,s,F)` returns a fixed-point object with value `v`, signedness `s`, 16-bit word length, best-precision fraction length, and `embedded.fimath` `F`.
- `a = fi(v,s,w,F)` returns a fixed-point object with value `v`, signedness `s`, word length `w`, best-precision fraction length, and `embedded.fimath` `F`.
- `a = fi(v,s,w,f,F)` returns a fixed-point object with value `v`, signedness `s`, word length `w`, fraction length `f`, and `embedded.fimath` `F`.
- `a = fi(v,s,w,slope,bias,F)` returns a fixed-point object with value `v`, signedness `s`, word length `w`, `slope`, `bias`, and `embedded.fimath` `F`.
- `a = fi(v,s,w,slopeadjustmentfactor,fixedexponent,bias,F)` returns a fixed-point object with value `v`, signedness `s`, word length `w`, `slopeadjustmentfactor`, `fixedexponent`, `bias`, and `embedded.fimath` `F`.

- `a = fi(...'PropertyName',PropertyValue...)` and `a = fi('PropertyName',PropertyValue...)` allow you to set fixed-point objects for a `fi` object by property name/property value pairs.

The `fi` object has the following three general types of properties:

- "Data Properties" on page 3-68
- "fimath Properties" on page 3-69
- "numerictype Properties" on page 3-70

Note These properties are described in detail in "fi Object Properties" on page 1-2 in the Properties Reference.

Data Properties

The data properties of a `fi` object are always writable.

- `bin` — Stored integer value of a `fi` object in binary
- `data` — Numerical real-world value of a `fi` object
- `dec` — Stored integer value of a `fi` object in decimal
- `double` — Real-world value of a `fi` object, stored as a MATLAB `double`
- `hex` — Stored integer value of a `fi` object in hexadecimal
- `int` — Stored integer value of a `fi` object, stored in a built-in MATLAB integer data type. You can also use `int8`, `int16`, `int32`, `uint8`, `uint16`, and `uint32` to get the stored integer value of a fi object in these formats
- `oct` — Stored integer value of a `fi` object in octal

These properties are described in detail in "fi Object Properties" on page 1-2.

fimath Properties

When you create a `fi` object, a `fimath` object is also automatically created as a property of the `fi` object.

- `fimath` — `fimath` object associated with a `fi` object

The following `fimath` properties are, by transitivity, also properties of a `fi` object. The properties of the `fimath` object listed below are always writable.

- `CastBeforeSum` — Whether both operands are cast to the sum data type before addition
- `MaxProductWordLength` — Maximum allowable word length for the product data type
- `MaxSumWordLength` — Maximum allowable word length for the sum data type
- `OverflowMode` — Overflow mode
- `ProductBias` — Bias of the product data type
- `ProductFixedExponent` — Fixed exponent of the product data type
- `ProductFractionLength` — Fraction length, in bits, of the product data type
- `ProductMode` — Defines how the product data type is determined
- `ProductSlope` — Slope of the product data type
- `ProductSlopeAdjustmentFactor` — Slope adjustment factor of the product data type
- `ProductWordLength` — Word length, in bits, of the product data type
- `RoundMode` — Rounding mode
- `SumBias` — Bias of the sum data type
- `SumFixedExponent` — Fixed exponent of the sum data type

- SumFractionLength — Fraction length, in bits, of the sum data type
- SumMode — Defines how the sum data type is determined
- SumSlope — Slope of the sum data type
- SumSlopeAdjustmentFactor — Slope adjustment factor of the sum data type
- SumWordLength — The word length, in bits, of the sum data type

These properties are described in detail in "fimath Object Properties" on page 1-6.

numerictype Properties

When you create a fi object, a numerictype object is also automatically created as a property of the fi object.

- numerictype — Object containing all the numeric type attributes of a fi object

The following numerictype properties are, by transitivity, also properties of a fi object. The properties of the numerictype object listed below are not writable once the fi object has been created. However, you can create a copy of a fi object with new values specified for the numerictype properties.

- Bias — Bias of a fi object
- DataType — Data type category associated with a fi object
- DataTypeMode — Data type and scaling mode of a fi object
- FixedExponent — Fixed-point exponent associated with a fi object
- SlopeAdjustmentFactor — Slope adjustment associated with a fi object
- FractionLength — Fraction length of the stored integer value of a fi object in bits
- Scaling — Fixed-point scaling mode of a fi object

- `Signed` — Whether a `fi` object is signed or unsigned
- `Slope` — Slope associated with a `fi` object
- `WordLength` — Word length of the stored integer value of a `fi` object in bits

These properties are described in detail in “numerictype Object Properties” on page 1-17.

Examples

Note For information about the display format of fi objects, refer to Display Settings.

Example 1

For example, the following creates a `fi` object with a value of `pi`, a word length of 8 bits, and a fraction length of 3 bits:

```
a = fi(pi, 1, 8, 3)

a =

    3.1250

          DataTypeMode: Fixed-point: binary point scaling
                Signed: true
            WordLength: 8
        FractionLength: 3
```

Example 2

The value `v` can also be an array:

```
a = fi((magic(3)/10), 1, 16, 12)

a =
```

```
        0.8000    0.1001    0.6001
        0.3000    0.5000    0.7000
        0.3999    0.8999    0.2000

              DataTypeMode: Fixed-point: binary point scaling
                    Signed: true
                WordLength: 16
            FractionLength: 12
```

Example 3

If you omit the argument `f`, it is set automatically to the best precision possible:

```
 a = fi(pi, 1, 8)

a =

    3.1563

          DataTypeMode: Fixed-point: binary point scaling
                Signed: true
            WordLength: 8
        FractionLength: 5
```

Example 4

If you omit `w` and `f`, they are set automatically to 16 bits and the best precision possible, respectively:

```
a = fi(pi, 1)

a =

    3.1416

          DataTypeMode: Fixed-point: binary point scaling
```

```
          Signed: true
      WordLength: 16
  FractionLength: 13
```

Example 5

You can use property name/property value pairs to set `fi` properties when you create the object:

```
a = fi(pi, 'roundmode', 'floor', 'overflowmode', 'wrap')

a =

    3.1415

           DataTypeMode: Fixed-point: binary point scaling
                 Signed: true
             WordLength: 16
         FractionLength: 13
```

See Also

`fimath`, `fipref`, `numerictype`, `quantizer`

fimath

Purpose

Construct `fimath` object

Syntax

```
F = fimath
F = fimath(...'PropertyName',PropertyValue...)
```

Description

You can use the `fimath` constructor function in the following ways:

- `F = fimath` creates a default `fimath` object.
- `F = fimath(...'PropertyName',PropertyValue...)` allows you to set the attributes of a `fimath` object using property name/property value pairs.

The properties of the `fimath` object are listed below. These properties are described in detail in "fimath Object Properties" on page 1-6 in the Properties Reference.

- `CastBeforeSum` — Whether both operands are cast to the sum data type before addition
- `MaxProductWordLength` — Maximum allowable word length for the product data type
- `MaxSumWordLength` — Maximum allowable word length for the sum data type
- `OverflowMode` — Overflow-handling mode
- `ProductBias` — Bias of the product data type
- `ProductFixedExponent` — Fixed exponent of the product data type
- `ProductFractionLength` — Fraction length, in bits, of the product data type
- `ProductMode` — Defines how the product data type is determined
- `ProductSlope` — Slope of the product data type
- `ProductSlopeAdjustmentFactor` — Slope adjustment factor of the product data type

- `ProductWordLength` — Word length, in bits, of the product data type
- `RoundMode` — Rounding mode
- `SumBias` — Bias of the sum data type
- `SumFixedExponent` — Fixed exponent of the sum data type
- `SumFractionLength` — Fraction length, in bits, of the sum data type
- `SumMode` — Defines how the sum data type is determined
- `SumSlope` — Slope of the sum data type
- `SumSlopeAdjustmentFactor` — Slope adjustment factor of the sum data type
- `SumWordLength` — Word length, in bits, of the sum data type

Examples

Example 1

Type

```
F = fimath
```

to create a default `fimath` object.

```
F = fimath

F =

               RoundMode: nearest
            OverflowMode: saturate
             ProductMode: FullPrecision
    MaxProductWordLength: 128
                 SumMode: FullPrecision
        MaxSumWordLength: 128
           CastBeforeSum: true
```

Example 2

You can set properties of `fimath` objects at the time of object creation by including properties after the arguments of the `fimath` constructor function. For example, to set the overflow mode to `saturate` and the rounding mode to `convergent`,

```
F = fimath('OverflowMode','saturate',...
           'RoundMode','convergent')

F =

             RoundMode: convergent
          OverflowMode: saturate
           ProductMode: FullPrecision
  MaxProductWordLength: 128
               SumMode: FullPrecision
      MaxSumWordLength: 128
         CastBeforeSum: true
```

See Also

`fi`, `fipref`, `numerictype`, `quantizer`

Purpose

Construct `fipref` object

Syntax

```
P = fipref
P = fipref(...'PropertyName',PropertyValue...)
```

Description

You can use the `fipref` constructor function in the following ways:

- `P = fipref` creates a default `fipref` object.
- `P = fipref(...'PropertyName',PropertyValue...)` allows you to set the attributes of a object using property name/property value pairs.

The properties of the `fipref` object are listed below. These properties are described in detail in "fipref Object Properties" on page 1-14.

- `FimathDisplay` — Display options for the `fimath` attributes of a `fi` object
- `DataTypeOverride` — Data type override options
- `LoggingMode` — Logging options for operations performed on `fi` objects
- `NumericTypeDisplay` — Display options for the numeric type attributes of a `fi` object
- `NumberDisplay` — Display options for the value of a `fi` object

Your `fipref` settings persist throughout your MATLAB session. Use `reset(fipref)` to return to the default settings during your session. Use `savefipref` to save your display preferences for subsequent MATLAB sessions.

Examples

Example 1

Type

```
P = fipref
```

to create a default fipref object.

```
P =

        NumberDisplay: 'RealWorldValue'
   NumericTypeDisplay: 'full'
        FimathDisplay: 'full'
          LoggingMode: 'Off'
     DataTypeOverride: 'ForceOff'
```

Example 2

You can set properties of fipref objects at the time of object creation by including properties after the arguments of the fipref constructor function. For example, to set NumberDisplay to bin and AttributesDisplay to short,

```
P =

        NumberDisplay: 'bin'
   NumericTypeDisplay: 'short'
        FimathDisplay: 'full'
          LoggingMode: 'Off'
     DataTypeOverride: 'ForceOff'
```

See Also

fi, fimath, numerictype, quantizer, savefipref

Purpose Flip array along specified dimension

Description Refer to the MATLAB `flipdim` reference page for more information.

fliplr

Purpose Flip matrix left to right

Description Refer to the MATLAB `fliplr` reference page for more information.

Purpose Flip matrix up to down

Description Refer to the MATLAB `flipud` reference page for more information.

fplot

Purpose Plot function between specified limits

Description Refer to the MATLAB `fplot` reference page for more information.

Purpose Fraction length of `quantizer` object

Syntax

```
fractionlength(q)
```

Description `fractionlength(q)` returns the fraction length of `quantizer` object `q`.

Algorithm For floating-point `quantizer` objects, $f = w - e - 1$, where w is the word length and e is the exponent length.

For fixed-point `quantizer` objects, f is part of the format [w f].

See Also `fi`, `numerictype`, `quantizer`, `wordlength`

Purpose Determine whether real-world value of one `fi` object is greater than or equal to another

Syntax

```
c = ge(a,b)
a >= b
```

Description `c = ge(a,b)` is called for the syntax `a >= b` when `a` or `b` is a `fi` object. `a` and `b` must have the same dimensions unless one is a scalar. A scalar can be compared with another object of any size.

`a >= b` does an element-by-element comparison between `a` and `b` and returns a matrix of the same size with elements set to `1` where the relation is true, and `0` where the relation is false.

See Also `eq`, `gt`, `le`, `lt`, `ne`

Purpose

Property values of object

Syntax

```
value = get(o,'propertyname')
structure = get(o)
```

Description

`value = get(o,'propertyname')` returns the property value of the property `'propertyname'` for the object `o`. If you replace the string `'propertyname'` by a cell array of a vector of strings containing property names, `get` returns a cell array of a vector of corresponding values.

`structure = get(o)` returns a structure containing the properties and states of object `o`.

`o` can be a `fi`, `fimath`, `fipref`, `numerictype`, or `quantizer` object.

See Also

`set`

gplot

Purpose Plot set of nodes using adjacency matrix

Description Refer to the MATLAB `gplot` reference page for more information.

Purpose

Determine whether real-world value of one `fi` object is greater than another

Syntax

```
c = gt(a,b)
a > b
```

Description

`c = gt(a,b)` is called for the syntax `a > b` when `a` or `b` is a `fi` object. `a` and `b` must have the same dimensions unless one is a scalar. A scalar can be compared with another object of any size.

`a > b` does an element-by-element comparison between `a` and `b` and returns a matrix of the same size with elements set to `1` where the relation is true, and `0` where the relation is false.

See Also

`eq`, `ge`, `le`, `lt`, `ne`

hankel

Purpose Hankel matrix

Description Refer to the MATLAB `hankel` reference page for more information.

Purpose Hexadecimal representation of stored integer of `fi` object

Syntax

```
hex(a)
```

Description Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times stored\ integer$$

or, equivalently,

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

`hex(a)` returns the stored integer of `fi` object `a` in hexadecimal format as a string.

Examples The following code

```
a = fi([-1 1],1,8,7);
hex(a)
```

returns

```
80   7f
```

See Also `bin`, `dec`, `int`, `oct`

hex2num

Purpose Convert hexadecimal string to number using `quantizer` object

Syntax

```
x = hex2num(q,h)
[x1,x2,...] = hex2num(q,h1,h2,...)
```

Description `x = hex2num(q,h)` converts hexadecimal string `h` to numeric matrix `x`. The attributes of the numbers in `x` are specified by `quantizer` object `q`. When `h` is a cell array containing hexadecimal strings, `hex2num` returns `x` as a cell array of the same dimension containing numbers. For fixed-point hexadecimal strings, `hex2num` uses two's complement representation. For floating-point strings, the representation is IEEE Standard 754 style.

When there are fewer hexadecimal digits than needed to represent the number, the fixed-point conversion zero-fills on the left. Floating-point conversion zero-fills on the right.

`[x1,x2,...]  = hex2num(q,h1,h2,...)` converts hexadecimal strings `h1`, `h2`,... to numeric matrices `x1`, `x2`,....

`hex2num` and `num2hex` are inverses of one another, with the distinction that `num2hex` returns the hexadecimal strings in a column.

Examples To create all the 4-bit fixed-point two's complement numbers in fractional form, use the following code.

```
q = quantizer([4 3]);
h = ['7 3 F B';'6 2 E A';'5 1 D 9';'4 0 C 8'];
x = hex2num(q,h)

x =

    0.8750    0.3750   -0.1250   -0.6250
    0.7500    0.2500   -0.2500   -0.7500
    0.6250    0.1250   -0.3750   -0.8750
    0.5000         0   -0.5000   -1.0000
```

See Also `bin2num`, `num2bin`, `num2hex`, `num2int`

Purpose Create histogram plot

Description Refer to the MATLAB `hist` reference page for more information.

Purpose Histogram count

Description Refer to the MATLAB `histc` reference page for more information.

Purpose

Horizontally concatenate multiple `fi` objects

Syntax

```
c = horzcat(a,b,...)
[a, b, ...]
```

Description

`c = horzcat(a,b,...)` is called for the syntax `[a, b, ...]` when any of `a`, `b`, ... , is a `fi` object.

`[a b, ...]` or `[a,b, ...]` is the horizontal concatenation of matrices `a` and `b`. `a` and `b` must have the same number of rows. Any number of matrices can be concatenated within one pair of brackets. N-D arrays are horizontally concatenated along the second dimension. The first and remaining dimensions must match.

Horizontal and vertical concatenation can be combined together as in `[1 2;3 4]`.

`[a b; c]` is allowed if the number of rows of `a` equals the number of rows of `b`, and if the number of columns of `a` plus the number of columns of `b` equals the number of columns of `c`.

The matrices in a concatenation expression can themselves be formed via a concatenation as in `[a b;[c d]]`.

Note The `fimath` and `numerictype` objects of a concatenated matrix of `fi` objects `c` are taken from the leftmost `fi` object in the list `(a,b,...)`.

See Also

`vertcat`

imag

Purpose Imaginary part of complex number

Description Refer to the MATLAB `imag` reference page for more information.

Purpose

Number of integer bits needed for fixed-point inner product

Syntax

```
innerprodintbits(a,b)
```

Description

`innerprodintbits(a,b)` computes the minimum number of integer bits necessary in the inner product of `a'*b` to guarantee that no overflows occur and to preserve best precision.

- `a` and `b` are `fi` vectors.
- The values of `a` are known.
- Only the numeric type of `b` is relevant. The values of `b` are ignored.

Examples

The primary use of this function is to determine the number of integer bits necessary in the output `Y` of an FIR filter that computes the inner product between constant coefficient row vector `B` and state column vector `Z`. For example,

```
for k=1:length(X);
  Z = [X(k);Z(1:end-1)];
  Y(k) = B * Z;
end
```

Algorithm

In general, an inner product grows `log2(n)` bits for vectors of length `n`. However, in the case of this function the vector `a` is known and its values do not change. This knowledge is used to compute the smallest number of integer bits that are necessary in the output to guarantee that no overflow will occur.

The largest gain occurs when the vector `b` has the same sign as the constant vector `a`. Therefore, the largest gain due to the vector `a` is `a*sign(a')`, which is equal to `sum(abs(a))`.

The overall number of integer bits necessary to guarantee that no overflow occurs in the inner product is computed by:

```
log2(sum(abs(a))
+ number of integer bits in b
```

```
+ 1 sign bit
```

Purpose Smallest built-in integer in which stored integer value of `fi` object will fit

Syntax `int(a)`

Description Fixed-point numbers can be represented as

$$\text{real-world value} = 2^{-\text{fraction length}} \times \text{stored integer}$$

or, equivalently,

$$\text{real-world value} = (\text{slope} \times \text{stored integer}) + \text{bias}$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

`int(a)` returns the smallest built-in integer of the data type in which the stored integer value of `fi` object `a` will fit.

The following table gives the return type of the `int` function.

Word Length	Return Type for Signed fi	Return Type for Unsigned fi
word length <= 8 bits	`int8`	`uint8`
8 bits < word length <= 16 bits	`int16`	`uint16`
16 bits < word length <= 32 bits	`int32`	`uint32`
32 < word length	`double`	`double`

Note When the word length is greater than 52 bits, the return value can have quantization error. For bit-true integer representation of very large word lengths, use `bin`, `oct`, `dec`, `hex`, or `sdec`.

See Also `int8`, `int16`, `int32`, `uint8`, `uint16`, `uint32`

Purpose Stored integer value of `fi` object as built-in `int8`

Syntax `int8(a)`

Description Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times stored\ integer$$

or, equivalently,

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

`int8(a)` returns the stored integer value of `fi` object `a` as a built-in `int8`. If the stored integer word length is too big for an `int8`, or if the stored integer is unsigned, the returned value saturates to an `int8`.

See Also `int`, `int16`, `int32`, `uint8`, `uint16`, `uint32`

Purpose Stored integer value of fi object as built-in int16

Syntax int16(a)

Description Fixed-point numbers can be represented as

$$\textit{real-world value} = 2^{-\textit{fraction length}} \times \textit{stored integer}$$

or, equivalently,

$$\textit{real-world value} = (\textit{slope} \times \textit{stored integer}) + \textit{bias}$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

int16(a) returns the stored integer value of fi object a as a built-in int16. If the stored integer word length is too big for an int16, or if the stored integer is unsigned, the returned value saturates to an int16.

See Also int, int8, int32, uint8, uint16, uint32

Purpose

Stored integer value of `fi` object as built-in `int32`

Syntax

`int32(a)`

Description

Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times stored\ integer$$

or, equivalently,

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

`int32(a)` returns the stored integer value of `fi` object `a` as a built-in `int32`. If the stored integer word length is too big for an `int32`, or if the stored integer is unsigned, the returned value saturates to an `int32`.

See Also

`int`, `int8`, `int16`, `uint8`, `uint16`, `uint32`

Purpose Largest positive stored integer value representable by `numerictype` of `fi` object

Syntax `x = intmax(a)`

Description `x = intmax(a)` returns the largest positive stored integer value representable by the `numerictype` of `a`.

See Also `eps`, `intmin`, `lowerbound`, `lsb`, `range`, `realmax`, `realmin`, `stripscaling`, `upperbound`

Purpose

Smallest stored integer value representable by `numerictype` of `fi` object

Syntax

`x = intmin(a)`

Description

`x = intmin(a)` returns the smallest stored integer value representable by the `numerictype` of `a`.

Examples

```
a = fi(pi, true, 16, 12);
x = intmin(a)

x =

      -32768

           DataTypeMode: Fixed-point: binary point scaling
                 Signed: true
             WordLength: 16
         FractionLength: 0
```

See Also

`eps`, `intmax`, `lowerbound`, `lsb`, `range`, `realmax`, `realmin`, `stripscaling`, `upperbound`

ipermute

Purpose Inverse permute dimensions of multidimensional array

Description Refer to the MATLAB `ipermute` reference page for more information.

Purpose Determine whether `fi` object is column vector

Syntax `iscolumn(a)`

Description `iscolumn(a)` returns `1` if the `fi` object `a` is a column vector, and `0` otherwise.

See Also `isrow`

isempty

Purpose Determine whether array is empty

Description Refer to the MATLAB `isempty` reference page for more information.

Purpose

Determine whether real-world values of two `fi` objects are equal, or determine whether properties of two `fimath`, `numerictype`, or `quantizer` objects are equal

Syntax

```
isequal(a,b,...)
isequal(F,G,...)
isequal(T,U,...)
isequal(q,r,...)
```

Description

`isequal(a,b,...)` returns `1` if all the `fi` object inputs have the same real-world value. Otherwise, the function returns `0`.

`isequal(F,G,...)` returns `1` if all the `fimath` object inputs have the same properties. Otherwise, the function returns `0`.

`isequal(T,U,...)` returns `1` if all the `numerictype` object inputs have the same properties. Otherwise, the function returns `0`.

`isequal(q,r,...)` returns `1` if all the `quantizer` object inputs have the same properties. Otherwise, the function returns `0`.

See Also

`eq`, `ispropequal`

Purpose Determine whether variable is `fi` object

Syntax `isfi(a)`

Description `isfi(a)` returns `1` if `a` is a `fi` object, and `0` otherwise.

See Also `fi`, `isfimath`, `isnumerictype`

Purpose Determine whether variable is `fimath` object

Syntax `isfimath(F)`

Description `isfimath(F)` returns `1` if `F` is a `fimath` object, and `0` otherwise.

See Also `fimath`, `isfi`, `isnumerictype`

Purpose Determine whether array elements are finite

Description Refer to the MATLAB `isfinite` reference page for more information.

Purpose Determine whether array elements are infinite

Description Refer to the MATLAB `isinf` reference page for more information.

isnan

Purpose Determine whether array elements are NaN

Description Refer to the MATLAB `isnan` reference page for more information.

Purpose Determine whether input is numeric array

Description Refer to the MATLAB `isnumeric` reference page for more information.

isnumerictype

Purpose Determine whether variable is `numerictype` object

Syntax `isnumerictype(T)`

Description `isnumerictype(T)` returns `1` if `a` is a `numerictype` object, and `0` otherwise.

See Also `isfi`, `isfimath`, `numerictype`

Purpose Determine whether input is MATLAB OOPS object

Description Refer to the MATLAB `isobject` reference page for more information.

ispropequal

Purpose Determine whether properties of two `fi` objects are equal

Syntax `ispropequal(a,b,...)`

Description `ispropequal(a,b,...)` returns `1` if all the inputs are `fi` objects and all the inputs have the same properties. Otherwise, the function returns `0`.

To compare the real-world values of two `fi` objects `a` and `b`, use `a == b` or `isequal(a,b)`.

See Also `fi`, `isequal`

Purpose Determine whether array elements are real

Description Refer to the MATLAB `isreal` reference page for more information.

isrow

Purpose Determine whether `fi` object is row vector

Syntax `isrow(a)`

Description `isrow(a)` returns `1` if the `fi` object `a` is a row vector, and `0` otherwise.

See Also `iscolumn`

Purpose Determine whether input is scalar

Description Refer to the MATLAB `isscalar` reference page for more information.

issigned

Purpose Determine whether `fi` object is signed

Syntax `issigned(a)`

Description `issigned(a)` returns `1` if the `fi` object `a` is signed, and `0` if it is unsigned.

Purpose Determine whether input is vector

Description Refer to the MATLAB `isvector` reference page for more information.

Purpose

Determine whether real-world value of `fi` object is less than or equal to another

Syntax

```
c = le(a,b)
a <= b
```

Description

`c = le(a,b)` is called for the syntax `a <= b` when `a` or `b` is a `fi` object. `a` and `b` must have the same dimensions unless one is a scalar. A scalar can be compared with another object of any size.

`a <= b` does an element-by-element comparison between `a` and `b` and returns a matrix of the same size with elements set to `1` where the relation is true, and `0` where the relation is false.

See Also

`eq`, `ge`, `gt`, `lt`, `ne`

Purpose Vector length

Description Refer to the MATLAB `length` reference page for more information.

Purpose Create line object

Description Refer to the MATLAB `line` reference page for more information.

Purpose Convert numeric values to logical

Description Refer to the MATLAB `logical` reference page for more information.

loglog

Purpose Create log-log scale plot

Description Refer to the MATLAB `loglog` reference page for more information.

Purpose

Lower bound of range of `fi` object

Syntax

`lowerbound(a)`

Description

`lowerbound(a)` returns the lower bound of the range of `fi` object `a`. If `L=lowerbound(a)` and `U=upperbound(a)`, then `[L,U]=range(a)`.

See Also

`eps`, `intmax`, `intmin`, `lsb`, `range`, `realmax`, `realmin`, `upperbound`

Purpose

Scaling of least significant bit of `fi` object

Syntax

`lsb(a)`

Description

`lsb(a)` returns the scaling of the least significant bit of `fi` object a. The result is equivalent to the result given by the `eps` function.

See Also

`eps`, `intmax`, `intmin`, `lowerbound`, `range`, `realmax`, `realmin`, `upperbound`

Purpose

Determine whether real-world value of one `fi` object is less than another

Syntax

```
c = lt(a,b)
a < b
```

Description

`c = lt(a,b)` is called for the syntax `a < b` when `a` or `b` is a `fi` object. `a` and `b` must have the same dimensions unless one is a scalar. A scalar can be compared with another object of any size.

`a < b` does an element-by-element comparison between `a` and `b` and returns a matrix of the same size with elements set to `1` where the relation is true, and `0` where the relation is false.

See Also

`eq`, `ge`, `gt`, `le`, `ne`

Purpose

Largest element in array of `fi` objects

Syntax

```
max(a)
max(a,b)
[y,v] = max(a)
[y,v] = max(a,[],dim)
```

Description

- For vectors, `max(a)` is the largest element in `a`.
- For matrices, `max(a)` is a row vector containing the maximum element from each column.
- For N-D arrays, `max(a)` operates along the first nonsingleton dimension.

`max(a,b)` returns an array the same size as `a` and `b` with the largest elements taken from `a` or `b`. Either one can be a scalar.

`[y,v] = max(a)` returns the indices of the maximum values in vector `v`. If the values along the first nonsingleton dimension contain more than one maximal element, the index of the first one is returned.

`[y,v] = max(a,[],dim)` operates along the dimension `dim`.

When complex, the magnitude `max(abs(a))` is used, and the angle `angle(a)` is ignored. NaNs are ignored when computing the maximum.

See Also

`min`

Purpose

Largest real-world value of `fi` object or maximum value of `quantizer` object before quantization

Syntax

```
maxlog(a)
maxlog(q)
```

Description

`maxlog(a)` returns the largest real-world value of `fi` object `a` since logging was turned on or since the last time the log was reset for the object.

Turn on logging by setting the `fipref` property `LoggingMode` to `on`. Reset logging for a `fi` object using the `resetlog` function.

`maxlog(q)` is the maximum value before quantization during a call to `quantize(q,...)` for `quantizer` object `q`. This value is the maximum value encountered over successive calls to `quantize` and is reset with `resetlog(q)`. `maxlog(q)` is equivalent to `get(q,'maxlog')` and `q.maxlog`.

Examples

```
P = fipref('LoggingMode','on');
x = fi([-1.5 eps 0.5], true, 16, 15);
x(1) = 3.0;
maxlog(x)

ans =

     3
```

See Also

`fipref`, `minlog`, `noverflows`, `nunderflows`, `resetlog`

Purpose Create mesh plot

Description Refer to the MATLAB `mesh` reference page for more information.

Purpose Create mesh plot with contour plot

Description Refer to the MATLAB `meshc` reference page for more information.

Purpose Create mesh plot with curtain plot

Description Refer to the MATLAB `meshz` reference page for more information.

Purpose

Smallest element in array of `fi` objects

Syntax

```
min(a)
min(a,b)
[y,v] = min(a)
[y,v] = min(a,[],dim)
```

Description

- For vectors, `min(a)` is the smallest element in `a`.
- For matrices, `min(a)` is a row vector containing the minimum element from each column.
- For N-D arrays, `min(a)` operates along the first nonsingleton dimension.

`min(a,b)` returns an array the same size as `a` and `b` with the smallest elements taken from `a` or `b`. Either one can be a scalar.

`[y,v] = min(a)` returns the indices of the minimum values in vector `v`. If the values along the first nonsingleton dimension contain more than one minimal element, the index of the first one is returned.

`[y,v] = min(a,[],dim)` operates along the dimension `dim`.

When complex, the magnitude `min(abs(a))` is used, and the angle `angle(a)` is ignored. NaNs are ignored when computing the minimum.

See Also

`max`

minlog

Purpose

Smallest real-world value of `fi` object or minimum value of `quantizer` object before quantization

Syntax

```
minlog(a)
minlog(q)
```

Description

`minlog(a)` returns the smallest real-world value of `fi` object `a` since logging was turned on or since the last time the log was reset for the object.

Turn on logging by setting the `fipref` property `LoggingMode` to `on`. Reset logging for a `fi` object using the `resetlog` function.

`minlog(q)` is the minimum value before quantization during a call to `quantize(q,...)` for `quantizer` object `q`. This value is the minimum value encountered over successive calls to `quantize` and is reset with `resetlog(q)`. `minlog(q)` is equivalent to `get(q,'minlog')` and `q.minlog`.

Examples

```
P = fipref('LoggingMode','on');
x = fi([-1.5 eps 0.5], true, 16, 15);
x(1) = 3.0;
minlog(x)

ans =

     -1.5
```

See Also

`fipref`, `maxlog`, `noverflows`, `nunderflows`, `resetlog`

Purpose Matrix difference between `fi` objects

Syntax `minus(a,b)`

Description `minus(a,b)` is called for the syntax `a - b` when `a` or `b` is an object.

`a - b` subtracts matrix `b` from matrix `a`. `a` and `b` must have the same dimensions unless one is a scalar (a 1-by-1 matrix). A scalar can be subtracted from anything.

`minus` does not support `fi` objects of data type `Boolean`.

See Also `mtimes`, `plus`, `times`, `uminus`

Purpose

Multiply two objects using `fimath` object

Syntax

```
c = F.mpy(a,b)
```

Description

`c = F.mpy(a,b)` performs elementwise multiplication on `a` and `b` using `fimath` object `F`. This is helpful in cases when you want to override the `fimath` objects of `a` and `b`, or if the `fimath` objects of `a` and `b` are different.

`a` and `b` must have the same dimensions unless one is a scalar. If either `a` or `b` is scalar, then `c` has the dimensions of the nonscalar object.

If either `a` or `b` is a `fi` object, and the other is a MATLAB built-in numeric type, then the built-in object is cast to the word length of the `fi` object, preserving best-precision fraction length.

Examples

In this example, `c` is the 40-bit product of `a` and `b` with fraction length 30.

```
a = fi(pi);
b = fi(exp(1));
F = fimath('ProductMode','SpecifyPrecision',...
  'ProductWordLength',40,'ProductFractionLength',30);
c = F.mpy(a, b)

c =

    8.5397


           DataTypeMode: Fixed-point: binary point scaling
                 Signed: true
             WordLength: 40
         FractionLength: 30

              RoundMode: nearest
           OverflowMode: saturate
            ProductMode: SpecifyPrecision
      ProductWordLength: 40
  ProductFractionLength: 30
```

```
             SumMode: FullPrecision
   MaxSumWordLength: 128
       CastBeforeSum: true
```

Algorithm

`c = F.mpy(a,b)` is equivalent to

```
a.fimath = F;
b.fimath = F;
c = a .* b;
```

except that the `fimath` properties of `a` and `b` are not modified when you use the functional form.

See Also

`add`, `divide`, `fi`, `fimath`, `numerictype`, `sub`, `sum`

Purpose

Matrix product of fi objects

Syntax

```
mtimes(a,b)
```

Description

mtimes(a,b) is called for the syntax a * b when a or b is an object.

a * b is the matrix product of a and b. Any scalar (a 1-by-1 matrix) can multiply anything. Otherwise, the number of columns of a must equal the number of rows of b.

mtimes does not support fi objects of data type Boolean.

See Also

plus, minus, times, uminus

Purpose Number of array dimensions

Description Refer to the MATLAB `ndims` reference page for more information.

Purpose Determine whether real-world values of two `fi` objects are not equal

Syntax

```
c = ne(a,b)
a ~= b
```

Description `c = ne(a,b)` is called for the syntax `a ~= b` when `a` or `b` is a `fi` object. `a` and `b` must have the same dimensions unless one is a scalar. A scalar can be compared with another object of any size.

`a ~= b` does an element-by-element comparison between `a` and `b` and returns a matrix of the same size with elements set to `1` where the relation is true, and `0` where the relation is false.

See Also `eq`, `ge`, `gt`, `le`, `lt`

Purpose

Number of operations

Syntax

```
noperations(q)
```

Description

noperations(q) is the number of quantization operations during a call to quantize(q,...) for quantizer object q. This value accumulates over successive calls to quantize. You reset the value of noperations to zero by issuing the command resetlog(q).

Each time any data element is quantized, noperations is incremented by one. The real and complex parts are counted separately. For example, (complex * complex) counts four quantization operations for products and two for sum, because(a+bi)*(c+di) = (a*c - b*d) + (a*d + b*c). In contrast, (real*real) counts one quantization operation.

In addition, the real and complex parts of the inputs are quantized individually. As a result, for a complex input of length 204 elements, noperations counts 408 quantizations: 204 for the real part of the input and 204 for the complex part.

If any inputs, states, or coefficients are complex-valued, they are all expanded from real values to complex values, with a corresponding increase in the number of quantization operations recorded by noperations. In concrete terms, (real*real) requires fewer quantizations than (real*complex) and (complex*complex). Changing all the values to complex because one is complex, such as the coefficient, makes the (real*real) into (real*complex), raising noperations count.

See Also

maxlog, minlog

Purpose Find logical NOT of array or scalar input

Description Refer to the MATLAB `not` reference page for more information.

Purpose Number of overflows

Syntax

```
noverflows(a)
noverflows(q)
```

Description

`noverflows(a)` returns the number of overflows of `fi` object `a` since logging was turned on or since the last time the log was reset for the object.

Turn on logging by setting the `fipref` property `LoggingMode` to `on`. Reset logging for a `fi` object using the `resetlog` function.

`noverflows(q)` returns the accumulated number of overflows resulting from quantization operations performed by a `quantizer` object `q`.

See Also `maxlog`, `minlog`, `nunderflows`, `resetlog`

num2bin

Purpose Convert number to binary string using `quantizer` object

Syntax

```
y = num2bin(q,x)
```

Description `y = num2bin(q,x)` converts numeric array `x` into binary strings returned in `y`. When `x` is a cell array, each numeric element of `x` is converted to binary. If `x` is a structure, each numeric field of `x` is converted to binary.

`num2bin` and `bin2num` are inverses of one another, differing in that `num2bin` returns the binary strings in a column.

Examples

```
x = magic(3)/9;
q = quantizer([4,3]);
y = num2bin(q,x)
Warning: 1 overflow.
y =

0111
0010
0011
0000
0100
0111
0101
0110
0001
```

See Also `bin2num`, `hex2num`, `num2hex`, `num2int`

Purpose

Convert number to hexadecimal equivalent using `quantizer` object

Syntax

```
y = num2hex(q,x)
```

Description

`y = num2hex(q,x)` converts numeric array `x` into hexadecimal strings returned in `y`. When `x` is a cell array, each numeric element of `x` is converted to hexadecimal. If `x` is a structure, each numeric field of `x` is converted to hexadecimal.

For fixed-point `quantizer` objects, the representation is two's complement. For floating-point `quantizer` objects, the representation is IEEE Standard 754 style.

For example, for `q = quantizer('double')`

```
num2hex(q,nan)

ans =

fff8000000000000
```

The leading fraction bit is 1, all other fraction bits are 0. Sign bit is 1, exponent bits are all 1.

```
num2hex(q,inf)

ans =

7ff0000000000000
```

Sign bit is 0, exponent bits are all 1, all fraction bits are 0.

```
num2hex(q,-inf)

ans =

fff0000000000000
```

Sign bit is 1, exponent bits are all 1, all fraction bits are 0.

`num2hex` and `hex2num` are inverses of each other, except that `num2hex` returns the hexadecimal strings in a column.

Examples

This is a floating-point example using a `quantizer` object `q` that has 6-bit word length and 3-bit exponent length.

```
x = magic(3);
q = quantizer('float',[6 3]);
y = num2hex(q,x)

y =

18
12
14
0c
15
18
16
17
10
```

See Also

`bin2num`, `hex2num`, `num2bin`, `num2int`

Purpose

Convert number to signed integer

Syntax

```
y = num2int(q,x)
[y1,y,...] = num2int(q,x1,x,...)
```

Description

`y = num2int(q,x)` uses `q.format` to convert numeric `x` to an integer.

`[y1,y,...]  = num2int(q,x1,x,...)` uses `q.format` to convert numeric values `x1, x2,...` to integers `y1,y2,...`

Examples

All the two's complement 4-bit numbers in fractional form are given by

```
x = [0.875 0.375 -0.125 -0.625
     0.750 0.250 -0.250 -0.750
     0.625 0.125 -0.375 -0.875
     0.500 0.000 -0.500 -1.000];

q=quantizer([4 3]);

y = num2int(q,x)
y =

     7     3    -1    -5
     6     2    -2    -6
     5     1    -3    -7
     4     0    -4    -8
```

Algorithm

When `q` is a fixed-point `quantizer` object, f is equal to `fractionlength(q)`, and x is numeric

$$y = x \times 2^f$$

When `q` is a floating-point `quantizer` object, $y = x$. `num2int` is meaningful only for fixed-point `quantizer` objects.

See Also

`bin2num`, `hex2num`, `num2bin`, `num2hex`

numberofelements

Purpose

Number of data elements in `fi` array

Syntax

```
numberofelements(a)
```

Description

`numberofelements(a)` returns the number of data elements in a `fi` array. `numberofelements(a) == prod(size(a))`.

Note that `fi` is a MATLAB object, and therefore `numel(a)` returns `1` when `a` is a `fi` object. Refer to the information about classes in the MATLAB `numel` reference page.

See Also

`max`, `min`, `numel`

Purpose

Construct `numerictype` object

Syntax

```
T = numerictype
T = numerictype(s)
T = numerictype(s,w)
T = numerictype(s,w,f)
T = numerictype(s,w,slope,bias)
T = numerictype(s,w,slopeadjustmentfactor,fixedexponent,bias)
T = numerictype(property1,value1, ...)
T = numerictype(T1, property1, value1, ...)
T = numerictype('double')
T = numerictype('single')
T = numerictype('boolean')
```

Description

You can use the `numerictype` constructor function in the following ways:

- `T = numerictype` creates a default `numerictype` object.
- `T = numerictype(s)` creates a `numerictype` object with `Fixed-point: unspecified scaling`, signedness `s`, and 16-bit word length.
- `T = numerictype(s,w)` creates a `numerictype` object with `Fixed-point: unspecified scaling`, signedness `s`, and word length `w`.
- `T = numerictype(s,w,f)` creates a `numerictype` object with `Fixed-point: binary point scaling`, signedness `s`, word length `w` and fraction length `f`.
- `T = numerictype(s,w,slope,bias)` creates a `numerictype` object with `Fixed-point: slope and bias scaling`, signedness `s`, word length `w`, `slope`, and `bias`.
- `T = numerictype(s,w,slopeadjustmentfactor,fixedexponent,bias)` creates a `numerictype` object with `Fixed-point: slope and bias scaling`, signedness `s`, word length `w`, `slopeadjustmentfactor`, `fixedexponent`, and `bias`.

- T = numerictype(property1,value1, ...) allows you to set properties for a numerictype object using property name/property value pairs.
- T = numerictype(T1, property1, value1, ...) allows you to make a copy of an existing numerictype object, while modifying any or all of the property values.
- T = numerictype('double') creates a double numerictype.
- T = numerictype('single') creates a single numerictype.
- T = numerictype('boolean') creates a Boolean numerictype.

The properties of the numerictype object are listed below. These properties are described in detail in "numerictype Object Properties" on page 1-17.

- Bias — Bias
- DataType — Data type category
- DataTypeMode — Data type and scaling mode
- FixedExponent — Fixed-point exponent
- SlopeAdjustmentFactor — Slope adjustment
- FractionLength — Fraction length of the stored integer value, in bits
- Scaling — Fixed-point scaling mode
- Signed — Signed or unsigned
- Slope — Slope
- WordLength — Word length of the stored integer value, in bits

Examples

Example 1

Type

```
T = numerictype
```

to create a default numerictype object.

```
T =
                DataType: Fixed
                 Scaling: BinaryPoint
                  Signed: true
              WordLength: 16
          FractionLength: 15
```

Example 2

The following creates a signed numerictype object with a 32-bit word length and 30-bit fraction length.

```
T = numerictype(1, 32, 30)

T =

            DataTypeMode: Fixed-point: binary point scaling
                  Signed: true
              WordLength: 32
          FractionLength: 30
```

Example 3

If you omit the argument f, the scaling is unspecified.

```
T = numerictype(1, 32)

T =

            DataTypeMode: Fixed-point: unspecified scaling
                  Signed: true
              WordLength: 32
```

Example 4

If you omit the arguments w and f, the word length is automatically set to 16 bits and the scaling is unspecified.

```
T = numerictype(1)

T =

          DataTypeMode: Fixed-point: unspecified scaling
                Signed: true
            WordLength: 16
```

Example 5

You can use property name/property value pairs to set numerictype properties when you create the object.

```
T = numerictype('Signed', true, ...
      'DataTypeMode', 'Fixed-point: slope and bias', ...
      'WordLength', 32, 'Slope', 2^-2, 'Bias', 4)

T =

          DataTypeMode: Fixed-point: slope and bias scaling
                Signed: true
            WordLength: 32
                 Slope: 0.25
                  Bias: 4
```

See Also

fi, fimath, fipref, quantizer

Purpose

Number of underflows

Syntax

```
nunderflows(a)
nunderflows(q)
```

Description

`nunderflows(a)` returns the number of underflows of `fi` object a since logging was turned on or since the last time the log was reset for the object.

Turn on logging by setting the `fipref` property `LoggingMode` to `on`. Reset logging for a `fi` object using the `resetlog` function.

`nunderflows(q)` returns the accumulated number of underflows resulting from quantization operations performed by a `quantizer` object `q`.

See Also

`maxlog`, `minlog`, `noverflows`, `resetlog`

Purpose

Octal representation of stored integer of `fi` object

Syntax

```
oct(a)
```

Description

Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times stored\ integer$$

or, equivalently,

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

`oct(a)` returns the stored integer of `fi` object `a` in octal format as a string.

Examples

The following code

```
a = fi([-1 1],1,8,7);
oct(a)
```

returns

```
200   177
```

See Also

`bin`, `dec`, `hex`, `int`

Purpose Find logical OR of array or scalar inputs

Description Refer to the MATLAB `or` reference page for more information.

Purpose

Create patch graphics object

Description

Refer to the MATLAB `patch` reference page for more information.

Purpose Create pseudocolor plot

Description Refer to the MATLAB `pcolor` reference page for more information.

permute

Purpose Rearrange dimensions of multidimensional array

Description Refer to the MATLAB `permute` reference page for more information.

Purpose Create linear 2-D plot

Description Refer to the MATLAB `plot` reference page for more information.

plot3

Purpose Create 3-D line plot

Description Refer to the MATLAB `plot3` reference page for more information.

Purpose Draw scatter plots

Description Refer to the MATLAB `plotmatrix` reference page for more information.

plotyy

Purpose Create graph with y-axes on right and left sides

Description Refer to the MATLAB `plotyy` reference page for more information.

Purpose

Matrix sum of fi objects

Syntax

```
plus(a,b)
```

Description

plus(a,b) is called for the syntax a + b when a or b is an object.

a + b adds matrices a and b. a and b must have the same dimensions unless one is a scalar (a 1-by-1 matrix). A scalar can be added to anything.

plus does not support fi objects of data type Boolean.

See Also

minus, mtimes, times, uminus

Purpose Plot polar coordinates

Description Refer to the MATLAB `polar` reference page for more information.

Purpose

Multiply by 2^K

Syntax

```
b = pow2(a, K)
```

Description

`b = pow2(a, K)` returns

$$b = a \times 2^K$$

where `K` is an integer and `a` and `b` are fi objects. If `K` is a non-integer, it will be rounded to `floor` before the calculation is performed. The scaling of `a` must be equivalent to binary point-only scaling; in other words, it must have a fractional slope of 1 and a bias of 0.

The syntax `b = pow2(a)` is not supported when `a` is a `fi` object.

`a` can be real or complex. If `a` is complex, `pow2` operates on both the real and complex portions of `a`.

`pow2` does not support `fi` objects of data type `Boolean`.

Examples

The following example shows the use of `pow2` with a complex `fi` object:

```
format long g
P = fipref('NumericTypeDisplay', 'short', ...
           'FimathDisplay', 'none');
a = fi(57 - 2i, 1, 16, 8)

a =

                        57 -                         2i
      s16,8

pow2(a, 2)

ans =

              127.99609375 -                         8i
      s16,8
```

See Also `bitshift`

Purpose Apply quantizer object to data

Syntax

```
y = quantize(q, x)
[y1,y2,...] = quantize(q,x1,x2,...)
```

Description y = quantize(q, x) uses the quantizer object q to quantize x. When x is a numeric array, each element of x is quantized. When x is a cell array, each numeric element of the cell array is quantized. When x is a structure, each numeric field of x is quantized. Nonnumeric elements or fields of x are left unchanged and quantize does not issue warnings for nonnumeric values.

[y1,y2,...] = quantize(q,x1,x2,...) is equivalent to

```
y1 = quantize(q,x1), y2 = quantize(q,x2),...
```

The quantizer object states

- max — Maximum value before quantizing
- min — Minimum value before quantizing
- noverflows — Number of overflows
- nunderflows — Number of underflows
- noperations — Number of quantization operations

are updated during the call to quantize, and running totals are kept until a call to resetlog is made.

Examples The following examples demonstrate using quantize to quantize data.

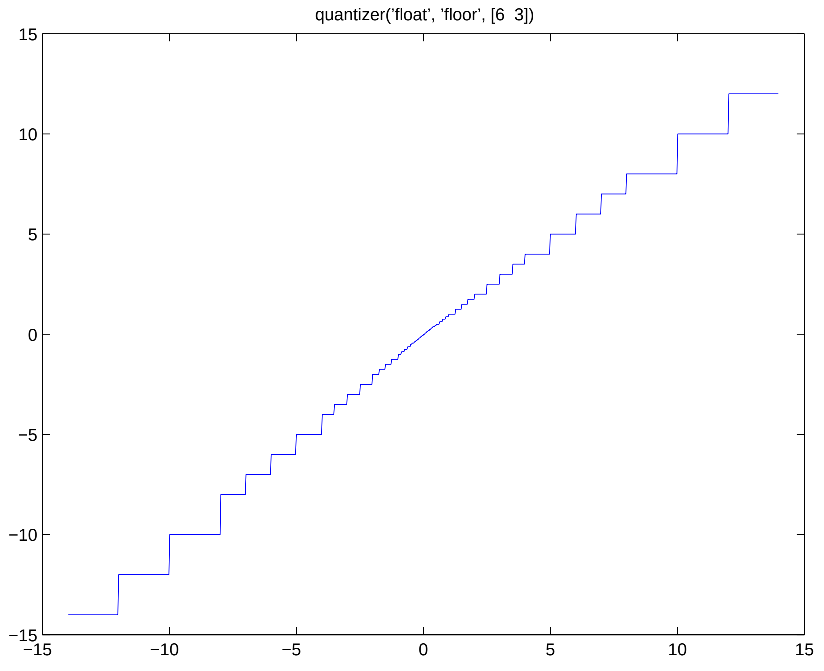
Example 1 - Custom Precision Floating-Point

The code listed here produces the plot shown in the following figure.

```
u=linspace(-15,15,1000);
q=quantizer([6 3],'float');
range(q)
```

```
ans =

   -14    14
y=quantize(q,u);
plot(u,y);title(tostring(q))
Warning: 68 overflows.
```



Example 2 - Fixed-Point

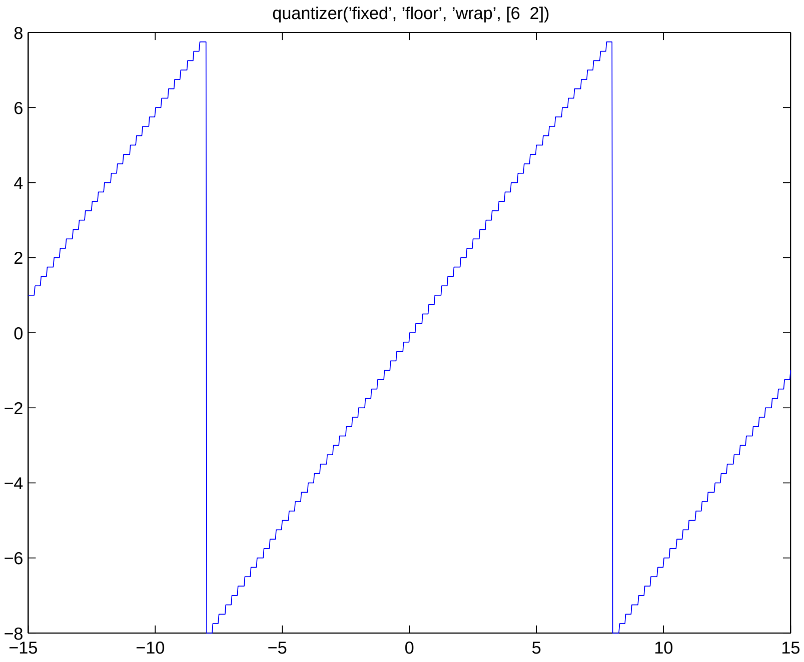
The code listed here produces the plot shown in the following figure.

```
u=linspace(-15,15,1000);
q=quantizer([6 2],'wrap');
```

```
range(q)

ans =

   -8.0000    7.7500
y=quantize(q,u);
plot(u,y);title(tostring(q))
Warning: 468 overflows.
```



See Also

`quantizer`, `set`

Purpose Construct quantizer object

Syntax

```
q = quantizer
q = quantizer('PropertyName1',PropertyValue1,...)
q = quantizer(PropertyValue1,PropertyValue2,...)
q = quantizer(struct)
q = quantizer(pn,pv)
```

Description

`q = quantizer` creates a `quantizer` object with properties set to their default values.

`q = quantizer('PropertyName1',PropertyValue1,...)` uses property name/ property value pairs.

`q = quantizer(PropertyValue1,PropertyValue2,...)` creates a `quantizer` object with the listed property values. When two values conflict, `quantizer` sets the last property value in the list. Property values are unique; you can set the property names by specifying just the property values in the command.

`q = quantizer(struct)`, where `struct` is a structure whose field names are property names, sets the properties named in each field name with the values contained in the structure.

`q = quantizer(pn,pv)` sets the named properties specified in the cell array of strings `pn` to the corresponding values in the cell array `pv`.

The `quantizer` object property values are listed below. These properties are described in detail in "quantizer Object Properties" on page 1-21.

Property Name	Property Value	Description
`mode`	`'double'`	Double-precision mode. Override all other parameters.
	`'float'`	Custom-precision floating-point mode.

Property Name	Property Value	Description
	'fixed'	Signed fixed-point mode.
	'single'	Single-precision mode. Override all other parameters.
	'ufixed'	Unsigned fixed-point mode.
roundmode	'ceil'	Round toward positive infinity.
	'convergent'	Convergent rounding.
	'fix'	Round toward zero.
	'floor'	Round toward negative infinity.
	'nearest'	Round toward nearest.
overflowmode (fixed-point only)	'saturate'	Saturate on overflow.
	'wrap'	Wrap on overflow.
format	[wordlength fractionlength]	Format for fixed or ufixed mode.
	[wordlength exponentlength]	Format for float mode.

The default property values for a quantizer object are

```
mode = 'fixed';
roundmode = 'floor';
overflowmode = 'saturate';
format = [16 15];
```

Along with the preceding properties, quantizer objects have read-only states: max, min, noverflows, nunderflows, and noperations. They can be accessed through quantizer/get or q.maxlog, q.minlog, q.noverflows, q.nunderflows, and q.noperations, but they cannot be set. They are updated during the quantizer/quantize method, and are reset by the resetlog function.

The following table lists the read-only quantizer object states:

Property Name	Description
max	Maximum value before quantizing
min	Minimum value before quantizing
noverflows	Number of overflows
nunderflows	Number of underflows
noperations	Number of data points quantized

Examples

The following example operations are equivalent.

Setting quantizer object properties by listing property values only in the command,

```
q = quantizer('fixed', 'ceil', 'saturate', [5 4])
```

Using a structure struct to set quantizer object properties,

```
struct.mode = 'fixed';
struct.roundmode = 'ceil';
struct.overflowmode = 'saturate';
struct.format = [5 4];
q = quantizer(struct);
```

Using property name and property value cell arrays pn and pv to set quantizer object properties,

```
pn = {'mode',  'roundmode', 'overflowmode', 'format'};
pv = {'fixed', 'ceil', 'saturate', [5 4]};
q = quantizer(pn, pv)
```

Using property name/property value pairs to configure a quantizer object,

```
q = quantizer( 'mode', fixed','roundmode','ceil',...
'overflowmode', 'saturate', 'format', [5 4]);
```

See Also

fi, fimath, fipref, numerictype, quantize, set

Purpose Create quiver or velocity plot

Description Refer to the MATLAB `quiver` reference page for more information.

Purpose Create 3-D quiver or velocity plot

Description Refer to the MATLAB `quiver3` reference page for more information.

Purpose

Generate uniformly distributed, quantized random number using `quantizer` object

Syntax

```
randquant(q,n)
randquant(q,m,n)
randquant(q,m,n,p,...)
randquant(q,[m,n])
randquant(q,[m,n,p,...])
```

Description

`randquant(q,n)` uses `quantizer` object `q` to generate an `n`-by-`n` matrix with random entries whose values cover the range of `q` when `q` is a fixed-point `quantizer` object. When `q` is a floating-point `quantizer` object, `randquant` populates the `n`-by-`n` array with values covering the range

```
-[square root of realmax(q)] to [square root of realmax(q)]
```

`randquant(q,m,n)` uses `quantizer` object `q` to generate an `m`-by-`n` matrix with random entries whose values cover the range of `q` when `q` is a fixed-point `quantizer` object. When `q` is a floating-point `quantizer` object, `randquant` populates the `m`-by-`n` array with values covering the range

```
-[square root of realmax(q)] to [square root of realmax(q)]
```

`randquant(q,m,n,p,...)` uses `quantizer` object `q` to generate an `m`-by-`n`-by-`p`-by ... matrix with random entries whose values cover the range of `q` when `q` is fixed-point `quantizer` object. When `q` is a floating-point `quantizer` object, `randquant` populates the matrix with values covering the range

```
-[square root of realmax(q)] to [square root of realmax(q)]
```

`randquant(q,[m,n])` uses `quantizer` object `q` to generate an `m`-by-`n` matrix with random entries whose values cover the range of `q` when `q` is a fixed-point `quantizer` object. When `q` is a floating-point `quantizer` object, `randquant` populates the `m`-by-`n` array with values covering the range

```
-[square root of realmax(q)] to [square root of realmax(q)]
```

`randquant(q,[m,n,p,...])` uses `quantizer` object `q` to generate `p` `m`-by-`n` matrices containing random entries whose values cover the range of `q` when `q` is a fixed-point `quantizer` object. When `q` is a floating-point `quantizer` object, `randquant` populates the `m`-by-`n` arrays with values covering the range

```
-[square root of realmax(q)] to [square root of realmax(q)]
```

`randquant` produces pseudorandom numbers. The number sequence `randquant` generates during each call is determined by the state of the generator. Because MATLAB resets the random number generator state at startup, the sequence of random numbers generated by the function remains the same unless you change the state.

`randquant` works like `rand` in most respects, including the generator used, but it does not support the `'state'` and `'seed'` options available in `rand`.

Examples

```
q=quantizer([4 3]);
rand('state',0)
randquant(q,3)

ans =

    0.7500   -0.1250   -0.2500
   -0.6250    0.6250   -1.0000
    0.1250    0.3750    0.5000
```

See Also

`quantizer`, `rand`, `range`, `realmax`

range

Purpose

Numerical range of `fi` or `quantizer` object

Syntax

```
range(a)
[min, max]= range(a)
r = range(q)
[min, max] = range(q)
```

Description

`range(a)` returns a fi object with the minimum and maximum possible values of `fi` object `a`. All possible quantized real-world values of `a` are in the range returned. If `a` is a complex number, then all possible values of `real(a)` and `imag(a)` are in the range returned.

`[min, max]= range(a)` returns the minimum and maximum values of `fi` object `a` in separate output variables.

`r = range(q)` returns the two-element row vector $r = [a\ b]$ such that for all real x, `y = quantize(q,x)` returns y in the range $a \le y \le b$.

`[min, max] = range(q)` returns the minimum and maximum values of the range in separate output variables.

Examples

```
q = quantizer('float',[6 3]);
r = range(q)

r =

   -14    14
q = quantizer('fixed',[4 2],'floor');
[min,max] = range(q)

min =

    -2


max =

    1.7500
```

Algorithm

If q is a floating-point quantizer object, a = -realmax(q), b = realmax(q).

If q is a signed fixed-point quantizer object (datamode = 'fixed'),

$$a = -\mathrm{realmax}(q) - \mathrm{eps}(q) = \frac{-2^{w-1}}{2^f}$$

$$b = \mathrm{realmax}(q) = \frac{2^{w-1} - 1}{2^f}$$

If q is an unsigned fixed-point quantizer object (datamode = 'ufixed'),

$$a = 0$$

$$b = \mathrm{realmax}(q) = \frac{2^{w} - 1}{2^f}$$

See realmax for more information.

See Also

eps, exponentmax, exponentmin, fractionlength, intmax, intmin, lowerbound, lsb, max, min, realmax, realmin, upperbound

real

Purpose Real part of complex number

Description Refer to the MATLAB `real` reference page for more information.

Purpose

Largest positive fixed-point value or quantized number

Syntax

```
realmax(a)
realmax(q)
```

Description

`realmax(a)` is the largest real-world value that can be represented in the data type of `fi` object `a`. Anything larger overflows.

`realmax(q)` is the largest quantized number that can be represented where `q` is a `quantizer` object. Anything larger overflows.

Examples

```
q = quantizer('float',[6 3]);
x = realmax(q)

x =

     14
```

Algorithm

If `q` is a floating-point `quantizer` object, the largest positive number, x, is

$$x = 2^{E_{max}} \cdot (2 - eps(q))$$

If `q` is a signed fixed-point `quantizer` object, the largest positive number, x, is

$$x = \frac{2^{w-1} - 1}{2^f}$$

If `q` is an unsigned fixed-point `quantizer` object (`datamode = 'ufixed'`), the largest positive number, x, is

$$x = \frac{2^w - 1}{2^f}$$

See Also

`eps`, `exponentmax`, `exponentmin`, `fractionlength`, `intmax`, `intmin`, `lowerbound`, `lsb`, `quantizer`, `range`, `realmin`, `upperbound`

Purpose

Smallest positive normalized fixed-point value or quantized number

Syntax

```
realmin(a)
realmin(q)
```

Description

`realmin(a)` is the smallest real-world value that can be represented in the data type of `fi` object `a`. Anything smaller underflows.

`realmin(q)` is the smallest positive normal quantized number where `q` is a `quantizer` object. Anything smaller than `x` underflows or is an IEEE "denormal" number.

Examples

```
q = quantizer('float',[6 3]);
realmin(q)

ans =

    0.2500
```

Algorithm

If `q` is a floating-point `quantizer` object, $x = 2^{E_{min}}$ where $E_{min} = \text{exponentmin}(q)$ is the minimum exponent.

If `q` is a signed or unsigned fixed-point `quantizer` object, $x = 2^{-f} = \varepsilon$ where f is the fraction length.

See Also

`eps`, `exponentmax`, `exponentmin`, `fractionlength`, `intmax`, `intmin`, `lowerbound`, `lsb`, `range`, `realmax`, `upperbound`

repmat

Purpose Replicate and tile array

Description Refer to the MATLAB `repmat` reference page for more information.

Purpose

Change scaling of fi object

Syntax

```
b = rescale(a, fractionlength)
b = rescale(a, slope, bias)
b = rescale(a, slopeadjustmentfactor, fixedexponent, bias)
b = rescale(a, ..., PropertyName, PropertyValue, ...)
```

Description

The rescale function acts similarly to the fi copy function with the following exceptions:

- The fi copy constructor preserves the real-world value, while rescale preserves the stored integer value.
- rescale does not allow the Signed and WordLength properties to be changed.

Examples

In the following example, fi object a is rescaled to create fi object b. The real-world values of a and b are different, while their stored integer values are the same:

```
p = fipref('FimathDisplay','none',...
  'NumericTypeDisplay','short');
a = fi(10, 1, 8, 3)

a =

    10
      s8,3

b = rescale(a, 1)

b =

    40
      s8,1
```

```
stored_integer_a = a.int;
stored_integer_b = b.int;
isequal(stored_integer_a, stored_integer_b)

ans =

1
```

See Also

fi

Purpose Reset objects to initial conditions

Syntax `reset(obj)`

Description `reset(obj)` resets `fi`, `fimath`, `fipref`, or `quantizer` object `obj` to its initial conditions.

See Also `resetlog`

resetlog

Purpose

Clear log for `fi` or `quantizer` object

Syntax

```
resetlog(a)
resetlog(q)
```

Description

`resetlog(a)` clears the log for `fi` object a.

`resetlog(q)` clears the log for `quantizer` object q.

Turn logging on or off by setting the `fipref` property `LoggingMode`.

See Also

`fipref`, `maxlog`, `minlog`, `noperations`, `noverflows`, `nunderflows`, `reset`

Purpose Reshape array

Description Refer to the MATLAB `reshape` reference page for more information.

rgbplot

Purpose Plot colormap

Description Refer to the MATLAB `rgbplot` reference page for more information.

Purpose Create ribbon plot

Description Refer to the MATLAB `ribbon` reference page for more information.

Purpose Create angle histogram

Description Refer to the MATLAB `rose` reference page for more information.

Purpose

Round input data using `quantizer` object without checking for overflow

Syntax

`round(q,x)`

Description

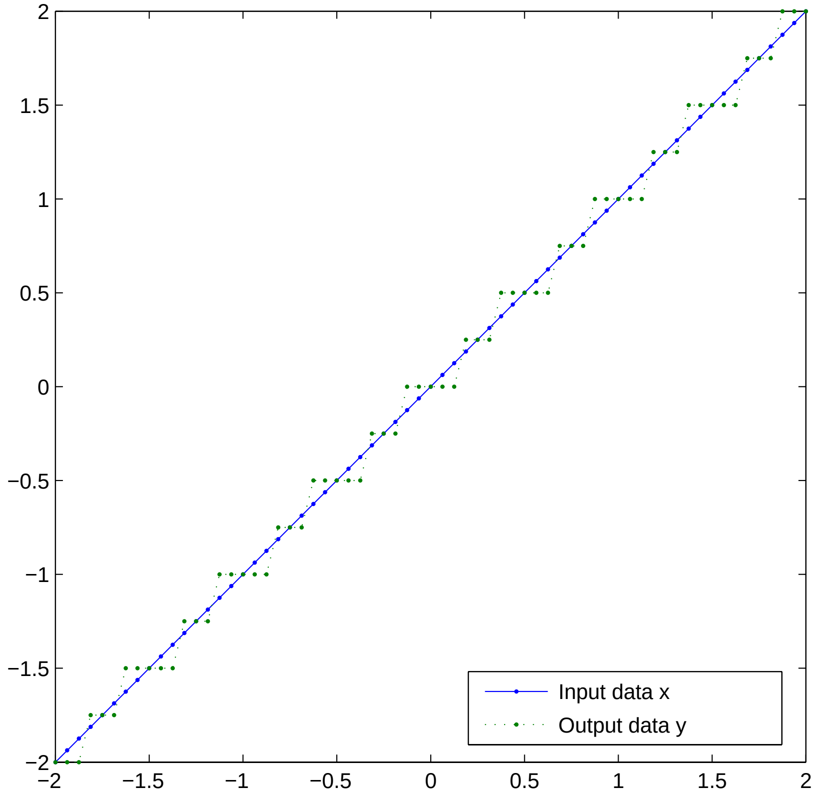
`round(q,x)` uses the `RoundMode` and `FractionLength` settings of `q` to round the numeric data `x`, but does not check for overflows during the operation. Compare to `quantize`.

Examples

Create a `quantizer` object and use it to quantize input data. The `quantizer` object applies its properties to the input data to return quantized output.

```
q = quantizer('fixed', 'convergent', 'wrap', [3 2]);
x = (-2:eps(q)/4:2)';
y = round(q,x);
plot(x,[x,y],'.-'); axis square;
```

Applying `quantizer` object `q` to the data results in the staircase shape output plot shown here. Where the input data is linear, output `y` shows distinct quantization levels.



See Also

`quantize`, `quantizer`

Purpose Save `fi` preferences for next MATLAB session

Syntax `savefipref`

Description `savefipref` saves the settings of the current `fipref` object for the next MATLAB session.

See Also `fipref`

Purpose Create scatter or bubble plot

Description Refer to the MATLAB `scatter` reference page for more information.

Purpose Create 3-D scatter or bubble plot

Description Refer to the MATLAB `scatter3` reference page for more information.

sdec

Purpose

Signed decimal representation of stored integer of `fi` object

Syntax

`sdec(a)`

Description

Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times stored\ integer$$

or, equivalently,

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

`sdec(a)` returns the stored integer of `fi` object `a` in signed decimal format as a string.

Examples

The code

```
a = fi([-1 1],1,8,7);
sdec(a)
```

returns

```
-128   127
```

See Also

`bin`, `dec`, `hex`, `int`, `oct`

Purpose Create semilogarithmic plot with logarithmic x-axis

Description Refer to the MATLAB `semilogx` reference page for more information.

semilogy

Purpose Create semilogarithmic plot with logarithmic y-axis

Description Refer to the MATLAB `semilogy` reference page for more information.

Purpose

Set or display property values for `quantizer` objects

Syntax

```
set(q, PropertyValue1, PropertyValue2,...)
set(q,s)
set(q,pn,pv)
set(q,'PropertyName1',PropertyValue1,'PropertyName2',
PropertyValue2,...)
q.PropertyName = Value
s = set(q)
```

Description

`set(q, PropertyValue1, PropertyValue2,...)` sets the properties of `quantizer` object `q`. If two property values conflict, the last value in the list is the one that is set.

`set(q,s)`, where `s` is a structure whose field names are object property names, sets the properties named in each field name with the values contained in the structure.

`set(q,pn,pv)` sets the named properties specified in the cell array of strings `pn` to the corresponding values in the cell array `pv`.

`set(q,'PropertyName1',PropertyValue1,'PropertyName2', PropertyValue2,...)` sets multiple property values with a single statement. Note that you can use property name/property value string pairs, structures, and property name/property value cell array pairs in the same call to `set`.

`q.PropertyName = Value` uses dot notation to set property `PropertyName` to `Value`.

`set(q)` displays the possible values for all properties of `quantizer` object `q`.

`s = set(q)` returns a structure containing the possible values for the properties of `quantizer` object `q`.

See Also

`get`

Purpose Shift dimensions

Description Refer to the MATLAB `shiftdim` reference page for more information.

Purpose Perform signum function on array

Syntax

```
c = sign(a)
```

Description `c = sign(a)` returns an array `c` the same size as `a`, where each element of `c` is

- `1` if the corresponding element of `a` is greater than zero
- `0` if the corresponding element of `a` is zero
- `-1` if the corresponding element of `a` is less than zero

The elements of `c` are of data type `int8`.

`sign` does not support complex `fi` inputs.

single

Purpose Single-precision floating-point real-world value of `fi` object

Syntax `single(a)`

Description Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times stored\ integer$$

or, equivalently,

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

`single(a)` returns the real-world value of a `fi` object in single-precision floating point.

See Also `double`

Purpose Array dimensions

Description Refer to the MATLAB `size` reference page for more information.

Purpose

Create volumetric slice plot

Description

Refer to the MATLAB `slice` reference page for more information.

Purpose Visualize sparsity pattern

Description Refer to the MATLAB `spy` reference page for more information.

Purpose

Square root of `fi` object

Syntax

```
c = sqrt(a)
c = sqrt(a,T)
c = sqrt(a,F)
c = sqrt(a,T,F)
```

Description

This function computes the square root of a `fi` object using a bisection algorithm.

`c = sqrt(a)` returns the square root of `fi` object `a` with the same `fimath` object as `a`. Intermediate quantities are also calculated using the `fimath` object of `a`. The `numerictype` object of `c` is determined automatically for you using an internal rule.

`c = sqrt(a,T)` returns the square root of `fi` object `a` with `numerictype` object `T` and the same `fimath` object as `a`. Intermediate quantities are calculated using the `fimath` object of `a`. See "Data Type Propagation Rules" on page 3-211.

`c = sqrt(a,F)` returns the square root of `fi` object `a` with `fimath` object `F`. Intermediate quantities are also calculated using `fimath` object `F`. The `numerictype` object of `c` is determined automatically for you using an internal rule. When `a` is a built-in `double` or `single` data type, this syntax is equivalent to `c = sqrt(a)` and the `fimath` object `F` is ignored.

`c = sqrt(a,T,F)` returns the square root `fi` object `a` with `numerictype` object `T` and `fimath` object `F`. Intermediate quantities are also calculated using `fimath` object `F`. See "Data Type Propagation Rules" on page 3-211.

`sqrt` does not support complex, negative-valued, or [Slope Bias] inputs.

Internal Rule

For syntaxes where the `numerictype` object of the output is not specified as an input to the `sqrt` function, it is automatically calculated according to the following internal rule:

$$sign_c = sign_a$$

$$WL_c = \text{ceil}(\frac{WL_a}{2})$$

$$FL_c = WL_c - \text{ceil}(\frac{WL_a - FL_a}{2})$$

Data Type Propagation Rules

For syntaxes for which you specify a `numerictype` object `T`, the `sqrt` function follows the data type propagation rules listed in the following table. In general, these rules can be summarized as "floating-point data types are propagated." This allows you to write code that can be used with both fixed-point and floating-point inputs.

Data Type of Input fi Object a	Data Type of numerictype object T	Data Type of Output c
Built-in `double`	Any	Built-in `double`
Built-in `single`	Any	Built-in `single`
`fi Fixed`	`fi Fixed`	Data type of `numerictype` object `T`
`fi ScaledDouble`	`fi Fixed`	`ScaledDouble` with properties of `numerictype` object `T`
`fi double`	`fi Fixed`	`fi double`
`fi single`	`fi Fixed`	`fi single`
Any `fi` data type	`fi double`	`fi double`
Any `fi` data type	`fi single`	`fi single`

squeeze

Purpose Remove singleton dimensions

Description Refer to the MATLAB `squeeze` reference page for more information.

Purpose Create stairstep graph

Description Refer to the MATLAB `stairs` reference page for more information.

Purpose Plot discrete sequence data

Description Refer to the MATLAB `stem` reference page for more information.

Purpose Plot 3-D discrete sequence data

Description Refer to the MATLAB `stem3` reference page for more information.

Purpose Create 3-D stream ribbon plot

Description Refer to the MATLAB `streamribbon` reference page for more information.

Purpose Draw streamlines in slice planes

Description Refer to the MATLAB `streamslice` reference page for more information.

Purpose Create 3-D stream tube plot

Description Refer to the MATLAB `streamtube` reference page for more information.

Purpose

Stored integer of `fi` object

Syntax

```
I = stripscaling(a)
```

Description

`I = stripscaling(a)` returns the stored integer of `a` as a `fi` object with zero bias and the same word length and sign as `a`.

Examples

Stripscaling is useful for converting the value of a `fi` object to its stored integer value without changing any other parameters.

```
fipref('NumericTypeDisplay','short', ...
       'FimathDisplay','none');
format long g
a = fi(0.1,true,48,47)

a =

         0.100000000000001
      s48,47
b = stripscaling(a)

b =

            14073748835533
      s48,0
bin(a)

ans =

000011001100110011001100110011001100110011001101

bin(b)

ans =

000011001100110011001100110011001100110011001101
```

Notice that the stored integer values of a and b are identical, while their real-world values are different.

Purpose

Subtract two objects using `fimath` object

Syntax

```
c = F.sub(a,b)
```

Description

`c = F.sub(a,b)` subtracts objects `a` and `b` using `fimath` object `F`. This is helpful in cases when you want to override the `fimath` objects of `a` and `b`, or if the `fimath` objects of `a` and `b` are different.

`a` and `b` must have the same dimensions unless one is a scalar. If either `a` or `b` is scalar, then `c` has the dimensions of the nonscalar object.

If either `a` or `b` is a `fi` object, and the other is a MATLAB built-in numeric type, then the built-in object is cast to the word length of the `fi` object, preserving best-precision fraction length.

Examples

In this example, `c` is the 32-bit difference of `a` and `b` with fraction length 16.

```
a = fi(pi);
b = fi(exp(1));
F = fimath('SumMode','SpecifyPrecision',...
  'SumWordLength',32,'SumFractionLength',16);
c = F.sub(a, b)

c =

    0.4233


          DataTypeMode: Fixed-point: binary point scaling
                Signed: true
            WordLength: 32
        FractionLength: 16

             RoundMode: nearest
          OverflowMode: saturate
           ProductMode: FullPrecision
  MaxProductWordLength: 128
```

```
              SumMode: SpecifyPrecision
        SumWordLength: 32
    SumFractionLength: 16
        CastBeforeSum: true
```

Algorithm

`c = F.sub(a,b)` is equivalent to

```
a.fimath = F;
b.fimath = F;
c = a - b;
```

except that the `fimath` properties of `a` and `b` are not modified when you use the functional form.

See Also

`add`, `divide`, `fi`, `fimath`, `mpy`, `numerictype`

Purpose

Subscripted assignment

Syntax

```
a(I) = b
a(I,J) = b
a(I,:) = b
a(:,I) = b
a(I,J,K,...) = b
a = subsasgn(a,S,b)
```

Description

`a(I) = b` assigns the values of `b` into the elements of `a` specified by the subscript vector `I`. `b` must have the same number of elements as `I` or be a scalar.

`a(I,J) = b` assigns the values of `b` into the elements of the rectangular submatrix of `a` specified by the subscript vectors `I` and `J`. `b` must have `LENGTH(I)` rows and `LENGTH(J)` columns.

A colon used as a subscript, as in `a(I,:) = b` or `a(:,I) = b` indicates the entire column or row.

For multidimensional arrays, `a(I,J,K,...) = b` assigns `b` to the specified elements of `a`. `b` must be `length(I)`-by-`length(J)`-by-`length(K)`-... or be shiftable to that size by adding or removing singleton dimensions.

`a = subsasgn(a,S,b)` is called for the syntax `a(i)=b`, `a{i}=b`, or `a.i=b` when `a` is an object. `S` is a structure array with the fields

- type — String containing `'()'`, `'{}'`, or `'.'` specifying the subscript type
- subs — Cell array or string containing the actual subscripts

For instance, the syntax `a(1:2,:) = b` calls `a=subsasgn(a,S,b)` where `S` is a 1-by-1 structure with `S.type='()'` and `S.subs = {1:2,':'}`. A colon used as a subscript is passed as the string `':'`.

Examples

For `fi` objects `a` and `b`, there is a difference between

```
a = b
```

and

```
a(:) = b
```

In the first case, `a = b` replaces `a` with `b`, and `a` assumes the value, `numerictype` object, and `fimath` object of `b`.

In the second case, `a(:)  = b` assigns the value of `b` into `a` while keeping the `numerictype` object of `a`. You can use this to cast a value with one `numerictype` object into another `numerictype` object.

For example, cast a 16-bit number into an 8-bit number:

```
a = fi(0, 1, 8, 7)

a =

     0

          DataTypeMode: Fixed-point: binary point scaling
                Signed: true
            WordLength: 8
        FractionLength: 7

b = fi(pi/4, 1, 16, 15)

b =

    0.7854

          DataTypeMode: Fixed-point: binary point scaling
                Signed: true
            WordLength: 16
        FractionLength: 15
```

```
a(:) = b

a =

    0.7891

          DataTypeMode: Fixed-point: binary point scaling
                Signed: true
            WordLength: 8
        FractionLength: 7
```

In this kind of assignment operation, the `fimath` objects of `a` and `b` can be different. A common use for this is when casting the result of an accumulation to an output data type, where the two have different rounding and overflow modes. Another common use is in a series of multiply/accumulate operations. For example,

```
for k = 1:n
 acc(1) = acc + b * x(k)
end
```

See Also

`subsref`

subsref

Purpose Subscripted reference

Description Refer to the MATLAB `subsref` reference page for more information.

Purpose

Sum of array elements

Syntax

```
b = sum(a)
b = sum(a, dim)
```

Description

`b = sum(a)` returns the sum along different dimensions of the `fi` array `a`.

If `a` is a vector, `sum(a)` returns the sum of the elements.

If `a` is a matrix, `sum(a)` treats the columns of `a` as vectors, returning a row vector of the sums of each column.

If `a` is a multidimensional array, `sum(a)` treats the values along the first nonsingleton dimension as vectors, returning an array of row vectors.

`b = sum(a, dim)` sums along the dimension `dim` of `a`.

The `fimath` object is used in the calculation of the sum. If `SumMode` is `FullPrecision`, `KeepLSB`, or `KeepMSB`, then the number of integer bits of growth for `sum(a)` is `ceil(log2(length(a)))`.

`sum` does not support `fi` objects of data type `Boolean`.

See Also

`add`, `divide`, `fi`, `fimath`, `mpy`, `numerictype`, `sub`

Purpose Create 3-D shaded surface plot

Description Refer to the MATLAB `surf` reference page for more information.

Purpose Create 3-D shaded surface plot with contour plot

Description Refer to the MATLAB `surfc` reference page for more information.

Purpose

Create surface plot with colormap-based lighting

Description

Refer to the MATLAB `surfl` reference page for more information.

Purpose Compute and display 3-D surface normals

Description Refer to the MATLAB `surfnorm` reference page for more information.

Purpose Create text object in current axes

Description Refer to the MATLAB `text` reference page for more information.

Purpose Element-by-element multiplication of `fi` objects

Syntax

```
times(a,b)
```

Description `times(a,b)` is called for the syntax `a .* b` when `a` or `b` is an object.

`a.*b` denotes element-by-element multiplication. `a` and `b` must have the same dimensions unless one is a scalar. A scalar can be multiplied into anything.

`times` does not support `fi` objects of data type `Boolean`.

See Also `plus`, `minus`, `mtimes`, `uminus`

toeplitz

Purpose Create Toeplitz matrix

Syntax

```
t = toeplitz(a,b)
t = toeplitz(b)
```

Description

`t = toeplitz(a,b)` returns a nonsymmetric Toeplitz matrix having `a` as its first column and `b` as its first row. `b` is cast to the `numerictype` of `a`.

`t = toeplitz(b)` returns the symmetric or Hermitian Toeplitz matrix formed from vector `b`, where `b` is the first row of the matrix.

The `numerictype` and `fimath` objects of the leftmost input that is a `fi` object are applied to the output `t`.

Examples

`toeplitz(a,b)` casts `b` into the data type of `a`. In this example, overflow occurs:

```
fipref('NumericTypeDisplay','short', ...
       'FimathDisplay','none');
format short g
a = fi([1 2 3],true,8,5)

a =

     1     2     3
      s8,5
b = fi([1 4 8],true,16,10)

b =

     1     4     8
      s16,10
```

```
toeplitz(a,b)

ans =

           1      3.9688      3.9688
           2           1      3.9688
           3           2           1
     s8,5
```

toeplitz(b,a) casts a into the data type of b. In this example, overflow does not occur:

```
toeplitz(b,a)

ans =

     1     2     3
     4     1     2
     8     4     1
      s16,10
```

If one of the arguments of toeplitz is a built-in data type, it is cast to the data type of the fi object.

```
x = [1 exp(1) pi]

x =

           1      2.7183      3.1416

toeplitz(a,x)

ans =

           1      2.7188      3.1563
           2           1      2.7188
           3           2           1
     s8,5
```

```
toeplitz(x,a)

ans =

            1             2             3
       2.7188             1             2
       3.1563        2.7188             1
      s8,5
```

Purpose Convert `quantizer` object to string

Syntax

```
s = tostring(q)
```

Description `s = tostring(q)` converts `quantizer` object `q` to a string `s`. After converting `q` to a string, the function `eval(s)` can use `s` to create a `quantizer` object with the same properties as `q`.

See Also `quantizer`

transpose

Purpose Transpose operation

Description Refer to the MATLAB `arithmetic operators` reference page for more information.

Purpose Plot picture of tree

Description Refer to the MATLAB `treeplot` reference page for more information.

Purpose Lower triangular part of matrix

Description Refer to the MATLAB `tril` reference page for more information.

Purpose Create triangular mesh plot

Description Refer to the MATLAB `trimesh` reference page for more information.

triplot

Purpose Create 2-D triangular plot

Description Refer to the MATLAB `triplot` reference page for more information.

Purpose Create triangular surface plot

Description Refer to the MATLAB `trisurf` reference page for more information.

Purpose Upper triangular part of matrix

Description Refer to the MATLAB `triu` reference page for more information.

Purpose

Stored integer value of `fi` object as built-in `uint8`

Syntax

`uint8(a)`

Description

Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times stored\ integer$$

or, equivalently,

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

`uint8(a)` returns the stored integer value of `fi` object `a` as a built-in `uint8`. If the stored integer word length is too big for a `uint8`, or if the stored integer is signed, the returned value saturates to a `uint8`.

See Also

`int`, `int8`, `int16`, `int32`, `uint16`, `uint32`

uint16

Purpose

Stored integer value of `fi` object as built-in `uint16`

Syntax

```
uint16(a)
```

Description

Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times stored\ integer$$

or, equivalently,

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

`uint16(a)` returns the stored integer value of `fi` object `a` as a built-in `uint16`. If the stored integer word length is too big for a `uint16`, or if the stored integer is signed, the returned value saturates to a `uint16`.

See Also

`int`, `int8`, `int16`, `int32`, `uint8`, `uint32`

Purpose

Stored integer value of `fi` object as built-in `uint32`

Syntax

`uint32(a)`

Description

Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times stored\ integer$$

or, equivalently,

$$real\text{-}world\ value = (slope \times stored\ integer) + bias$$

The stored integer is the raw binary number, in which the binary point is assumed to be at the far right of the word.

`uint32(a)` returns the stored integer value of `fi` object `a` as a built-in `uint32`. If the stored integer word length is too big for a `uint32`, or if the stored integer is signed, the returned value saturates to a `uint32`.

See Also

`int`, `int8`, `int16`, `int32`, `uint8`, `uint16`

Purpose

Negate elements of `fi` object array

Syntax

```
uminus(a)
```

Description

`uminus(a)` is called for the syntax `-a` when `a` is an object. `-a` negates the elements of `a`.

`uminus` does not support `fi` objects of data type `Boolean`.

Examples

When wrap occurs, -(-1) = -1 :

```
fipref('NumericTypeDisplay','short', ...
       'fimathDisplay','none');
format short g
a = fi(-1,true,8,7,'overflowmode','wrap')

a =

    -1
      s8,7
-a

ans =

    -1
      s8,7
b = fi([-1-i -1-i],true,8,7,'overflowmode','wrap')

b =

           -1 -          1i            -1 -          1i
      s8,7
-b

ans =

           -1 -          1i            -1 -          1i
```

```
      s8,7
b'

ans =

             -1 -           1i
             -1 -           1i
      s8,7
```

When saturation occurs, -(-1) = 0.99... :

```
c = fi(-1,true,8,7,'overflowmode','saturate')

c =

    -1
      s8,7
-c

ans =

      0.99219
      s8,7
d = fi([-1-i -1-i],true,8,7,'overflowmode','saturate')

d =

             -1 -           1i              -1 -           1i
      s8,7
-d

ans =

      0.99219 +    0.99219i       0.99219 +    0.99219i
      s8,7
d'
```

```
ans =

          -1 +    0.99219i
          -1 +    0.99219i
     s8,7
```

See Also

`plus`, `minus`, `mtimes`, `times`

Purpose Unary plus

Description Refer to the MATLAB `arithmetic operators` reference page for more information.

upperbound

Purpose

Upper bound of range of `fi` object

Syntax

`upperbound(a)`

Description

`upperbound(a)` returns the upper bound of the range of `fi` object `a`. If `L = lowerbound(a)` and `U = upperbound(a)`, then `[L,U] = range(a)`.

See Also

`eps`, `intmax`, `intmin`, `lowerbound`, `lsb`, `range`, `realmax`, `realmin`

Purpose

Vertically concatenate multiple `fi` objects

Syntax

```
c = vertcat(a,b,...)
[a; b; ...]
[a;b]
```

Description

`c = vertcat(a,b,...)` is called for the syntax `[a; b; ...]` when any of `a`, `b`, `...` , is a `fi` object.

`[a;b]` is the vertical concatenation of matrices `a` and `b`. `a` and `b` must have the same number of columns. Any number of matrices can be concatenated within one pair of brackets. N-D arrays are vertically concatenated along the first dimension. The remaining dimensions must match.

Horizontal and vertical concatenation can be combined, as in `[1 2;3 4]`.

`[a b; c]` is allowed if the number of rows of `a` equals the number of rows of `b`, and if the number of columns of `a` plus the number of columns of `b` equals the number of columns of `c`.

The matrices in a concatenation expression can themselves be formed via a concatenation, as in `[a b;[c d]]`.

Note The `fimath` and `numerictype` objects of a concatenated matrix of `fi` objects `c` are taken from the leftmost `fi` object in the list `(a,b,...)`.

See Also

`horzcat`

voronoi

Purpose Create Voronoi diagram

Description Refer to the MATLAB `voronoi` reference page for more information.

Purpose Create n-D Voronoi diagram

Description Refer to the MATLAB `voronoin` reference page for more information.

waterfall

Purpose Create waterfall plot

Description Refer to the MATLAB `waterfall` reference page for more information.

Purpose Word length of `quantizer` object

Syntax `wordlength(q)`

Description `wordlength(q)` returns the word length of the `quantizer` object `q`.

Examples

```
q = quantizer([16 15]);
wordlength(q)

ans =

    16
```

See Also `fi`, `fractionlength`, `exponentlength`, `numerictype`, `quantizer`

xlim

Purpose Set or query x-axis limits

Description Refer to the MATLAB `xlim` reference page for more information.

Purpose Set or query y-axis limits

Description Refer to the MATLAB `ylim` reference page for more information.

zlim

Purpose Set or query z-axis limits

Description Refer to the MATLAB `zlim` reference page for more information.

Glossary

This glossary defines terms related to fixed-point data types and numbers. These terms may appear in some or all of the documents that describe products from The MathWorks that have fixed-point support.

arithmetic shift

Shift of the bits of a binary word for which the sign bit is recycled for each bit shift to the right. A zero is incorporated into the least significant bit of the word for each bit shift to the left. In the absence of overflows, each arithmetic shift to the right is equivalent to a division by 2, and each arithmetic shift to the left is equivalent to a multiplication by 2.

See also binary point, binary word, bit, logical shift, most significant bit

bias

Part of the numerical representation used to interpret a fixed-point number. Along with the slope, the bias forms the scaling of the number. Fixed-point numbers can be represented as

$$real\text{-}world\ value = (slope \times integer) + bias$$

where the slope can be expressed as

$$slope = fractional\,slope \times 2^{exponent}$$

See also fixed-point representation, fractional slope, integer, scaling, slope, [Slope Bias]

binary number

Value represented in a system of numbers that has two as its base and that uses 1's and 0's (bits) for its notation.

See also bit

binary point

Symbol in the shape of a period that separates the integer and fractional parts of a binary number. Bits to the left of the binary point are integer bits and/or sign bits, and bits to the right of the binary point are fractional bits.

See also binary number, bit, fraction, integer, radix point

binary point-only scaling

Scaling of a binary number that results from shifting the binary point of the number right or left, and which therefore can only occur by powers of two.

See also binary number, binary point, scaling

binary word

Fixed-length sequence of bits (1's and 0's). In digital hardware, numbers are stored in binary words. The way in which hardware components or software functions interpret this sequence of 1's and 0's is described by a data type.

See also bit, data type, word

bit

Smallest unit of information in computer software or hardware. A bit can have the value 0 or 1.

ceiling (round toward)

Rounding mode that rounds to the closest representable number in the direction of positive infinity. This is equivalent to the `ceil` mode in Fixed-Point Toolbox.

See also convergent rounding, floor (round toward), nearest (round toward), rounding, truncation, zero (round toward)

contiguous binary point

Binary point that occurs within the word length of a data type. For example, if a data type has four bits, its contiguous binary point must be understood to occur at one of the following five positions:

.0000

0.000

00.00

000.0

0000.

See also data type, noncontiguous binary point, word length

convergent rounding

Rounding mode that rounds to the nearest allowable quantized value. Numbers that are exactly halfway between the two nearest allowable quantized values are rounded up only if the least significant bit (after rounding) would be set to `0`.

See also ceiling (round toward), floor (round toward), nearest (round toward), rounding, truncation, zero (round toward)

data type

Set of characteristics that define a group of values. A fixed-point data type is defined by its word length, its fraction length, and whether it is signed or unsigned. A floating-point data type is defined by its word length and whether it is signed or unsigned.

See also fixed-point representation, floating-point representation, fraction length, word length

data type override

Parameter in the Fixed-Point Tool that allows you to set the output data type and scaling of fixed-point blocks on a system or subsystem level.

See also data type, scaling

exponent

Part of the numerical representation used to express a floating-point or fixed-point number.

1. Floating-point numbers are typically represented as

$$\text{real-world value} = \text{mantissa} \times 2^{\text{exponent}}$$

2. Fixed-point numbers can be represented as

$$\text{real-world value} = (\text{slope} \times \text{integer}) + \text{bias}$$

where the slope can be expressed as

$$\text{slope} = \text{fractional slope} \times 2^{\text{exponent}}$$

The exponent of a fixed-point number is equal to the negative of the fraction length:

$$\text{exponent} = -1 \times \text{fraction length}$$

See also bias, fixed-point representation, floating-point representation, fraction length, fractional slope, integer, mantissa, slope

fixed-point representation

Method for representing numerical values and data types that have a set range and precision.

1. Fixed-point numbers can be represented as

$$\text{real-world value} = (\text{slope} \times \text{integer}) + \text{bias}$$

where the slope can be expressed as

$$\text{slope} = \text{fractional slope} \times 2^{\text{exponent}}$$

The slope and the bias together represent the scaling of the fixed-point number.

2. Fixed-point data types can be defined by their word length, their fraction length, and whether they are signed or unsigned.

See also bias, data type, exponent, fraction length, fractional slope, integer, precision, range, scaling, slope, word length

floating-point representation

Method for representing numerical values and data types that can have changing range and precision.

1. Floating-point numbers can be represented as

$$\textit{real-world value} = \textit{mantissa} \times 2^{\textit{exponent}}$$

2. Floating-point data types are defined by their word length.

See also data type, exponent, mantissa, precision, range, word length

floor (round toward)

Rounding mode that rounds to the closest representable number in the direction of negative infinity.

See also ceiling (round toward), convergent rounding, nearest (round toward), rounding, truncation, zero (round toward)

fraction

Part of a fixed-point number represented by the bits to the right of the binary point. The fraction represents numbers that are less than one.

See also binary point, bit, fixed-point representation

fraction length

Number of bits to the right of the binary point in a fixed-point representation of a number.

See also binary point, bit, fixed-point representation, fraction

fractional slope

Part of the numerical representation used to express a fixed-point number. Fixed-point numbers can be represented as

$$real\text{-}world\ value = (slope \times integer) + bias$$

where the slope can be expressed as

$$slope = fractional\,slope \times 2^{exponent}$$

The term *slope adjustment* is sometimes used as a synonym for fractional slope.

See also bias, exponent, fixed-point representation, integer, slope

guard bits

Extra bits in either a hardware register or software simulation that are added to the high end of a binary word to ensure that no information is lost in case of overflow.

See also binary word, bit, overflow

integer

1. Part of a fixed-point number represented by the bits to the left of the binary point. The integer represents numbers that are greater than or equal to one.

2. Also called the "stored integer." The raw binary number, in which the binary point is assumed to be at the far right of the word. The integer is part of the numerical representation used to express a fixed-point number. Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times integer$$

or

$$real\text{-}world\ value = (slope \times integer) + bias$$

where the slope can be expressed as

$$slope = fractional slope \times 2^{exponent}$$

See also bias, fixed-point representation, fractional slope, integer, real-world value, slope

integer length

Number of bits to the left of the binary point in a fixed-point representation of a number.

See also binary point, bit, fixed-point representation, fraction length, integer

least significant bit (LSB)

Bit in a binary word that can represent the smallest value. The LSB is the rightmost bit in a big-endian-ordered binary word. The weight of the LSB is related to the fraction length according to

$$weight\ of\ LSB = 2^{-fraction\ length}$$

See also big-endian, binary word, bit, most significant bit

logical shift

Shift of the bits of a binary word, for which a zero is incorporated into the most significant bit for each bit shift to the right and into the least significant bit for each bit shift to the left.

See also arithmetic shift, binary point, binary word, bit, most significant bit

mantissa

Part of the numerical representation used to express a floating-point number. Floating-point numbers are typically represented as

$$real\text{-}world\ value = mantissa \times 2^{exponent}$$

See also exponent, floating-point representation

most significant bit (MSB)

Bit in a binary word that can represent the largest value. The MSB is the leftmost bit in a big-endian-ordered binary word.

See also binary word, bit, least significant bit

nearest (round toward)

Rounding mode that rounds to the closest representable number, with the exact midpoint rounded to the closest representable number in the direction of positive infinity. This is equivalent to the `nearest` mode in Fixed-Point Toolbox.

See also ceiling (round toward), convergent rounding, floor (round toward), rounding, truncation, zero (round toward)

noncontiguous binary point

Binary point that is understood to fall outside the word length of a data type. For example, the binary point for the following 4-bit word is understood to occur two bits to the right of the word length,

0000_ _.

thereby giving the bits of the word the following potential values:

$2^5 2^4 2^3 2^2$_ _.

See also binary point, data type, word length

one's complement representation

Representation of signed fixed-point numbers. Negating a binary number in one's complement requires a bitwise complement. That is, all 0's are flipped to 1's and all 1's are flipped to 0's. In one's complement notation there are two ways to represent zero. A binary word of all 0's represents "positive" zero, while a binary word of all 1's represents "negative" zero.

See also binary number, binary word, sign/magnitude representation, signed fixed-point, two's complement representation

overflow

Situation that occurs when the magnitude of a calculation result is too large for the range of the data type being used. In many cases you can choose to either saturate or wrap overflows.

See also saturation, wrapping

padding

Extending the least significant bit of a binary word with one or more zeros.

See also least significant bit

precision

1. Measure of the smallest numerical interval that a fixed-point data type and scaling can represent, determined by the value of the number's least significant bit. The precision is given by the slope, or the number of fractional bits. The term *resolution* is sometimes used as a synonym for this definition.

2. Measure of the difference between a real-world numerical value and the value of its quantized representation. This is sometimes called quantization error or quantization noise.

See also data type, fraction, least significant bit, quantization, quantization error, range, slope

Q format

Representation used by Texas Instruments to encode signed two's complement fixed-point data types. This fixed-point notation takes the form

$$Qm.n$$

where

- Q indicates that the number is in Q format.
- m is the number of bits used to designate the two's complement integer part of the number.

- n is the number of bits used to designate the two's complement fractional part of the number, or the number of bits to the right of the binary point.

In Q format notation, the most significant bit is assumed to be the sign bit.

See also binary point, bit, data type, fixed-point representation, fraction, integer, two's complement

quantization

Representation of a value by a data type that has too few bits to represent it exactly.

See also bit, data type, quantization error

quantization error

Error introduced when a value is represented by a data type that has too few bits to represent it exactly, or when a value is converted from one data type to a shorter data type. Quantization error is also called quantization noise.

See also bit, data type, quantization

radix point

Symbol in the shape of a period that separates the integer and fractional parts of a number in any base system. Bits to the left of the radix point are integer and/or sign bits, and bits to the right of the radix point are fraction bits.

See also binary point, bit, fraction, integer, sign bit

range

Span of numbers that a certain data type can represent.

See also data type, precision

real-world value

Stored integer value with fixed-point scaling applied. Fixed-point numbers can be represented as

$$real\text{-}world\ value = 2^{-fraction\ length} \times integer$$

or

$$real\text{-}world\ value = (slope \times integer) + bias$$

where the slope can be expressed as

$$slope = fractional\,slope \times 2^{exponent}$$

See also integer

resolution

See **precision**

rounding

Limiting the number of bits required to express a number. One or more least significant bits are dropped, resulting in a loss of precision. Rounding is necessary when a value cannot be expressed exactly by the number of bits designated to represent it.

See also bit, ceiling (round toward), convergent rounding, floor (round toward), least significant bit, nearest (round toward), precision, truncation, zero (round toward)

saturation

Method of handling numeric overflow that represents positive overflows as the largest positive number in the range of the data type being used, and negative overflows as the largest negative number in the range.

See also overflow, wrapping

scaled double

A double data type that retains fixed-point scaling information. For example, in Simulink and Fixed-Point Toolbox you can use data type override to convert your fixed-point data types to scaled doubles. You can then simulate to determine the ideal floating-point behavior of your system. After you gather that information you can turn data type override off to return to fixed-point data types, and your quantities still have their original scaling information because it was held in the scaled double data types.

scaling

1. Format used for a fixed-point number of a given word length and signedness. The slope and bias together form the scaling of a fixed-point number.

2. Changing the slope and/or bias of a fixed-point number without changing the stored integer.

See also bias, fixed-point representation, integer, slope

shift

Movement of the bits of a binary word either toward the most significant bit ("to the left") or toward the least significant bit ("to the right"). Shifts to the right can be either logical, where the spaces emptied at the front of the word with each shift are filled in with zeros, or arithmetic, where the word is sign extended as it is shifted to the right.

See also arithmetic shift, logical shift, sign extension

sign bit

Bit (or bits) in a signed binary number that indicates whether the number is positive or negative.

See also binary number, bit

sign extension

Addition of bits that have the value of the most significant bit to the high end of a two's complement number. Sign extension does not change the value of the binary number.

See also binary number, guard bits, most significant bit, two's complement representation, word

sign/magnitude representation

Representation of signed fixed-point or floating-point numbers. In sign/magnitude representation, one bit of a binary word is always the dedicated sign bit, while the remaining bits of the word encode the magnitude of the number. Negation using sign/magnitude representation consists of flipping the sign bit from 0 (positive) to 1 (negative), or from 1 to 0.

See also binary word, bit, fixed-point representation, floating-point representation, one's complement representation, sign bit, signed fixed-point, two's complement representation

signed fixed-point

Fixed-point number or data type that can represent both positive and negative numbers.

See also data type, fixed-point representation, unsigned fixed-point

slope

Part of the numerical representation used to express a fixed-point number. Along with the bias, the slope forms the scaling of a fixed-point number. Fixed-point numbers can be represented as

$$real\text{-}world\ value = (slope \times integer) + bias$$

where the slope can be expressed as

$$slope = fractional\,slope \times 2^{exponent}$$

See also bias, fixed-point representation, fractional slope, integer, scaling, [Slope Bias]

slope adjustment

See **fractional slope**

[Slope Bias]

Representation used to define the scaling of a fixed-point number.

See also bias, scaling, slope

stored integer

See **integer**

trivial scaling

Scaling that results in the real-world value of a number being simply equal to its stored integer value:

$$\text{real-world value} = integer$$

In [Slope Bias] representation, fixed-point numbers can be represented as

$$\text{real-world value} = (slope \times integer) + bias$$

In the trivial case, slope = 1 and bias = 0.

In terms of binary point-only scaling, the binary point is to the right of the least significant bit for trivial scaling, meaning that the fraction length is zero:

$$\text{real-world value} = integer \times 2^{-\text{fraction length}} = integer \times 2^{0}$$

Scaling is always trivial for pure integers, such as `int8`, and also for the true floating-point types `single` and `double`.

See also bias, binary point, binary point-only scaling, fixed-point representation, fraction length, integer, least significant bit, scaling, slope, [Slope Bias]

truncation

Rounding mode that drops one or more least significant bits from a number.

See also ceiling (round toward), convergent rounding, floor (round toward), nearest (round toward), rounding, zero (round toward)

two's complement representation

Common representation of signed fixed-point numbers. Negation using signed two's complement representation consists of a translation into one's complement followed by the binary addition of a one.

See also binary word, one's complement representation, sign/magnitude representation, signed fixed-point

unsigned fixed-point

Fixed-point number or data type that can only represent numbers greater than or equal to zero.

See also data type, fixed-point representation, signed fixed-point

word

Fixed-length sequence of binary digits (1's and 0's). In digital hardware, numbers are stored in words. The way hardware components or software functions interpret this sequence of 1's and 0's is described by a data type.

See also binary word, data type

word length

Number of bits in a binary word or data type.

See also binary word, bit, data type

wrapping

Method of handling overflow. Wrapping uses modulo arithmetic to cast a number that falls outside of the representable range the data type being used back into the representable range.

See also data type, overflow, range, saturation

zero (round toward)

Rounding mode that rounds to the closest representable number in the direction of zero. This is equivalent to the `fix` mode in Fixed-Point Toolbox.

See also ceiling (round toward), convergent rounding, floor (round toward), nearest (round toward), rounding, truncation

Index

Q

R

X

Y

Z