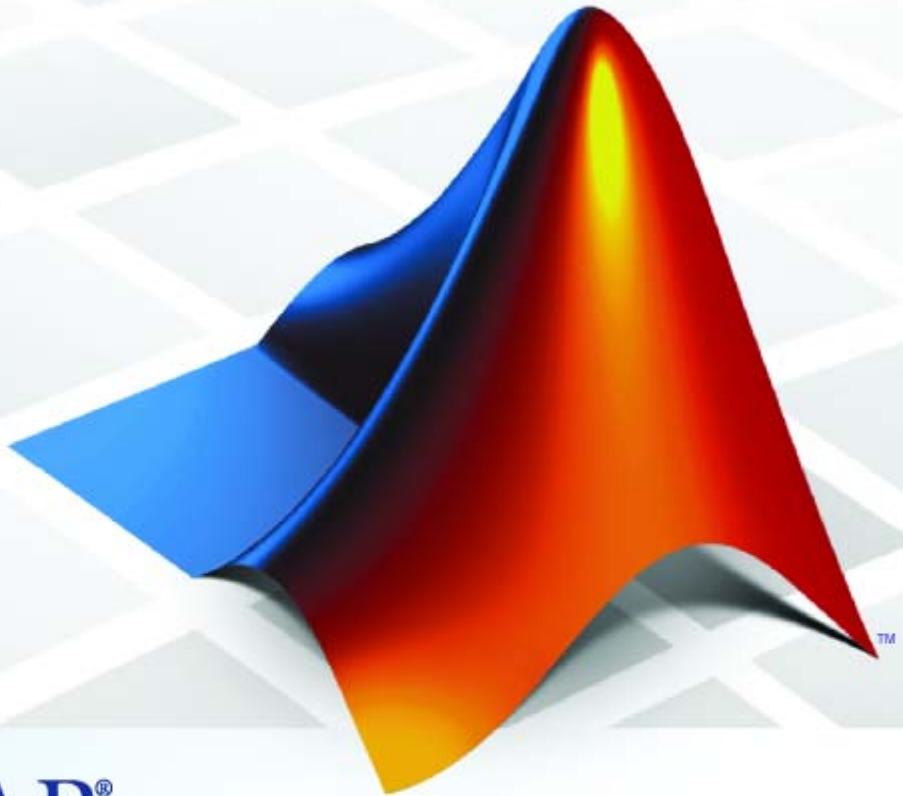


Fixed-Point Toolbox™ 2

Reference



MATLAB®

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

© COPYRIGHT 2004–2009 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

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)
September 2007	Online only	Revised for Version 2.1 (R2007b)
March 2008	Online only	Revised for Version 2.2 (R2008a)
October 2008	Online only	Revised for Version 2.3 (R2008b)
March 2009	Online only	Revised for Version 2.4 (R2009a)

Property Reference

1

fi Object Properties	1-2
bin	1-2
data	1-2
dec	1-2
double	1-2
fimath	1-2
hex	1-3
int	1-3
NumericType	1-3
oct	1-3
fimath Object Properties	1-4
CastBeforeSum	1-4
MaxProductWordLength	1-4
MaxSumWordLength	1-4
OverflowMode	1-4
ProductBias	1-5
ProductFixedExponent	1-5
ProductFractionLength	1-5
ProductMode	1-5
ProductSlope	1-7
ProductSlopeAdjustmentFactor	1-7
ProductWordLength	1-7
RoundMode	1-8
SumBias	1-8
SumFixedExponent	1-8
SumFractionLength	1-9
SumMode	1-9
SumSlope	1-11
SumSlopeAdjustmentFactor	1-11
SumWordLength	1-11
fipref Object Properties	1-12
DataTypeOverride	1-12
FimathDisplay	1-12

LoggingMode	1-12
NumericTypeDisplay	1-13
NumberDisplay	1-13
numericType Object Properties	1-15
Bias	1-15
DataType	1-15
DataTypeMode	1-15
FixedExponent	1-16
FractionLength	1-17
Scaling	1-17
Signed	1-17
Signedness	1-18
Slope	1-18
SlopeAdjustmentFactor	1-18
WordLength	1-19
quantizer Object Properties	1-20
DataMode	1-20
Format	1-20
OverflowMode	1-21
RoundMode	1-22

Function Reference

2

Bitwise Operations	2-2
Constructors and Properties	2-3
Data Manipulation	2-4
Data Type Operations	2-6
Data Quantizing	2-7
Element-Wise Logical Operators	2-8

Math Operations	2-8
Matrix Manipulation	2-10
Plots	2-12
Radix Conversion	2-15
Relational Operators	2-16
Statistics	2-16
Subscripted Assignment and Reference	2-17
fi Object Operations	2-18
fimath Object Operations	2-29
fipref Object Operations	2-30
numerictype Object Operations	2-31
quantizer Object Operations	2-32

Functions — Alphabetical List

3

Glossary

Index

Property Reference

- “fi Object Properties” on page 1-2
- “fimath Object Properties” on page 1-4
- “fipref Object Properties” on page 1-12
- “numerictype Object Properties” on page 1-15
- “quantizer Object Properties” on page 1-20

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 1-4 and “`numericType` Object Properties” on page 1-15 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 MATLAB factory 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 `fi` objects, refer to “Working with `fi` Objects”. For more information about each of the `fi` object properties, refer to “`fi` Object Properties” on page 1-4.

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`, `int64`, `uint8`, `uint16`, `uint32`, and `uint64` to get the stored integer value of a `fi` object in these formats.

NumericType

The `numericType` object contains all the data type and scaling attributes of a fixed-point object. The `numericType` object behaves like any MATLAB structure, except that it only lets you set valid values for defined fields. For a table of the possible settings of each field of the structure, see “Valid Values for `numericType` Structure Properties” in the *Fixed-Point Toolbox™ User’s Guide*.

Note 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 MATLAB factory default value of this property is 1 (true).

MaxProductWordLength

Maximum allowable word length for the product data type.

The MATLAB factory default value of this property is 128.

MaxSumWordLength

Maximum allowable word length for the sum data type.

The MATLAB factory 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 MATLAB factory 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 MATLAB factory 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.

$$ProductSlope = ProductSlopeAdjustmentFactor \times 2^{ProductFixedExponent}$$
 Changing one of these properties changes the others.

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

The MATLAB factory 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 MATLAB factory 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.

$$W_p = \text{specified in the ProductWordLength property}$$

$$F_p = F_a + F_b$$

- **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.

$$W_p = \text{specified in the ProductWordLength property}$$

$$F_p = W_p - \text{integer length}$$

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.

$$W_p = \text{specified in the ProductWordLength property}$$

$$F_p = \text{specified in the ProductFractionLength property}$$

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

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

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

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

The MATLAB factory 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.

$$ProductSlope = ProductSlopeAdjustmentFactor \times 2^{ProductFixedExponent}$$
 .
Changing one of these properties changes the others.

The MATLAB factory 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.

$$ProductSlope = ProductSlopeAdjustmentFactor \times 2^{ProductFixedExponent}$$
 .
Changing one of these properties changes the others.

The MATLAB factory 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 MATLAB factory 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 toward nearest. Ties round to the nearest even stored integer. This is the least biased rounding method provided by Fixed-Point Toolbox software.
- `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 MATLAB factory default value of this property is `nearest`.

See “Rounding Methods” in the Fixed-Point Toolbox User’s Guide for more information.

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 MATLAB factory 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

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

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

The MATLAB factory 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 MATLAB factory 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 $\text{ceil}(\log_2(\text{NumberOfSummands})) = 1$. 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

$$\text{integer length} = \max(W_a - F_a, W_b - F_b) + \text{ceil}(\log_2(\text{NumberOfSummands}))$$

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

- **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.

$$W_s = \text{specified in the SumWordLength property}$$

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

- **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.

$$W_s = \text{specified in the SumWordLength property}$$

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

where

$$\text{integer length} = \max(W_a - F_a, W_b - F_b) + \text{ceil}(\log_2(\text{NumberOfSummands}))$$

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

$$W_s = \text{specified in the SumWordLength property}$$

$$F_s = \text{specified in the SumFractionLength property}$$

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

$$S_s = \text{specified in the SumSlope property}$$

$$B_s = \text{specified in the SumBias property}$$

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

The MATLAB factory 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.

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

The MATLAB factory 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.

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

The MATLAB factory 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 MATLAB factory 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 is `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

$$\frac{1}{(2^{\text{fixed-point exponent}})} \times \text{stored integer}$$

numericType Object Properties

This section describes the properties associated with numericType objects.

Bias

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

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

where the slope can be expressed as

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

DataType

The possible value of the DataType 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 the 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 the object. The exponent is part of the numerical representation used to express a fixed-point number. Fixed-point numbers can be represented as

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

where the slope can be expressed as

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

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

$$\textit{fixed exponent} = -\textit{fraction length}$$

FractionLength

Fraction length of the stored integer value of the object, in bits. The fraction length can be any integer value.

This property automatically defaults to the best precision possible based on the value of the word length and the real-world value of the `fi` object.

Scaling

Scaling mode of the 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, to allow for the automatic assignment of a binary point best-precision scaling.

The default value of this property is `BinaryPoint`.

Signed

Whether the 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`.

Note Although the `Signed` property is still supported, the `Signedness` property always appears in the `numericType` object display. If you choose to change or set the signedness of your `numericType` objects using the `Signed` property, MATLAB updates the corresponding value of the `Signedness` property.

Signedness

Whether the object is signed, unsigned, or has an unspecified sign. The possible values of this property are:

- Signed — signed
- Unsigned — unsigned
- Auto — unspecified sign

The default value of this property is `Signed`.

All `numeric_type` object properties of a `fi` object must be specified at the time of `fi` object creation. If this property is set to `Auto` at the time of `fi` object creation, the property automatically defaults to `Signed`.

Slope

Slope associated with the 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

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

where the slope can be expressed as

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

SlopeAdjustmentFactor

Slope adjustment associated with the 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

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

where the slope can be expressed as

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

WordLength

Word length of the stored integer value of the 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	<p>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.</p> <p>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.</p>
float	<p>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.</p> <p>The word length can range from 2 to the limits of memory on your PC. The exponent length can range from 0 to 11.</p>
double	<p>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.</p>
single	<p>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.</p>

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`.

Function Reference

Bitwise Operations (p. 2-2)	Operate on and manipulate bits
Constructors and Properties (p. 2-3)	Create and manipulate objects and properties
Data Manipulation (p. 2-4)	Manipulate and get information about objects
Data Type Operations (p. 2-6)	Convert objects or values to different data types
Data Quantizing (p. 2-7)	Quantize data
Element-Wise Logical Operators (p. 2-8)	Get information about array elements
Math Operations (p. 2-8)	Operate on objects
Matrix Manipulation (p. 2-10)	Manipulate and get information about arrays
Plots (p. 2-12)	Create plots
Radix Conversion (p. 2-15)	Binary point representations and conversions
Relational Operators (p. 2-16)	Compare real-world values of objects
Statistics (p. 2-16)	Get statistical information about objects
Subscripted Assignment and Reference (p. 2-17)	Get and set array elements
fi Object Operations (p. 2-18)	All functions that operate directly on <code>fi</code> objects

fimath Object Operations (p. 2-29)	All functions that operate directly on fimath objects
fipref Object Operations (p. 2-30)	All functions that operate directly on fipref objects
numerictype Object Operations (p. 2-31)	All functions that operate directly on numerictype objects
quantizer Object Operations (p. 2-32)	All functions that operate directly on quantizer objects

Bitwise Operations

bitand	Bitwise AND of two <code>fi</code> objects
bitandreduce	Bitwise AND of consecutive range of bits
bitcmp	Bitwise complement of <code>fi</code> object
bitconcat	Concatenate bits of <code>fi</code> objects
bitget	Bit at certain position
bitor	Bitwise OR of two <code>fi</code> objects
bitorreduce	Bitwise OR of consecutive range of bits
bitreplicate	Replicate and concatenate bits of <code>fi</code> object
bitrol	Bitwise rotate left
bitror	Bitwise rotate right
bitset	Set bit at certain position
bitshift	Shift bits specified number of places
bitsliceget	Consecutive slice of bits
bitsll	Bit shift left logical
bitsra	Bit shift right arithmetic

<code>bitshr1</code>	Bit shift right logical
<code>bitxor</code>	Bitwise exclusive OR of two <code>fi</code> objects
<code>bitxorreduce</code>	Bitwise exclusive OR of consecutive range of bits
<code>getlsb</code>	Least significant bit
<code>getmsb</code>	Most significant bit

Constructors and Properties

<code>assignmentquantizer</code>	Assignment quantizer object of <code>fi</code> object
<code>copyobj</code>	Make independent copy of quantizer object
<code>fi</code>	Construct fixed-point numeric object
<code>fimath</code>	Construct <code>fimath</code> object
<code>fipref</code>	Construct <code>fipref</code> object
<code>get</code>	Property values of object
<code>numericity</code>	Construct <code>numericity</code> object
<code>quantizer</code>	Construct quantizer object
<code>reset</code>	Reset objects to initial conditions
<code>resetdefaultfimath</code>	Set default <code>fimath</code> object to MATLAB factory default
<code>savedefaultfimathpref</code>	Save default <code>fimath</code> object for next MATLAB session
<code>savefipref</code>	Save <code>fi</code> preferences for next MATLAB session
<code>set</code>	Set or display property values for quantizer objects
<code>setdefaultfimath</code>	Set MATLAB default <code>fimath</code> object

<code>tostring</code>	Convert <code>numeric</code> type or quantizer object to string
<code>unitquantizer</code>	Constructor for <code>unitquantizer</code> object

Data Manipulation

<code>assignmentquantizer</code>	Assignment quantizer object of <code>fi</code> object
<code>denormalmax</code>	Largest denormalized quantized number for quantizer object
<code>denormalmin</code>	Smallest denormalized quantized number for quantizer object
<code>eps</code>	Quantized relative accuracy for <code>fi</code> or quantizer objects
<code>exponentbias</code>	Exponent bias for quantizer object
<code>exponentlength</code>	Exponent length of quantizer object
<code>exponentmax</code>	Maximum exponent for quantizer object
<code>exponentmin</code>	Minimum exponent for quantizer object
<code>fractionlength</code>	Fraction length of quantizer object
<code>intmax</code>	Largest positive stored integer value representable by <code>numeric</code> type of <code>fi</code> object
<code>intmin</code>	Smallest stored integer value representable by <code>numeric</code> type of <code>fi</code> object
<code>isboolean</code>	Determine whether input is Boolean
<code>isdouble</code>	Determine whether input is double-precision data type

<code>isequal</code>	Determine whether real-world values of two <code>fi</code> objects are equal, or determine whether properties of two <code>fimath</code> , <code>numerictype</code> , or <code>quantizer</code> objects are equal
<code>isfi</code>	Determine whether variable is <code>fi</code> object
<code>isfimath</code>	Determine whether variable is <code>fimath</code> object
<code>isfipref</code>	Determine whether input is <code>fipref</code> object
<code>isfixed</code>	Determine whether input is fixed-point data type
<code>isfloat</code>	Determine whether input is floating-point data type
<code>isnumerictype</code>	Determine whether input is <code>numerictype</code> object
<code>ispropequal</code>	Determine whether properties of two <code>fi</code> objects are equal
<code>isquantizer</code>	Determine whether input is <code>quantizer</code> object
<code>isscaleddouble</code>	Determine whether input is scaled double data type
<code>isscaledtype</code>	Determine whether input is fixed-point or scaled double data type
<code>issigned</code>	Determine whether <code>fi</code> object is signed
<code>issingle</code>	Determine whether input is single-precision data type
<code>isslopebiasscaled</code>	Determine whether <code>numerictype</code> object has nontrivial slope and bias
<code>lowerbound</code>	Lower bound of range of <code>fi</code> object

<code>lsb</code>	Scaling of least significant bit of <code>fi</code> object, or value of least significant bit of quantizer object
<code>range</code>	Numerical range of <code>fi</code> or quantizer object
<code>realmax</code>	Largest positive fixed-point value or quantized number
<code>realmin</code>	Smallest positive normalized fixed-point value or quantized number
<code>sort</code>	Sort elements of real-valued <code>fi</code> object in ascending or descending order
<code>upperbound</code>	Upper bound of range of <code>fi</code> object
<code>wordlength</code>	Word length of quantizer object

Data Type Operations

<code>double</code>	Double-precision floating-point real-world value of <code>fi</code> object
<code>int</code>	Smallest built-in integer fitting stored integer value of <code>fi</code> object
<code>int16</code>	Stored integer value of <code>fi</code> object as built-in <code>int16</code>
<code>int32</code>	Stored integer value of <code>fi</code> object as built-in <code>int32</code>
<code>int64</code>	Stored integer value of <code>fi</code> object as built-in <code>int64</code>
<code>int8</code>	Stored integer value of <code>fi</code> object as built-in <code>int8</code>
<code>logical</code>	Convert numeric values to logical

<code>reinterpretcast</code>	Convert fixed-point data types without changing underlying data
<code>rescale</code>	Change scaling of <code>fi</code> object
<code>single</code>	Single-precision floating-point real-world value of <code>fi</code> object
<code>stripscaling</code>	Stored integer of <code>fi</code> object
<code>uint16</code>	Stored integer value of <code>fi</code> object as built-in <code>uint16</code>
<code>uint32</code>	Stored integer value of <code>fi</code> object as built-in <code>uint32</code>
<code>uint64</code>	Stored integer value of <code>fi</code> object as built-in <code>uint64</code>
<code>uint8</code>	Stored integer value of <code>fi</code> object as built-in <code>uint8</code>

Data Quantizing

<code>quantize</code>	Apply quantizer object to data
<code>randquant</code>	Generate uniformly distributed, quantized random number using quantizer object
<code>round</code>	Round <code>fi</code> object toward nearest integer or round input data using quantizer object
<code>unitquantize</code>	Quantize except numbers within <code>eps</code> of +1
<code>unitquantizer</code>	Constructor for <code>unitquantizer</code> object

Element-Wise Logical Operators

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

Math Operations

<code>abs</code>	Absolute value of <code>fi</code> object
<code>add</code>	Add two objects using <code>fimath</code> object
<code>ceil</code>	Round toward positive infinity
<code>complex</code>	Construct complex <code>fi</code> object from real and imaginary parts
<code>conj</code>	Complex conjugate of <code>fi</code> object
<code>convergent</code>	Round toward nearest integer with ties rounding to nearest even integer
<code>divide</code>	Divide two objects
<code>fix</code>	Round toward zero
<code>floor</code>	Round toward negative infinity
<code>imag</code>	Imaginary part of complex number
<code>innerprodintbits</code>	Number of integer bits needed for fixed-point inner product
<code>minus</code>	Matrix difference between <code>fi</code> objects

<code>mpy</code>	Multiply two objects using <code>fimath</code> object
<code>mrdivide</code>	Forward slash (/) or right-matrix division
<code>mtimes</code>	Matrix product of <code>fi</code> objects
<code>nearest</code>	Round toward nearest integer with ties rounding toward positive infinity
<code>plus</code>	Matrix sum of <code>fi</code> objects
<code>pow2</code>	Efficient fixed-point multiplication by 2^K
<code>rdivide</code>	Right-array division (<code>./</code>)
<code>real</code>	Real part of complex number
<code>round</code>	Round <code>fi</code> object toward nearest integer or round input data using quantizer object
<code>sign</code>	Perform signum function on array
<code>sqrt</code>	Square root of <code>fi</code> object
<code>sub</code>	Subtract two objects using <code>fimath</code> object
<code>sum</code>	Sum of array elements
<code>times</code>	Element-by-element multiplication of <code>fi</code> objects
<code>uminus</code>	Negate elements of <code>fi</code> object array
<code>uplus</code>	Unary plus

Matrix Manipulation

<code>buffer</code>	Buffer signal vector into matrix of data frames
<code>ctranspose</code>	Complex conjugate transpose of <code>fi</code> object
<code>diag</code>	Diagonal matrices or diagonals of matrix
<code>disp</code>	Display object
<code>end</code>	Last index of array
<code>flipdim</code>	Flip array along specified dimension
<code>fliplr</code>	Flip matrix left to right
<code>flipud</code>	Flip matrix up to down
<code>hankel</code>	Hankel matrix
<code>horzcat</code>	Horizontally concatenate multiple <code>fi</code> objects
<code>ipermute</code>	Inverse permute dimensions of multidimensional array
<code>iscolumn</code>	Determine whether <code>fi</code> object is column vector
<code>isempty</code>	Determine whether array is empty
<code>isfinite</code>	Determine whether array elements are finite
<code>isinf</code>	Determine whether array elements are infinite
<code>isnan</code>	Determine whether array elements are NaN
<code>isnumeric</code>	Determine whether input is numeric array
<code>isobject</code>	Determine whether input is MATLAB object

<code>isreal</code>	Determine whether array elements are real
<code>isrow</code>	Determine whether <code>fi</code> object is row vector
<code>isscalar</code>	Determine whether input is scalar
<code>isvector</code>	Determine whether input is vector
<code>length</code>	Vector length
<code>ndims</code>	Number of array dimensions
<code>permute</code>	Rearrange dimensions of multidimensional array
<code>repmat</code>	Replicate and tile array
<code>reshape</code>	Reshape array
<code>shiftdata</code>	Shift data to operate on specified dimension
<code>shiftdim</code>	Shift dimensions
<code>size</code>	Array dimensions
<code>sort</code>	Sort elements of real-valued <code>fi</code> object in ascending or descending order
<code>squeeze</code>	Remove singleton dimensions
<code>toeplitz</code>	Create Toeplitz matrix
<code>transpose</code>	Transpose operation
<code>tril</code>	Lower triangular part of matrix
<code>unshiftdata</code>	Inverse of <code>shiftdata</code>
<code>vertcat</code>	Vertically concatenate multiple <code>fi</code> objects
<code>xor</code>	Logical exclusive-OR

Plots

<code>area</code>	Create filled area 2-D plot
<code>bar</code>	Create vertical bar graph
<code>barh</code>	Create horizontal bar graph
<code>clabel</code>	Create contour plot elevation labels
<code>comet</code>	Create 2-D comet plot
<code>comet3</code>	Create 3-D comet plot
<code>compass</code>	Plot arrows emanating from origin
<code>coneplot</code>	Plot velocity vectors as cones in 3-D vector field
<code>contour</code>	Create contour graph of matrix
<code>contour3</code>	Create 3-D contour plot
<code>contourc</code>	Create two-level contour plot computation
<code>contourf</code>	Create filled 2-D contour plot
<code>errorbar</code>	Plot error bars along curve
<code>etreeplot</code>	Plot elimination tree
<code>ezcontour</code>	Easy-to-use contour plotter
<code>ezcontourf</code>	Easy-to-use filled contour plotter
<code>ezmesh</code>	Easy-to-use 3-D mesh plotter
<code>ezplot</code>	Easy-to-use function plotter
<code>ezplot3</code>	Easy-to-use 3-D parametric curve plotter
<code>ezpolar</code>	Easy-to-use polar coordinate plotter
<code>ezsurf</code>	Easy-to-use 3-D colored surface plotter
<code>ezsurfz</code>	Easy-to-use combination surface/contour plotter

<code>feather</code>	Plot velocity vectors
<code>fplot</code>	Plot function between specified limits
<code>gplot</code>	Plot set of nodes using adjacency matrix
<code>hist</code>	Create histogram plot
<code>histc</code>	Histogram count
<code>line</code>	Create line object
<code>loglog</code>	Create log-log scale plot
<code>mesh</code>	Create mesh plot
<code>meshc</code>	Create mesh plot with contour plot
<code>meshz</code>	Create mesh plot with curtain plot
<code>ndgrid</code>	Generate arrays for N-D functions and interpolation
<code>patch</code>	Create patch graphics object
<code>pcolor</code>	Create pseudocolor plot
<code>plot</code>	Create linear 2-D plot
<code>plot3</code>	Create 3-D line plot
<code>plotmatrix</code>	Draw scatter plots
<code>plotyy</code>	Create graph with y-axes on right and left sides
<code>polar</code>	Plot polar coordinates
<code>quiver</code>	Create quiver or velocity plot
<code>quiver3</code>	Create 3-D quiver or velocity plot
<code>rgbplot</code>	Plot colormap
<code>ribbon</code>	Create ribbon plot
<code>rose</code>	Create angle histogram
<code>scatter</code>	Create scatter or bubble plot

<code>scatter3</code>	Create 3-D scatter or bubble plot
<code>semilogx</code>	Create semilogarithmic plot with logarithmic x-axis
<code>semilogy</code>	Create semilogarithmic plot with logarithmic y-axis
<code>slice</code>	Create volumetric slice plot
<code>spy</code>	Visualize sparsity pattern
<code>stairs</code>	Create staircase graph
<code>stem</code>	Plot discrete sequence data
<code>stem3</code>	Plot 3-D discrete sequence data
<code>streamribbon</code>	Create 3-D stream ribbon plot
<code>streamslice</code>	Draw streamlines in slice planes
<code>streamtube</code>	Create 3-D stream tube plot
<code>surf</code>	Create 3-D shaded surface plot
<code>surfc</code>	Create 3-D shaded surface plot with contour plot
<code>surf1</code>	Create surface plot with colormap-based lighting
<code>surfnorm</code>	Compute and display 3-D surface normals
<code>text</code>	Create text object in current axes
<code>treeplot</code>	Plot picture of tree
<code>trimesh</code>	Create triangular mesh plot
<code>triplot</code>	Create 2-D triangular plot
<code>trisurf</code>	Create triangular surface plot
<code>triu</code>	Upper triangular part of matrix
<code>voronoi</code>	Create Voronoi diagram
<code>voronoin</code>	Create n-D Voronoi diagram
<code>waterfall</code>	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

bin	Binary representation of stored integer of <code>fi</code> object
bin2num	Convert two's complement binary string to number using <code>quantizer</code> object
dec	Unsigned decimal representation of stored integer of <code>fi</code> object
hex	Hexadecimal representation of stored integer of <code>fi</code> object
hex2num	Convert hexadecimal string to number using <code>quantizer</code> object
num2bin	Convert number to binary string using <code>quantizer</code> object
num2hex	Convert number to hexadecimal equivalent using <code>quantizer</code> object
num2int	Convert number to signed integer
oct	Octal representation of stored integer of <code>fi</code> object
sdec	Signed decimal representation of stored integer of <code>fi</code> object

Relational Operators

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

Statistics

errmean	Mean of quantization error
errpdf	Probability density function of quantization error
errvar	Variance of quantization error
logreport	Quantization report
max	Largest element in array of <code>fi</code> objects
maxlog	Log maximums
min	Smallest element in array of <code>fi</code> objects
minlog	Log minimums

noperations	Number of operations
noverflows	Number of overflows
numberofelements	Number of data elements in <code>fi</code> array
nunderflows	Number of underflows
resetlog	Clear log for <code>fi</code> or quantizer object

Subscripted Assignment and Reference

subsasgn	Subscripted assignment
subsref	Subscripted reference

fi Object Operations

<code>abs</code>	Absolute value of <code>fi</code> object
<code>all</code>	Determine whether all array elements are nonzero
<code>and</code>	Find logical AND of array or scalar inputs
<code>any</code>	Determine whether any array elements are nonzero
<code>area</code>	Create filled area 2-D plot
<code>assignmentquantizer</code>	Assignment quantizer object of <code>fi</code> object
<code>bar</code>	Create vertical bar graph
<code>barh</code>	Create horizontal bar graph
<code>bin</code>	Binary representation of stored integer of <code>fi</code> object
<code>bitand</code>	Bitwise AND of two <code>fi</code> objects
<code>bitandreduce</code>	Bitwise AND of consecutive range of bits
<code>bitcmp</code>	Bitwise complement of <code>fi</code> object
<code>bitconcat</code>	Concatenate bits of <code>fi</code> objects
<code>bitget</code>	Bit at certain position
<code>bitor</code>	Bitwise OR of two <code>fi</code> objects
<code>bitorreduce</code>	Bitwise OR of consecutive range of bits
<code>bitreplicate</code>	Replicate and concatenate bits of <code>fi</code> object
<code>bitrol</code>	Bitwise rotate left
<code>bitror</code>	Bitwise rotate right
<code>bitset</code>	Set bit at certain position

<code>bitshift</code>	Shift bits specified number of places
<code>bitsliceget</code>	Consecutive slice of bits
<code>bitsll</code>	Bit shift left logical
<code>bitsra</code>	Bit shift right arithmetic
<code>bitsrl</code>	Bit shift right logical
<code>bitxor</code>	Bitwise exclusive OR of two <code>fi</code> objects
<code>bitxorreduce</code>	Bitwise exclusive OR of consecutive range of bits
<code>buffer</code>	Buffer signal vector into matrix of data frames
<code>ceil</code>	Round toward positive infinity
<code>clabel</code>	Create contour plot elevation labels
<code>comet</code>	Create 2-D comet plot
<code>comet3</code>	Create 3-D comet plot
<code>compass</code>	Plot arrows emanating from origin
<code>complex</code>	Construct complex <code>fi</code> object from real and imaginary parts
<code>coneplot</code>	Plot velocity vectors as cones in 3-D vector field
<code>conj</code>	Complex conjugate of <code>fi</code> object
<code>contour</code>	Create contour graph of matrix
<code>contour3</code>	Create 3-D contour plot
<code>contourc</code>	Create two-level contour plot computation
<code>contourf</code>	Create filled 2-D contour plot
<code>convergent</code>	Round toward nearest integer with ties rounding to nearest even integer
<code>ctranspose</code>	Complex conjugate transpose of <code>fi</code> object

<code>dec</code>	Unsigned decimal representation of stored integer of <code>fi</code> object
<code>diag</code>	Diagonal matrices or diagonals of matrix
<code>disp</code>	Display object
<code>double</code>	Double-precision floating-point real-world value of <code>fi</code> object
<code>end</code>	Last index of array
<code>eps</code>	Quantized relative accuracy for <code>fi</code> or quantizer objects
<code>eq</code>	Determine whether real-world values of two <code>fi</code> objects are equal
<code>errorbar</code>	Plot error bars along curve
<code>etreeplot</code>	Plot elimination tree
<code>ezcontour</code>	Easy-to-use contour plotter
<code>ezcontourf</code>	Easy-to-use filled contour plotter
<code>ezmesh</code>	Easy-to-use 3-D mesh plotter
<code>ezplot</code>	Easy-to-use function plotter
<code>ezplot3</code>	Easy-to-use 3-D parametric curve plotter
<code>ezpolar</code>	Easy-to-use polar coordinate plotter
<code>ezsurf</code>	Easy-to-use 3-D colored surface plotter
<code>ezsurfz</code>	Easy-to-use combination surface/contour plotter
<code>feather</code>	Plot velocity vectors
<code>fi</code>	Construct fixed-point numeric object
<code>fimath</code>	Construct <code>fimath</code> object
<code>fix</code>	Round toward zero
<code>flipdim</code>	Flip array along specified dimension

<code>fliplr</code>	Flip matrix left to right
<code>flipud</code>	Flip matrix up to down
<code>floor</code>	Round toward negative infinity
<code>fplot</code>	Plot function between specified limits
<code>ge</code>	Determine whether real-world value of one <code>fi</code> object is greater than or equal to another
<code>get</code>	Property values of object
<code>getlsb</code>	Least significant bit
<code>getmsb</code>	Most significant bit
<code>gplot</code>	Plot set of nodes using adjacency matrix
<code>gt</code>	Determine whether real-world value of one <code>fi</code> object is greater than another
<code>hankel</code>	Hankel matrix
<code>hex</code>	Hexadecimal representation of stored integer of <code>fi</code> object
<code>hist</code>	Create histogram plot
<code>histc</code>	Histogram count
<code>horzcat</code>	Horizontally concatenate multiple <code>fi</code> objects
<code>imag</code>	Imaginary part of complex number
<code>innerprodintbits</code>	Number of integer bits needed for fixed-point inner product
<code>int</code>	Smallest built-in integer fitting stored integer value of <code>fi</code> object
<code>int16</code>	Stored integer value of <code>fi</code> object as built-in <code>int16</code>

<code>int32</code>	Stored integer value of <code>fi</code> object as built-in <code>int32</code>
<code>int64</code>	Stored integer value of <code>fi</code> object as built-in <code>int64</code>
<code>int8</code>	Stored integer value of <code>fi</code> object as built-in <code>int8</code>
<code>intmax</code>	Largest positive stored integer value representable by <code>numerictype</code> of <code>fi</code> object
<code>intmin</code>	Smallest stored integer value representable by <code>numerictype</code> of <code>fi</code> object
<code>ipermute</code>	Inverse permute dimensions of multidimensional array
<code>isboolean</code>	Determine whether input is Boolean
<code>iscolumn</code>	Determine whether <code>fi</code> object is column vector
<code>isdouble</code>	Determine whether input is double-precision data type
<code>isempty</code>	Determine whether array is empty
<code>isequal</code>	Determine whether real-world values of two <code>fi</code> objects are equal, or determine whether properties of two <code>fimath</code> , <code>numerictype</code> , or quantizer objects are equal
<code>isfi</code>	Determine whether variable is <code>fi</code> object
<code>isfinite</code>	Determine whether array elements are finite
<code>isfixed</code>	Determine whether input is fixed-point data type
<code>isfloat</code>	Determine whether input is floating-point data type

<code>isinf</code>	Determine whether array elements are infinite
<code>isnan</code>	Determine whether array elements are NaN
<code>isnumeric</code>	Determine whether input is numeric array
<code>isobject</code>	Determine whether input is MATLAB object
<code>ispropequal</code>	Determine whether properties of two <code>fi</code> objects are equal
<code>isreal</code>	Determine whether array elements are real
<code>isrow</code>	Determine whether <code>fi</code> object is row vector
<code>isscalar</code>	Determine whether input is scalar
<code>isscaleddouble</code>	Determine whether input is scaled double data type
<code>isscaledtype</code>	Determine whether input is fixed-point or scaled double data type
<code>issigned</code>	Determine whether <code>fi</code> object is signed
<code>issingle</code>	Determine whether input is single-precision data type
<code>isvector</code>	Determine whether input is vector
<code>le</code>	Determine whether real-world value of <code>fi</code> object is less than or equal to another
<code>length</code>	Vector length
<code>line</code>	Create line object
<code>logical</code>	Convert numeric values to logical
<code>loglog</code>	Create log-log scale plot

logreport	Quantization report
lowerbound	Lower bound of range of <code>fi</code> object
lsb	Scaling of least significant bit of <code>fi</code> object, or value of least significant bit of quantizer object
lt	Determine whether real-world value of one <code>fi</code> object is less than another
max	Largest element in array of <code>fi</code> objects
maxlog	Log maximums
mesh	Create mesh plot
meshc	Create mesh plot with contour plot
meshz	Create mesh plot with curtain plot
min	Smallest element in array of <code>fi</code> objects
minlog	Log minimums
minus	Matrix difference between <code>fi</code> objects
mrdivide	Forward slash (/) or right-matrix division
mtimes	Matrix product of <code>fi</code> objects
ndgrid	Generate arrays for N-D functions and interpolation
ndims	Number of array dimensions
ne	Determine whether real-world values of two <code>fi</code> objects are not equal
nearest	Round toward nearest integer with ties rounding toward positive infinity
not	Find logical NOT of array or scalar input
noverflows	Number of overflows

<code>numberofelements</code>	Number of data elements in <code>fi</code> array
<code>numerictype</code>	Construct <code>numerictype</code> object
<code>numberflows</code>	Number of underflows
<code>oct</code>	Octal representation of stored integer of <code>fi</code> object
<code>or</code>	Find logical OR of array or scalar inputs
<code>patch</code>	Create patch graphics object
<code>pcolor</code>	Create pseudocolor plot
<code>permute</code>	Rearrange dimensions of multidimensional array
<code>plot</code>	Create linear 2-D plot
<code>plot3</code>	Create 3-D line plot
<code>plotmatrix</code>	Draw scatter plots
<code>plotyy</code>	Create graph with y-axes on right and left sides
<code>plus</code>	Matrix sum of <code>fi</code> objects
<code>polar</code>	Plot polar coordinates
<code>pow2</code>	Efficient fixed-point multiplication by 2^K
<code>quantizer</code>	Construct quantizer object
<code>quiver</code>	Create quiver or velocity plot
<code>quiver3</code>	Create 3-D quiver or velocity plot
<code>range</code>	Numerical range of <code>fi</code> or quantizer object
<code>rdivide</code>	Right-array division (<code>./</code>)
<code>real</code>	Real part of complex number
<code>realmax</code>	Largest positive fixed-point value or quantized number

<code>realmin</code>	Smallest positive normalized fixed-point value or quantized number
<code>reinterpretpcast</code>	Convert fixed-point data types without changing underlying data
<code>repmat</code>	Replicate and tile array
<code>rescale</code>	Change scaling of <code>fi</code> object
<code>resetlog</code>	Clear log for <code>fi</code> or quantizer object
<code>reshape</code>	Reshape array
<code>rgbplot</code>	Plot colormap
<code>ribbon</code>	Create ribbon plot
<code>rose</code>	Create angle histogram
<code>round</code>	Round <code>fi</code> object toward nearest integer or round input data using quantizer object
<code>scatter</code>	Create scatter or bubble plot
<code>scatter3</code>	Create 3-D scatter or bubble plot
<code>sdec</code>	Signed decimal representation of stored integer of <code>fi</code> object
<code>semilogx</code>	Create semilogarithmic plot with logarithmic x-axis
<code>semilogy</code>	Create semilogarithmic plot with logarithmic y-axis
<code>shiftdata</code>	Shift data to operate on specified dimension
<code>shiftdim</code>	Shift dimensions
<code>sign</code>	Perform signum function on array
<code>single</code>	Single-precision floating-point real-world value of <code>fi</code> object
<code>size</code>	Array dimensions

<code>slice</code>	Create volumetric slice plot
<code>sort</code>	Sort elements of real-valued <code>fi</code> object in ascending or descending order
<code>spy</code>	Visualize sparsity pattern
<code>sqrt</code>	Square root of <code>fi</code> object
<code>squeeze</code>	Remove singleton dimensions
<code>stairs</code>	Create staircase graph
<code>stem</code>	Plot discrete sequence data
<code>stem3</code>	Plot 3-D discrete sequence data
<code>streamribbon</code>	Create 3-D stream ribbon plot
<code>streamslice</code>	Draw streamlines in slice planes
<code>streamtube</code>	Create 3-D stream tube plot
<code>stripscaling</code>	Stored integer of <code>fi</code> object
<code>subsasgn</code>	Subscripted assignment
<code>subsref</code>	Subscripted reference
<code>sum</code>	Sum of array elements
<code>surf</code>	Create 3-D shaded surface plot
<code>surfc</code>	Create 3-D shaded surface plot with contour plot
<code>surf1</code>	Create surface plot with colormap-based lighting
<code>surfnorm</code>	Compute and display 3-D surface normals
<code>text</code>	Create text object in current axes
<code>times</code>	Element-by-element multiplication of <code>fi</code> objects
<code>toeplitz</code>	Create Toeplitz matrix
<code>transpose</code>	Transpose operation
<code>treeplot</code>	Plot picture of tree

<code>tril</code>	Lower triangular part of matrix
<code>trimesh</code>	Create triangular mesh plot
<code>triplot</code>	Create 2-D triangular plot
<code>trisurf</code>	Create triangular surface plot
<code>triu</code>	Upper triangular part of matrix
<code>uint16</code>	Stored integer value of <code>fi</code> object as built-in <code>uint16</code>
<code>uint32</code>	Stored integer value of <code>fi</code> object as built-in <code>uint32</code>
<code>uint64</code>	Stored integer value of <code>fi</code> object as built-in <code>uint64</code>
<code>uint8</code>	Stored integer value of <code>fi</code> object as built-in <code>uint8</code>
<code>uminus</code>	Negate elements of <code>fi</code> object array
<code>unshiftdata</code>	Inverse of <code>shiftdata</code>
<code>uplus</code>	Unary plus
<code>upperbound</code>	Upper bound of range of <code>fi</code> object
<code>vertcat</code>	Vertically concatenate multiple <code>fi</code> objects
<code>voronoi</code>	Create Voronoi diagram
<code>voronoin</code>	Create n-D Voronoi diagram
<code>waterfall</code>	Create waterfall plot
<code>xlim</code>	Set or query x-axis limits
<code>xor</code>	Logical exclusive-OR
<code>ylim</code>	Set or query y-axis limits
<code>zlim</code>	Set or query z-axis limits

fimath Object Operations

<code>add</code>	Add two objects using <code>fimath</code> object
<code>disp</code>	Display object
<code>fimath</code>	Construct <code>fimath</code> object
<code>isequal</code>	Determine whether real-world values of two <code>fi</code> objects are equal, or determine whether properties of two <code>fimath</code> , <code>numericType</code> , or <code>quantizer</code> objects are equal
<code>isfimath</code>	Determine whether variable is <code>fimath</code> object
<code>mpy</code>	Multiply two objects using <code>fimath</code> object
<code>resetdefaultfimath</code>	Set default <code>fimath</code> object to MATLAB factory default
<code>savedefaultfimathpref</code>	Save default <code>fimath</code> object for next MATLAB session
<code>setdefaultfimath</code>	Set MATLAB default <code>fimath</code> object
<code>sqrt</code>	Square root of <code>fi</code> object
<code>sub</code>	Subtract two objects using <code>fimath</code> object

fipref Object Operations

<code>disp</code>	Display object
<code>fipref</code>	Construct <code>fipref</code> object
<code>isfipref</code>	Determine whether input is <code>fipref</code> object
<code>reset</code>	Reset objects to initial conditions
<code>savefipref</code>	Save <code>fi</code> preferences for next MATLAB session

numerictype Object Operations

<code>disp</code>	Display object
<code>divide</code>	Divide two objects
<code>isboolean</code>	Determine whether input is Boolean
<code>isdouble</code>	Determine whether input is double-precision data type
<code>isequal</code>	Determine whether real-world values of two <code>fi</code> objects are equal, or determine whether properties of two <code>fi</code> objects are equal
<code>isfixed</code>	Determine whether input is fixed-point data type
<code>isfloat</code>	Determine whether input is floating-point data type
<code>isnumerictype</code>	Determine whether input is numerictype object
<code>isscaleddouble</code>	Determine whether input is scaled double data type
<code>isscaledtype</code>	Determine whether input is fixed-point or scaled double data type
<code>issingle</code>	Determine whether input is single-precision data type
<code>isslopebiascaled</code>	Determine whether numerictype object has nontrivial slope and bias
<code>sqrt</code>	Square root of <code>fi</code> object
<code>tostring</code>	Convert numerictype or quantizer object to string

quantizer Object Operations

<code>bin2num</code>	Convert two's complement binary string to number using quantizer object
<code>copyobj</code>	Make independent copy of quantizer object
<code>denormalmax</code>	Largest denormalized quantized number for quantizer object
<code>denormalmin</code>	Smallest denormalized quantized number for quantizer object
<code>disp</code>	Display object
<code>eps</code>	Quantized relative accuracy for <code>fi</code> or quantizer objects
<code>errmean</code>	Mean of quantization error
<code>errpdf</code>	Probability density function of quantization error
<code>errvar</code>	Variance of quantization error
<code>exponentbias</code>	Exponent bias for quantizer object
<code>exponentlength</code>	Exponent length of quantizer object
<code>exponentmax</code>	Maximum exponent for quantizer object
<code>exponentmin</code>	Minimum exponent for quantizer object
<code>fractionlength</code>	Fraction length of quantizer object
<code>get</code>	Property values of object
<code>hex2num</code>	Convert hexadecimal string to number using quantizer object

<code>isequal</code>	Determine whether real-world values of two <code>fi</code> objects are equal, or determine whether properties of two <code>fimath</code> , <code>numericType</code> , or <code>quantizer</code> objects are equal
<code>isfixed</code>	Determine whether input is fixed-point data type
<code>isfloat</code>	Determine whether input is floating-point data type
<code>isquantizer</code>	Determine whether input is <code>quantizer</code> object
<code>length</code>	Vector length
<code>lsb</code>	Scaling of least significant bit of <code>fi</code> object, or value of least significant bit of <code>quantizer</code> object
<code>max</code>	Largest element in array of <code>fi</code> objects
<code>maxlog</code>	Log maximums
<code>min</code>	Smallest element in array of <code>fi</code> objects
<code>minlog</code>	Log minimums
<code>noperations</code>	Number of operations
<code>noverflows</code>	Number of overflows
<code>num2bin</code>	Convert number to binary string using <code>quantizer</code> object
<code>num2hex</code>	Convert number to hexadecimal equivalent using <code>quantizer</code> object
<code>num2int</code>	Convert number to signed integer
<code>nunderflows</code>	Number of underflows
<code>quantize</code>	Apply <code>quantizer</code> object to data
<code>quantizer</code>	Construct <code>quantizer</code> object

<code>randquant</code>	Generate uniformly distributed, quantized random number using quantizer object
<code>range</code>	Numerical range of <code>fi</code> or quantizer object
<code>realmax</code>	Largest positive fixed-point value or quantized number
<code>realmin</code>	Smallest positive normalized fixed-point value or quantized number
<code>reset</code>	Reset objects to initial conditions
<code>resetlog</code>	Clear log for <code>fi</code> or quantizer object
<code>round</code>	Round <code>fi</code> object toward nearest integer or round input data using quantizer object
<code>set</code>	Set or display property values for quantizer objects
<code>tostring</code>	Convert <code>numericType</code> or quantizer object to string
<code>unitquantize</code>	Quantize except numbers within <code>eps</code> of +1
<code>unitquantizer</code>	Constructor for <code>unitquantizer</code> object
<code>wordlength</code>	Word length of quantizer object

Functions — Alphabetical List

abs

Purpose Absolute value of `fi` object

Syntax

```
c = abs(a)
c = abs(a,T)
c = abs(a,F)
c = abs(a,T,F)
```

Description

`c = abs(a)` returns the absolute value of `fi` object `a` with the same `numericType` and `fiMath` objects as `a`. Intermediate quantities are calculated using the `fiMath` object of `a`.

`c = abs(a,T)` returns a `fi` object with a value equal to the absolute value of `a`, `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 3-3.

`c = abs(a,F)` returns a `fi` object with a value equal to the absolute value of `a`, `fiMath` object `F`, and the same `numericType` object as `a`. Intermediate quantities are calculated using `fiMath` object `F`.

`c = abs(a,T,F)` returns a `fi` object with a value equal to the absolute value of `a`, `numericType` object `T`, and `fiMath` object `F`. Intermediate quantities are calculated using `fiMath` object `F`. See “Data Type Propagation Rules” on page 3-3.

Note When the Signedness of the input `numericType` object `T` is `Auto`, the `abs` function always returns an `Unsigned fi` object.

`abs` only supports `fi` objects with [Slope Bias] scaling when the bias is zero and the fractional slope is one. `abs` does not support complex `fi` objects of data type `Boolean`.

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.

Data Type Propagation Rules

For syntaxes for which you specify a `numericType` object `T`, the `abs` 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
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

Examples

Example 1

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
 Signedness: Signed
 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
 Signedness: Signed
 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
 Signedness: Signed
 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
 Signedness: Signed
 WordLength: 16
 FractionLength: 8

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

Example 2

The following example shows the difference between the absolute value results for complex and real `fi` inputs that have the most negative value representable by a signed data type when `OverflowMode` is `wrap`.

```
re = fi(-1,1,16,15)
```

```
re =
```

```
-1
```

```
    DataTypeMode: Fixed-point: binary point scaling  
    Signedness: Signed  
    WordLength: 16  
    FractionLength: 15
```

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

```
im = fi(0,1,16,15)
```

```
im =
```

```
0
```

```
    DataTypeMode: Fixed-point: binary point scaling  
    Signedness: Signed  
    WordLength: 16  
    FractionLength: 15
```

```
    RoundMode: nearest  
    OverflowMode: saturate
```

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

a = complex(re,im)

a =

    -1

        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Signed
        WordLength: 16
        FractionLength: 15

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

abs(a,re.numericitytype,fimath('overflowmode','wrap'))

ans =

    1.0000

        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Signed
        WordLength: 16
        FractionLength: 15

        RoundMode: nearest
```

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

abs(re, re.numericity, fimath('overflowmode', 'wrap'))

ans =

    -1
```

```
        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Signed
        WordLength: 16
FractionLength: 15
```

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

Example 3

The following example shows how to specify `numericity` and `fimath` objects as optional arguments to control the result of the `abs` function for real inputs.

```
a = fi(-1,1,6,5, 'overflowmode', 'wrap')

a =

    -1
```



```
        DataTypeMode: Fixed-point: binary point scaling
          Signedness: Signed
          WordLength: 6
        FractionLength: 5
```

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

```
abs(a)
```

```
ans =
```

```
-1
```

```
        DataTypeMode: Fixed-point: binary point scaling
          Signedness: Signed
          WordLength: 6
        FractionLength: 5
```

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

```
f = fimath('overflowmode','saturate')
```

```
f =
```

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

```
abs(a,f)
```

```
ans =
```

```
0.9688
```

```
        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Signed
        WordLength: 6
FractionLength: 5
```

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

```
t = numerictype(a.numerictype, 'signed', false)
```

```
t =
```

```
        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Unsigned
        WordLength: 6
FractionLength: 5
```

```

abs(a,t,f)

ans =

    1

    DataTypeMode: Fixed-point: binary point scaling
    Signedness: Unsigned
    WordLength: 6
    FractionLength: 5

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

```

Example 4

The following example shows how to specify `numerictype` and `fimath` objects as optional arguments to control the result of the `abs` function for complex inputs.

```

a = fi(-1-i,1,16,15,'overflowmode','wrap')

a =

-1.0000 - 1.0000i

    DataTypeMode: Fixed-point: binary point scaling
    Signedness: Signed
    WordLength: 16
    FractionLength: 15

    RoundMode: nearest
    OverflowMode: wrap

```

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

t = numericitytype(a.numericitytype,'signed',false)

t =

        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Unsigned
        WordLength: 16
        FractionLength: 15

abs(a,t)

ans =

    1.4142

        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Unsigned
        WordLength: 16
        FractionLength: 15

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

f = fimath('overflowmode','saturate','summode',...
        'keepLSB','sumwordlength',a.wordlength,...
```

```
'productmode','specifyprecision',...
'productwordlength',a.wordlength,...
'productfractionlength',a.fractionlength)

f =

    RoundMode: nearest
    OverflowMode: saturate
    ProductMode: SpecifyPrecision
    ProductWordLength: 16
    ProductFractionLength: 15
    SumMode: KeepLSB
    SumWordLength: 16
    CastBeforeSum: true

abs(a,t,f)

ans =

    1.4142

    DataTypeMode: Fixed-point: binary point scaling
    Signedness: Unsigned
    WordLength: 16
    FractionLength: 15

    RoundMode: nearest
    OverflowMode: saturate
    ProductMode: SpecifyPrecision
    ProductWordLength: 16
    ProductFractionLength: 15
    SumMode: KeepLSB
    SumWordLength: 16
    CastBeforeSum: true
```

Algorithm

The absolute value y of a real input a is defined as follows:

$$y = a \text{ if } a \geq 0$$

$$y = -a \text{ if } a < 0$$

The absolute value y of a complex input a is related to its real and imaginary parts as follows:

$$y = \sqrt{\text{real}(a)^2 + \text{imag}(a)^2}$$

The `abs` function computes the absolute value of complex inputs as follows:

- 1 Calculate the real and imaginary parts of a using the following equations:

$$\text{re} = \text{real}(a)$$

$$\text{im} = \text{imag}(a)$$

- 2 Compute the squares of `re` and `im` using one of the following objects:
 - The `fimath` object `F` if `F` is specified as an argument.
 - The `fimath` object of `a` if `F` is not specified as an argument.
- 3 Cast the squares of `re` and `im` to unsigned types if the input is signed.
- 4 Add the squares of `re` and `im` using one of the following objects:
 - The `fimath` object `F` if `F` is specified as an argument.
 - The `fimath` object of `a` if `F` is not specified as an argument.
- 5 Compute the square root of the sum computed in step four using the `sqrt` function with the following additional arguments:
 - The `numericType` object `T` if `T` is specified, or the `numericType` object of `a` otherwise.

-
- The `fimath` object `F` if `F` is specified, or the `fimath` object of `a` otherwise.

Note Step three prevents the sum of the squares of the real and imaginary components from being negative. This is important because if either `re` or `im` has the maximum negative value and the `OverflowMode` property is set to `wrap` then an error will occur when taking the square root in step five.

add

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
Signedness: Signed
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`, `mrdivide`, `numericType`, `rdivide`, `sub`, `sum`

all

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.

any

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.

assignmentquantizer

Purpose Assignment quantizer object of `fi` object

Syntax `q = assignmentquantizer(a)`

Description `q = assignmentquantizer(a)` returns the quantizer object `q` that is used in assignment operations for the `fi` object `a`.

See Also `quantize`, `quantizer`

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

Fixed-point numbers can be represented as

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

or, equivalently as

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

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

Examples The following code

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

returns

```
y =
    10000000    01111111

z =
    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 AND of consecutive range of bits

Syntax

```
c = bitandreduce(a)
c = bitandreduce(a, lidx)
c = bitandreduce(a, lidx, ridx)
```

Description

`c = bitandreduce(a)` performs a bitwise AND operation on the entire set of bits in the `fi` object `a` and returns the result as a `u1,0` (unsigned integer of word length 1).

`c = bitandreduce(a, lidx)` performs a bitwise AND operation on a consecutive range of bits starting at position `lidx` and ending at the LSB (the bit at position 1). `lidx` is a constant that represents the position in the range closest to the MSB.

`c = bitandreduce(a, lidx, ridx)` performs a bitwise AND operation on a consecutive range of bits starting at position `lidx` and ending at position `ridx`. `ridx` is a constant that represents the position in the range closest to the LSB.

The `bitandreduce` arguments must satisfy the following condition:

$$a.\text{WordLength} \geq \text{lidx} \geq \text{ridx} \geq 1$$

`a` can be a scalar `fi` object or a vector `fi` object.

`bitandreduce` only supports `fi` objects with fixed-point data types; it does not support inputs with complex data types.

`bitandreduce` supports both signed and unsigned inputs with arbitrary scaling. The sign and scaling properties do not affect the result type and value. `bitandreduce` performs the operation on a two's complement bit representation of the stored integer.

Example

This example shows how to perform a bitwise AND operation on a range of bits of a `fi` object. Consider the following unsigned fixed-point `fi` object with a value 5, word length 4, and fraction length 0:

```
a = fi(5,0,4,0);
```

bitandreduce

```
disp(bin(a))
```

```
0101
```

Get the bitwise AND of the consecutive set of bits starting at position 2 and ending at position 1:

```
disp(bin(bitandreduce(a,2,1)))
```

```
0
```

See Also

bitconcat, bitorreduce, bitsliceget, bitxorreduce

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 numeric type, the bit representation of the stored integer is in two's complement representation.

`bitcmp` only supports `fi` objects with fixed-point data types. `a` can be a scalar `fi` object or a vector `fi` object.

Example This example shows how to get the bitwise complement of a `fi` object. Consider the following unsigned fixed-point `fi` object with a value of 10, word length 4, and fraction length 0:

```
a = fi(10,0,4,0);  
disp(bin(a))
```

```
1010
```

Complement the values of the bits in `a`:

```
c = bitcmp(a);  
disp(bin(c))
```

```
0101
```

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

bitconcat

Purpose Concatenate bits of `fi` objects

Syntax

```
y = bitconcat(a, b)
y = bitconcat([a, b, c])
y = bitconcat(a, b, c, d, ...)
```

Description `y = bitconcat(a, b)` concatenates the bits in the `fi` objects `a` and `b`.

`a` and `b` can both be vectors if the vectors are the same size. If `a` and `b` are vectors, `bitconcat` performs element-wise concatenation. `bitconcat` only supports vector input when both `a` and `b` are vectors.

`y = bitconcat([a, b, c])` performs element-wise concatenation of the bits of `fi` objects `a`, `b`, and `c`, as given by the input vector.

`y = bitconcat(a, b, c, d, ...)` concatenates the bits of the `fi` objects `a`, `b`, `c`, `d`,

`bitconcat` returns an unsigned fixed value with a word length equal to the sum of the word lengths of the input objects and a fraction length of zero. The bit representation of the stored integer is in two's complement representation.

The input `fi` objects can be signed or unsigned. `bitconcat` concatenates signed and unsigned bits the same way.

`bitconcat` only supports `fi` objects with fixed-point data types. `bitconcat` does not support inputs with complex data types. Scaling does not affect the result type and value. `bitconcat` accepts `varargin` number of inputs for concatenation.

Example

This example shows how to get the binary representation of the concatenated bits of two `fi` objects. Consider the following unsigned fixed-point `fi` objects. The first has a value of 5, word length 4, and fraction length 0. The second has a value of 10, word length 4, and fraction length 0:

```
a = fi(5,0,4,0);
disp(bin(a))
```



```
0101
```

```
b = fi(10,0,4,0);  
disp(bin(b))
```

```
1010
```

Concatenate the objects:

```
c = bitconcat(a,b);  
disp(bin(c))
```

```
01011010
```

See Also

bitand, bitcmp, bitor, bitreplicate, bitset, bitsliceget, 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` as a `u1,0` (unsigned integer of word length 1). `bit` must be an integer between 1 and the word length of `a`, inclusive. If `a` has a signed `numericType`, the bit representation of the stored integer is in two's complement representation.

`bitget` only supports `fi` objects with fixed-point data types. `bitget` does not support inputs with complex data types.

`bitget` supports variable indexing. This means that `bit` can be a variable instead of a constant.

`a` and `bit` can be vectors or scalars. `a` and `bit` must be the same size unless one is a scalar. If `a` is a vector and `bit` is a scalar, `c` is a vector of `u1,0` values of the bits at position `bit` in each `fi` object in `a`. If `a` is a scalar and `bit` is a vector, `c` is a vector of `u1,0` values of the bits in `a` at the positions specified in `bit`.

`bit` does not need to be a vector of sequential bit positions.

Examples

Example 1

This example shows how to get the binary representation of the bit at a specific position in a `fi` object. Consider the following unsigned fixed-point `fi` object with a value of 85, word length 8, and fraction length 0:

```
a = fi(85,0,8,0);  
disp(bin(a))
```

```
01010101
```

Get the binary representation of the bit at position 4:

```
bit4 = bitget(a,4);  
disp(bin(bit4))
```

0

Example 2

This example shows how to get the binary representation of the bits at a vector of positions in a `fi` object. Consider the following signed fixed-point `fi` object with a value of 55, word length 16, and best-precision fraction length 9:

```
a = fi(55);  
disp(bin(a))
```

```
0110111000000000
```

Get the binary representation of the bits at positions 16, 14, 12, 10, 8, 6, 4, and 2:

```
bitvec = bitget(a,[16:-2:1]);  
disp(bin(bitvec))
```

```
0 1 1 1 0 0 0 0
```

See Also

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

bitor

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

Bitwise OR of consecutive range of bits

Syntax

```
c = bitorreduce(a)
c = bitorreduce(a, lidx)
c = bitorreduce(a, lidx, ridx)
```

Description

`c = bitorreduce(a)` performs a bitwise OR operation on the entire set of bits in the `fi` object `a` and returns the result as a `u1,0` (unsigned integer of word length 1).

`c = bitorreduce(a, lidx)` performs a bitwise OR operation on a consecutive range of bits starting at position `lidx` and ending at the LSB (the bit at position 1). `lidx` is a constant that represents the position in the range closest to the MSB.

`c = bitorreduce(a, lidx, ridx)` performs a bitwise OR operation on a consecutive range of bits starting at position `lidx` and ending at position `ridx`. `ridx` is a constant that represents the position in the range closest to the LSB.

The `bitorreduce` arguments must satisfy the following condition:

```
a.WordLength >= lidx >= ridx >= 1
```

`a` can be a scalar `fi` object or a vector `fi` object.

`bitorreduce` only supports `fi` objects with fixed-point data types; it does not support inputs with complex data types.

`bitorreduce` supports both signed and unsigned inputs with arbitrary scaling. The sign and scaling properties do not affect the result type and value. `bitorreduce` performs the operation on a two's complement bit representation of the stored integer.

Example

This example shows how to perform a bitwise OR operation on a range of bits of a `fi` object. Consider the following unsigned fixed-point `fi` object with a value 5, word length 4, and fraction length 0:

```
a = fi(5,0,4,0);
```

bitorreduce

```
disp(bin(a))
```

```
0101
```

Get the bitwise OR of the consecutive set of bits starting at position 4 and ending at position 3:

```
disp(bin(bitreduce(a,4,3)))
```

```
1
```

See Also

bitandreduce, bitconcat, bitsliceget, bitxorreduce

Purpose Replicate and concatenate bits of `fi` object

Syntax `c = bitreplicate(a, n)`

Description `c = bitreplicate(a, n)` concatenates the bits in `fi` object `a` `n` times and returns an unsigned fixed value with a word length equal to `n` times the word length of `a` and a fraction length of zero. The bit representation of the stored integer is in two's complement representation.

The input `fi` object can be signed or unsigned. `bitreplicate` concatenates signed and unsigned bits the same way.

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

`bitreplicate` does not support inputs with complex data types.

Sign and scaling of the input `fi` object does not affect the result type and value.

See Also `bitand`, `bitconcat`, `bitget`, `bitset`, `bitor`, `bitsliceget`, `bitxor`

bitrol

Purpose Bitwise rotate left

Syntax `c = bitrol(a, k)`

Description `c = bitrol(a, k)` returns the value of the `fi` object `a` rotated left by `k` bits.

`a` can be a scalar `fi` object or a vector `fi` object. It can be any fixed-point numeric type. The `OverflowMode` and `RoundMode` properties are ignored. `bitrol` operates on both signed and unsigned fixed point inputs and does not check overflow or underflow. `bitrol` rotates bits from the MSB side into the LSB side.

`k` is an integer constant that must be greater than zero. `k` can be greater than the word length of `a`. It is always normalized to `mod(a.WordLength,k)`.

`a` and `c` have the same `fimath` and the `numericType` objects.

Example

This example shows how to rotate the bits of a `fi` object left. Consider the following unsigned fixed-point `fi` object with a value of 10, word length 4, and fraction length 0:

```
a = fi(10,0,4,0);  
disp(bin(a))
```

```
1010
```

Rotate `a` left one bit:

```
disp(bin(bitrol(a,1)))
```

```
0101
```

Rotate `a` left two bits:

```
disp(bin(bitrol(a,2)))
```

```
1010
```


See Also

`bitconcat`, `bitror`, `bitshift`, `bitsliceget`, `bitsll`, `bitsra`, `bitsrl`

bitror

Purpose Bitwise rotate right

Syntax `c = bitror(a, k)`

Description `c = bitror(a, k)` returns the value of the `fi` object `a` rotated right by `k` bits.

`a` can be a scalar `fi` object or a vector `fi` object. It can be any fixed-point numeric type. The `OverflowMode` and `RoundMode` properties are ignored. `bitror` operates on both signed and unsigned fixed point inputs and does not check overflow or underflow. `bitror` rotates bits from the LSB side into the MSB side.

`k` is an integer constant that must be greater than zero. `k` can be greater than the word length of `a`. It is always normalized to `mod(a.WordLength,k)`.

`a` and `c` have the same `fimath` and the `numericType` objects.

Example

This example shows how to rotate the bits of a `fi` object right. Consider the following unsigned fixed-point `fi` object with a value 5, word length 4, and fraction length 0:

```
a = fi(5,0,4,0);  
disp(bin(a))
```

```
0101
```

Rotate a right one bit:

```
disp(bin(bitror(a,1)))
```

```
1010
```

Rotate a right two bits:

```
disp(bin(bitror(a,2)))
```

```
0101
```

See Also

`bitconcat`, `bitrol`, `bitshift`, `bitsliceget`, `bitsll`, `bitsra`, `bitsrl`

bitset

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 have a value 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`, the bit representation of the stored integer is in two's complement representation.
`bitset` only supports `fi` objects with fixed-point data types. `a` can be a scalar `fi` object or a vector `fi` object. `bit` and `v` can be scalars or vectors.

Example This example shows how to set a bit of a `fi` object. Consider the following unsigned fixed-point `fi` object with a value of 5, word length 4, and fraction length 0:

```
a = fi(5,0,4,0);  
disp(bin(a))
```

```
0101
```

Set the bit at position 2 to 1:

```
c = bitset(a,2,1);  
disp(bin(c))
```

```
0111
```

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. The input `fi` object `a` may be a scalar value or a vector and can be any fixed-point numeric type. The output `fi` object `c` has the same numeric type as `a`. `k` must be a scalar value and a MATLAB built-in numeric type.

The `OverflowMode` property of `a` is obeyed, but the `RoundMode` is always `floor`. If obeying the `RoundMode` property of `a` is important, try using the `pow2` function.

When the overflow mode is `saturate` the sign bit is always preserved. The sign bit is also preserved when the overflow mode is `wrap`, and `k` is negative. When the overflow mode is `wrap` and `k` is positive, the sign bit is not preserved.

- When `k` is positive, 0-valued bits are shifted in on the right.
- When `k` is negative, and `a` is unsigned, or a signed and positive `fi` object, 0-valued bits are shifted in on the left.
- When `k` is negative and `a` is a signed and negative `fi` object, 1-valued bits are shifted in on the left.

Example

This example highlights how changing the `OverflowMode` property of the `fi` 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` `fi` 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
```

bitshift

```
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, bitsll, bitsra, bitsrl, bitxor, pow2

bitsliceget

Purpose Consecutive slice of bits

Syntax

```
c = bitsliceget(a)
c = bitsliceget(a, lidx)
c = bitsliceget(a, lidx, ridx)
```

Description `c = bitsliceget(a)` returns the entire set of bits in the `fi` object `a`. If `a` has a signed `numericType`, the bit representation of the stored integer is in two's complement representation.

`c = bitsliceget(a, lidx)` returns a consecutive slice of bits from `a` starting at position `lidx` and ending at the LSB (the bit at position 1). `lidx` is a constant that represents the position in the slice that is closest to the MSB.

`c = bitsliceget(a, lidx, ridx)` returns a consecutive slice of bits from `a` starting at position `lidx` and ending at position `ridx`. `ridx` is a constant that represents the position in the slice that is closest to the LSB.

The `bitsliceget` arguments must satisfy the following condition:

```
a.WordLength >= lidx >= ridx >= 1
```

If `lidx` and `ridx` are equal, `bitsliceget` only slices one bit, and `bitsliceget(a, lidx, ridx)` is the same as `bitget(a, lidx)`.

`bitsliceget` only supports `fi` objects with fixed-point data types. `bitsliceget` always returns a fixed point number with no scaling and with word length equal to slice length, `lidx-ridx+1`.

Example This example shows how to get the binary representation of a specified set of consecutive bits in a `fi` object. Consider the following unsigned fixed-point `fi` object with a value of 85, word length 8, and fraction length 0:

```
a = fi(85,0,8,0);
disp(bin(a))
```



```
01010101
```

Get the binary representation of the consecutive set of bits starting at position 8 and ending at position 3:

```
bits8to3 = bitsliceget(a,8,3);  
disp(bin(bits8to3))
```

```
010101
```

See Also

bitand, bitcmp, bitget, bitor, bitset, bitxor

bitsll

Purpose Bit shift left logical

Syntax `c = bitsll(a, k)`

Description `c = bitsll(a, k)` returns the value of the `fi` object `a` shifted left logical by `k` bits.

`a` can be a scalar `fi` object or a vector `fi` object. It can be any fixed-point numeric type. The `OverflowMode` and `RoundMode` properties are ignored. `bitsll` operates on both signed and unsigned fixed point inputs and does not check overflow or underflow. `bitsll` shifts zeros into the positions of bits that it shifts left.

`k` is an integer constant in the following range:

$$a.\text{WordLength} > k \geq 0$$

`a` and `c` have the same `fimath` and the `numericType` objects.

Example

This example shows how to shift bits using the `bitsll` function. Consider the following unsigned fixed-point `fi` object with a value of 10, word length 4, and fraction length 0:

```
a = fi(10,0,4,0);  
disp(bin(a))
```

```
1010
```

Shift `a` left by one bit:

```
disp(bin(bitsll(a,1)))
```

```
0100
```

Shift `a` left by one more bit:

```
disp(bin(bitsll(a,2)))
```

1000

Unlike the `bitshift` function, the output value does not saturate.

See Also

`bitconcat`, `bitrol`, `bitror`, `bitshift`, `bitsliceget`, `bitsra`, `bitsrl`,
`pow2`

bitsra

Purpose Bit shift right arithmetic

Syntax `c = bitsra(a, k)`

Description `c = bitsra(a, k)` returns the value of the `fi` object `a` shifted right arithmetic by `k` bits.

`a` can be a scalar `fi` object or a vector `fi` object. It can be any fixed-point numeric type. The `OverflowMode` and `RoundMode` properties are ignored. `bitsra` operates on both signed and unsigned fixed point inputs and does not check overflow or underflow. `bitsra` shifts zeros into the positions of bits that it shifts right if the input is unsigned. `bitsra` shifts the MSB into the positions of bits that it shifts right if the input is signed.

`k` is an integer constant in the following range:

$$a.\text{WordLength} > k \geq 0$$

`a` and `c` have the same `fimath` and the `numericType` objects.

Example

This example shows how to shift bits using the `bitsra` function. Consider the following signed fixed-point `fi` object with a value of -8, word length 4, and fraction length 0:

```
a = fi(-8,1,4,0);  
disp(bin(a))
```

```
1000
```

Shift `a` right by one bit:

```
disp(bin(bitsra(a,1)))
```

```
1100
```

`bitsra` shifts the MSB into the position of the bit that it shifts right.

See Also

`bitconcat`, `bitshift`, `bitsliceget`, `bitsll`, `bitsrl`, `pow2`

bitrsl

Purpose Bit shift right logical

Syntax `c = bitrsl(a, k)`

Description `c = bitrsl(a, k)` returns the value of `a` shifted right logical by `k` bits. `a` can be a scalar `fi` object or a vector `fi` object. It can be any fixed-point numeric type. The `OverflowMode` and `RoundMode` properties are ignored. `bitrsl` operates on both signed and unsigned fixed point inputs and does not check overflow or underflow. `bitrsl` shifts zeros into the positions of bits that it shifts right.

`k` is an integer constant in the following range:

$$a.\text{WordLength} > k \geq 0$$

`a` and `c` have the same `fimath` and the `numericType` objects.

Example

This example shows how to shift bits using the `bitrsl` function. Consider the following signed fixed-point `fi` object with a value of -8, word length 4, and fraction length 0:

```
a = fi(-8,1,4,0);  
disp(bin(a))
```

```
1000
```

Shift `a` right by one bit:

```
disp(bin(bitrsl(a,1)))
```

```
0100
```

`bitrsl` shifts a zero into the position of the bit that it shifts right.

See Also

`bitconcat`, `bitrol`, `bitror`, `bitshift`, `bitsliceget`, `bitrsl1`, `bitrslra`, `pow2`

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 `fi`math and the `numeric`type objects of `a` and `b` must be identical. If the `numeric`type 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`

bitxorreduce

Purpose Bitwise exclusive OR of consecutive range of bits

Syntax

```
c = bitxorreduce(a)
c = bitxorreduce(a, lidx)
c = bitxorreduce(a, lidx, ridx)
```

Description `c = bitxorreduce(a)` performs a bitwise exclusive OR operation on the entire set of bits in the `fi` object `a` and returns the result as a `u1,0` (unsigned integer of word length 1).

`c = bitxorreduce(a, lidx)` performs a bitwise exclusive OR operation on a consecutive range of bits starting at position `lidx` and ending at the LSB (the bit at position 1). `lidx` is a constant that represents the position in the range closest to the MSB.

`c = bitxorreduce(a, lidx, ridx)` performs a bitwise exclusive OR operation on a consecutive range of bits starting at position `lidx` and ending at position `ridx`. `ridx` is a constant that represents the position in the range closest to the LSB.

The `bitxorreduce` arguments must satisfy the following condition:

$$a.\text{WordLength} \geq lidx \geq ridx \geq 1$$

`a` can be a scalar `fi` object or a vector `fi` object.

`bitxorreduce` only supports `fi` objects with fixed-point data types; it does not support inputs with complex data types.

`bitxorreduce` supports both signed and unsigned inputs with arbitrary scaling. The sign and scaling properties do not affect the result type and value. `bitxorreduce` performs the operation on a two's complement bit representation of the stored integer.

Example This example shows how to perform a bitwise exclusive OR operation on a range of bits of a `fi` object. Consider the following unsigned fixed-point `fi` object with a value 5, word length 4, and fraction length 0:

0101

Get the bitwise exclusive OR of the consecutive set of bits starting at position 4 and ending at position 2:

1

See Also

bitandreduce, bitconcat, bitorreduce, bitsliceget

buffer

Purpose Buffer signal vector into matrix of data frames

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

Purpose Round toward positive infinity

Syntax `y = ceil(a)`

Description `y = ceil(a)` rounds `fi` object `a` to the nearest integer in the direction of positive infinity and returns the result in `fi` object `y`.

`y` and `a` have the same `fi` object and `DataType` property.

When the `DataType` property of `a` is `single`, `double`, or `boolean`, the `numericType` of `y` is the same as that of `a`.

When the fraction length of `a` is zero or negative, `a` is already an integer, and the `numericType` of `y` is the same as that of `a`.

When the fraction length of `a` is positive, the fraction length of `y` is 0, its sign is the same as that of `a`, and its word length is the difference between the word length and the fraction length of `a` plus one bit. If `a` is signed, then the minimum word length of `y` is 2. If `a` is unsigned, then the minimum word length of `y` is 1.

For complex `fi` objects, the imaginary and real parts are rounded independently.

`ceil` does not support `fi` objects with nontrivial slope and bias scaling. Slope and bias scaling is trivial when the slope is an integer power of 2 and the bias is 0.

Examples

Example 1

The following example demonstrates how the `ceil` function affects the `numericType` properties of a signed `fi` object with a word length of 8 and a fraction length of 3.

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

```
a =
```

```
3.1250
```

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

```
y = ceil(a)
```

```
y =
```

```
4
```

```
DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 6
FractionLength: 0
```

Example 2

The following example demonstrates how the `ceil` function affects the numeric type properties of a signed `fi` object with a word length of 8 and a fraction length of 12.

```
a = fi(0.025,1,8,12)
```

```
a =
```

```
0.0249
```

```
DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 8
FractionLength: 12
```

```
y = ceil(a)
```

```
y =
```

1

```

DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 2
FractionLength: 0
    
```

Example 3

The functions `ceil`, `fix`, and `floor` differ in the way they round `fi` objects:

- The `ceil` function rounds values to the nearest integer toward positive infinity
- The `fix` function rounds values toward zero
- The `floor` function rounds values to the nearest integer toward negative infinity

The following table illustrates these differences for a given `fi` object `a`.

a	ceil(a)	fix(a)	floor(a)
-2.5	-2	-2	-3
-1.75	-1	-1	-2
-1.25	-1	-1	-2
-0.5	0	0	-1
0.5	1	0	0
1.25	2	1	1
1.75	2	1	1
2.5	3	2	2

See Also

`convergent`, `fix`, `floor`, `nearest`, `round`

clabel

Purpose Create contour plot elevation labels

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

Purpose Create 2-D comet plot

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

comet3

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.

complex

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`

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

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

conj

Purpose Complex conjugate of `fi` object

Syntax `conj(a)`

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

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

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.

contour3

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.

contourf

Purpose Create filled 2-D contour plot

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

Purpose Round toward nearest integer with ties rounding to nearest even integer

Syntax
`y = convergent(a)`
`y = convergent(x)`

Description `y = convergent(a)` rounds `fi` object `a` to the nearest integer. In the case of a tie, `convergent(a)` rounds to the nearest even integer.

`y` and `a` have the same `fi` object and `DataType` property.

When the `DataType` property of `a` is `single`, `double`, or `boolean`, the `numericType` of `y` is the same as that of `a`.

When the fraction length of `a` is zero or negative, `a` is already an integer, and the `numericType` of `y` is the same as that of `a`.

When the fraction length of `a` is positive, the fraction length of `y` is 0, its sign is the same as that of `a`, and its word length is the difference between the word length and the fraction length of `a`, plus one bit. If `a` is signed, then the minimum word length of `y` is 2. If `a` is unsigned, then the minimum word length of `y` is 1.

For complex `fi` objects, the imaginary and real parts are rounded independently.

`convergent` does not support `fi` objects with nontrivial slope and bias scaling. Slope and bias scaling is trivial when the slope is an integer power of 2 and the bias is 0.

`y = convergent(x)` rounds the elements of `x` to the nearest integer. In the case of a tie, `convergent(x)` rounds to the nearest even integer.

Examples **Example 1**

The following example demonstrates how the `convergent` function affects the `numericType` properties of a signed `fi` object with a word length of 8 and a fraction length of 3.

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

```
a =
```

convergent

```
3.1250
```

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

```
y = convergent(a)
```

```
y =
```

```
3
```

```
    DataTypeMode: Fixed-point: binary point scaling  
    Signedness: Signed  
    WordLength: 6  
    FractionLength: 0
```

Example 2

The following example demonstrates how the `convergent` function affects the `numericType` properties of a signed `fi` object with a word length of 8 and a fraction length of 12.

```
a = fi(0.025,1,8,12)
```

```
a =
```

```
0.0249
```

```
    DataTypeMode: Fixed-point: binary point scaling  
    Signedness: Signed  
    WordLength: 8  
    FractionLength: 12
```

```
y = convergent(a)
```

y =

0

```

        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Signed
        WordLength: 2
        FractionLength: 0
    
```

Example 3

The functions `convergent`, `nearest` and `round` differ in the way they treat values whose least significant digit is 5:

- The `convergent` function rounds ties to the nearest even integer
- The `nearest` function rounds ties to the nearest integer toward positive infinity
- The `round` function rounds ties to the nearest integer with greater absolute value

The following table illustrates these differences for a given `fi` object `a`.

a	convergent(a)	nearest(a)	round(a)
-3.5	-4	-3	-4
-2.5	-2	-2	-3
-1.5	-2	-1	-2
-0.5	0	0	-1
0.5	0	1	1
1.5	2	2	2
2.5	2	3	3
3.5	4	4	4

convergent

See Also

ceil, fix, floor, nearest, round

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 <code>fi</code> object
Syntax	<code>ctranspose(a)</code>
Description	<code>ctranspose(a)</code> returns the complex conjugate transpose of <code>fi</code> object <code>a</code> . It is also called for the syntax <code>a'</code> .
See Also	<code>transpose</code>

- Purpose** Unsigned decimal representation of stored integer of `fi` object
- Syntax** `dec(a)`
- Description** `dec(a)` returns the stored integer of `fi` object `a` in unsigned decimal format as a string. `dec(a)` is equivalent to `a.dec`.

Fixed-point numbers can be represented as

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

or, equivalently as

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

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

Examples

The code

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

returns

```
y =
    128    127

z =
    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,

$$\text{denormalmax}(q) = \text{realmin}(q) - \text{denormalmin}(q)$$

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

$$\text{denormalmax}(q) = \text{eps}(q)$$

See Also `denormalmin`, `eps`, `quantizer`

Purpose	Smallest denormalized quantized number for quantizer object
Syntax	<code>x = denormalmin(q)</code>
Description	<code>x = denormalmin(q)</code> is the smallest positive denormalized quantized number where <code>q</code> is a quantizer object. Anything smaller than <code>x</code> underflows to zero with respect to the quantizer object <code>q</code> . Denormalized numbers apply only to floating-point format. When <code>q</code> represents a fixed-point number, <code>denormalmin</code> returns <code>eps(q)</code> .
Examples	<pre>q = quantizer('float',[6 3]); x = denormalmin(q) x = 0.0625</pre>
Algorithm	When <code>q</code> is a floating-point quantizer object, $x = 2^{E_{min} - f}$ where E_{min} is equal to <code>exponentmin(q)</code> . When <code>q</code> is a fixed-point quantizer object, $x = \text{eps}(q) = 2^{-f}$ where f is equal to <code>fractionlength(q)</code> .
See Also	<code>denormalmax</code> , <code>eps</code> , <code>quantizer</code>

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`.

If `a` and `b` are both `fi` objects, `c` has the same `fi` object as `a`. If `c` has a `fi` Fixed data type, and any one of the inputs have `fi` floating point data types, then the `fi` floating point is converted into a fixed-point value. Intermediate quantities are calculated using the `fi` object of `a`. See “Data Type Propagation Rules” on page 3-84.

`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. Intermediate quantities are calculated using the `fi` object of the input `fi` object. See “Data Type Propagation Rules” on page 3-84.

If `a` and `b` are both MATLAB built-in doubles, 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.

Data Type Propagation Rules

For syntaxes for which Fixed-Point Toolbox software uses the `numericType` object `T`, the `divide` 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 Objects a and b		Data Type of numeric type object T	Data Type of Output c
Built-in double	Built-in double	Any	Built-in double
fi Fixed	fi Fixed	fi Fixed	Data type of numeric type object T
fi Fixed	fi Fixed	fi double	fi double
fi Fixed	fi Fixed	fi single	fi single
fi Fixed	fi Fixed	fi ScaledDouble	fi ScaledDouble with properties of numeric type object T
fi double	fi double	fi Fixed	fi double
fi double	fi double	fi double	fi double
fi double	fi double	fi single	fi single
fi double	fi double	fi ScaledDouble	fi double
fi single	fi single	fi Fixed	fi single
fi single	fi single	fi double	fi double
fi single	fi single	fi single	fi single
fi single	fi single	fi ScaledDouble	fi single
fi ScaledDouble	fi ScaledDouble	fi Fixed	fi ScaledDouble with properties of numeric type object T

divide

Data Type of Input fi Objects a and b		Data Type of numeric type object T	Data Type of Output c
fi ScaledDouble	fi ScaledDouble	fi double	fi double
fi ScaledDouble	fi ScaledDouble	fi single	fi single
fi ScaledDouble	fi ScaledDouble	fi ScaledDouble	fi ScaledDouble with properties of numeric type object T

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 =

1100110011001100110011001100110011001100110011001100110011010000
00000000000000000000000000000000
      u80,83
```

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 =

11001100110011001100110011001100110011001100110011001100110011001100
110011001100110011001100
```

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);
```


Purpose	Double-precision floating-point real-world value of <code>fi</code> object						
Syntax	<code>double(a)</code>						
Description	<code>double(a)</code> returns the real-world value of a <code>fi</code> object in double-precision floating point. <code>double(a)</code> is equivalent to <code>a.double</code> . Fixed-point numbers can be represented as $\text{real-world value} = 2^{-\text{fraction length}} \times \text{stored integer}$ or, equivalently as $\text{real-world value} = (\text{slope} \times \text{stored integer}) + \text{bias}$						
Examples	The code <pre>a = fi([-1 1],1,8,7); y = double(a) z = a.double</pre> returns <table><tr><td>y =</td><td>-1</td><td>0.9922</td></tr><tr><td>z =</td><td>-1</td><td>0.9922</td></tr></table>	y =	-1	0.9922	z =	-1	0.9922
y =	-1	0.9922					
z =	-1	0.9922					
See Also	<code>single</code>						

end

Purpose Last index of array

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

Purpose	Quantized relative accuracy for <code>fi</code> or quantizer objects
Syntax	<code>eps(obj)</code>
Description	<code>eps(obj)</code> returns the value of the least significant bit of the value of the <code>fi</code> object or quantizer object <code>obj</code> . The result of this function is equivalent to that given by the Fixed-Point Toolbox function <code>lsb</code> .
See Also	<code>intmax</code> , <code>intmin</code> , <code>lowerbound</code> , <code>lsb</code> , <code>range</code> , <code>realmax</code> , <code>realmin</code> , <code>upperbound</code>

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 Mean of quantization error

Syntax `m = errmean(q)`

Description `m = errmean(q)` returns the mean of a uniformly distributed random quantization error that arises from quantizing a signal by quantizer object `q`.

Note The results are not exact when the signal precision is close to the precision of the quantizer.

Examples Find `m`, the mean of the quantization error for quantizer `q`:

```
q = quantizer;  
m = errmean(q)  
  
m =  
  
-1.525878906250000e-005
```

Now compare `m` to `m_est`, the sample mean from a Monte Carlo experiment:

```
r = realmax(q);  
u = 2*r*rand(1000,1)-r; % Original signal  
y = quantize(q,u); % Quantized signal  
e = y - u; % Error  
m_est = mean(e) % Estimate of the error mean  
  
m_est =  
  
-1.519507450175317e-005
```

See Also `errpdf`, `errvar`, `quantize`

errorbar

Purpose Plot error bars along curve

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

Purpose Probability density function of quantization error

Syntax `[f,x] = errpdf(q)`
`f = errpdf(q,x)`

Description `[f,x] = errpdf(q)` returns the probability density function `f` evaluated at the values in `x`. The vector `x` contains the uniformly distributed random quantization errors that arise from quantizing a signal by quantizer object `q`.

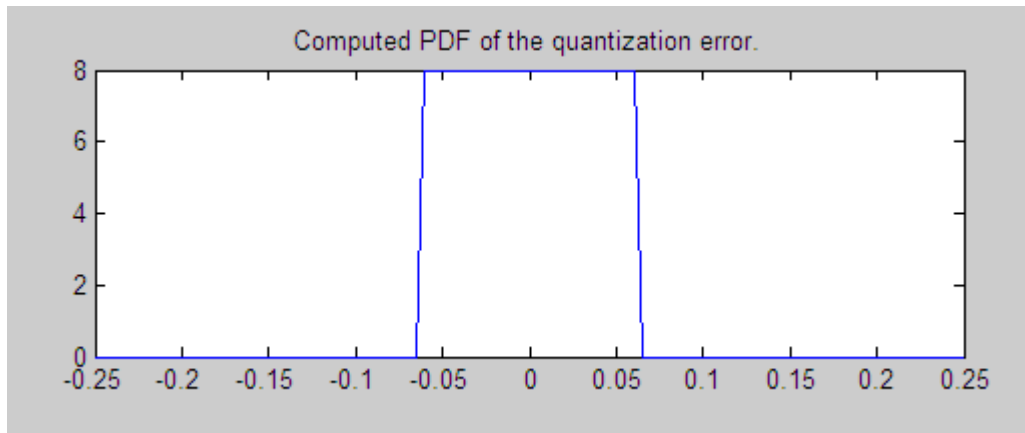
`f = errpdf(q,x)` returns the probability density function `f` evaluated at the values in vector `x`.

Note The results are not exact when the signal precision is close to the precision of the quantizer.

Examples

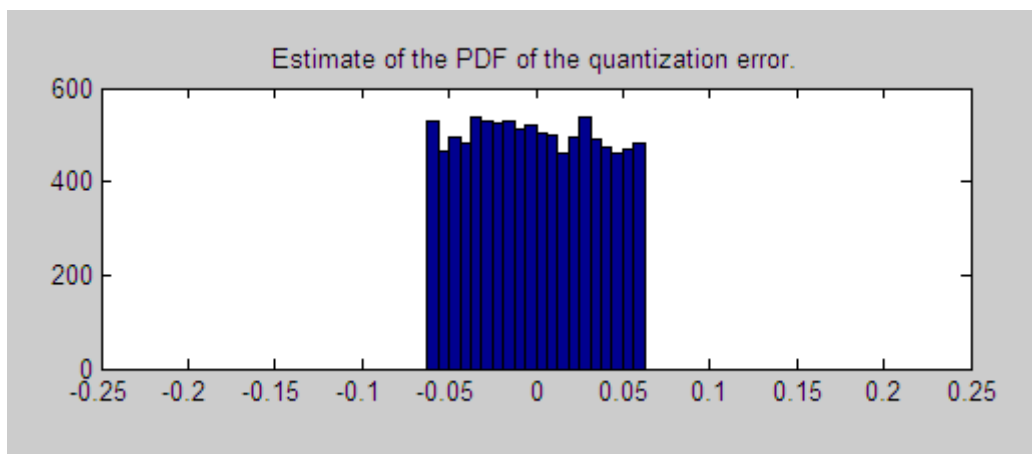
```
q = quantizer('nearest',[4 3]);  
[f,x] = errpdf(q);  
subplot(211)  
plot(x,f)  
title('Computed PDF of the quantization error.')
```

The output plot shows the probability density function of the quantization error.



Compare this result to a plot of the sample probability density function from a Monte Carlo experiment:

```
r = realmax(q);  
u = 2*r*rand(10000,1)-r; % Original signal  
y = quantize(q,u);      % Quantized signal  
e = y - u;              % Error  
subplot(212)  
hist(e,20);set(gca,'xlim',[min(x) max(x)])  
title('Estimate of the PDF of the quantization error.')
```

See Also `errmean`, `errvar`, `quantize`

errvar

Purpose Variance of quantization error

Syntax `v = errvar(q)`

Description `v = errvar(q)` returns the variance of a uniformly distributed random quantization error that arises from quantizing a signal by quantizer object `q`.

Note The results are not exact when the signal precision is close to the precision of the quantizer.

Examples

Find `v`, the variance of the quantization error for quantizer object `q`:

```
q = quantizer;  
v = errvar(q)  
  
v =  
  
7.761021455128987e-011
```

Now compare `v` to `v_est`, the sample variance from a Monte Carlo experiment:

```
r = realmax(q);  
u = 2*r*rand(1000,1)-r; % Original signal  
y = quantize(q,u); % Quantized signal  
e = y - u; % Error  
v_est = var(e) % Estimate of the error variance  
  
v_est =  
  
7.520208858166330e-011
```

See Also `errmean`, `errpdf`, `quantize`

Purpose Plot elimination tree

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

exponentbias

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`

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`

exponentmax

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');
emax = exponentmax(q)

emax =

    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`

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`

ezcontour

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.

ezmesh

Purpose Easy-to-use 3-D mesh plotter

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

Purpose Easy-to-use function plotter

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

ezplot3

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.

ezsurf

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

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

Purpose Easy-to-use combination surface/contour plotter

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

feather

Purpose Plot velocity vectors

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

Purpose

Construct fixed-point numeric 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`, Signed property value `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`, Signed property value `s`, word length `w`, and best-precision fraction length.
- `a = fi(v,s,w,f)` returns a fixed-point object with value `v`, Signed property value `s`, word length `w`, and fraction length `f`.

- `a = fi(v,s,w,slope,bias)` returns a fixed-point object with value `v`, Signed property value `s`, word length `w`, slope, and bias.
- `a = fi(v,s,w,slopeadjustmentfactor,fixedexponent,bias)` returns a fixed-point object with value `v`, Signed property value `s`, word length `w`, `slopeadjustmentfactor`, `fixedexponent`, and bias.
- `a = fi(v,T)` returns a fixed-point object with value `v` and `embedded.numericitytype T`. Refer to “Working with numericitytype Objects” for more information on `numericitytype` 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 `numericitytype` 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.numericitytype T`, and `embedded.fimath F`. The syntax `a = fi(v,T,F)` is equivalent to `a = fi(v,F,T)`.
- `a = fi(v,s,F)` returns a fixed-point object with value `v`, Signed property value `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`, Signed property value `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`, Signed property value `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`, Signed property value `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`, Signed property value `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 3-115
- “fimath Properties” on page 3-116
- “numerictype Properties” on page 3-117

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`, `int64`, `uint8`, `uint16`, `uint32`, and `uint64` 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 1-4.

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 data type information of a `fi` object, Simulink® signal or model parameter

The following `numericType` properties are, by transitivity, also properties of a `fi` object. The properties of the `numericType` object become read only after you create the `fi` object. 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
- **Signedness** — Whether a `fi` object is signed or unsigned

Note `numericType` objects can have a `Signedness` of `Auto`, but all `fi` objects must be `Signed` or `Unsigned`. If a `numericType` object with `Auto` `Signedness` is used to create a `fi` object, the `Signedness` property of the `fi` object automatically defaults to `Signed`.

- **Slope** — Slope associated with a `fi` object
- **WordLength** — Word length of the stored integer value of a `fi` object in bits

For further details on these properties, see “`numericType` Object Properties” on page 1-15.

Examples

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

For examples of casting, see “`Casting fi Objects`”.

Example 1

For example, the following creates a signed `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
        Signedness: Signed
        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
        Signedness: Signed
        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
        Signedness: Signed
        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  
      Signedness: Signed  
      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  
      Signedness: Signed  
      WordLength: 16  
      FractionLength: 13
```

See Also

`fimath`, `fipref`, `numericType`, `quantizer`

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.
You can set the default `fimath` object to be a user-configured `fimath` object or the MATLAB factory default. To learn how to set the default `fimath` object, see “Configuring the MATLAB 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 1-4 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. If you are using the MATLAB factory default as your default `fimath` object, you get the following output:

```
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`, `resetdefaultfimath`,
`savedefaultfimathpref`, `setdefaultfimath`

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 1-12.

- 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 Round toward zero

Syntax `y = fix(a)`

Description `y = fix(a)` rounds `fi` object `a` to the nearest integer in the direction of zero and returns the result in `fi` object `y`.

`y` and `a` have the same `fimath` object and `DataType` property.

When the `DataType` property of `a` is `single`, `double`, or `boolean`, the `numericType` of `y` is the same as that of `a`.

When the fraction length of `a` is zero or negative, `a` is already an integer, and the `numericType` of `y` is the same as that of `a`.

When the fraction length of `a` is positive, the fraction length of `y` is 0, its sign is the same as that of `a`, and its word length is the difference between the word length and the fraction length of `a`. If `a` is signed, then the minimum word length of `y` is 2. If `a` is unsigned, then the minimum word length of `y` is 1.

For complex `fi` objects, the imaginary and real parts are rounded independently.

`fix` does not support `fi` objects with nontrivial slope and bias scaling. Slope and bias scaling is trivial when the slope is an integer power of 2 and the bias is 0.

Examples

Example 1

The following example demonstrates how the `fix` function affects the `numericType` properties of a signed `fi` object with a word length of 8 and a fraction length of 3.

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

```
a =
```

```
3.1250
```

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

```
y = fix(a)
```

```
y =
```

```
3
```

```
DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 5
FractionLength: 0
```

Example 2

The following example demonstrates how the `fix` function affects the numeric type properties of a signed `fi` object with a word length of 8 and a fraction length of 12.

```
a = fi(0.025,1,8,12)
```

```
a =
```

```
0.0249
```

```
DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 8
FractionLength: 12
```

```
y = fix(a)
```

```
y =
```

0

DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 2
FractionLength: 0

Example 3

The functions `ceil`, `fix`, and `floor` differ in the way they round `fi` objects:

- The `ceil` function rounds values to the nearest integer toward positive infinity
- The `fix` function rounds values toward zero
- The `floor` function rounds values to the nearest integer toward negative infinity

The following table illustrates these differences for a given `fi` object `a`.

a	ceil(a)	fix(a)	floor(a)
- 2.5	-2	-2	-3
-1.75	-1	-1	-2
-1.25	-1	-1	-2
-0.5	0	0	-1
0.5	1	0	0
1.25	2	1	1
1.75	2	1	1
2.5	3	2	2

See Also

`ceil`, `convergent`, `floor`, `nearest`, `round`

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.

floor

Purpose Round toward negative infinity

Syntax `y = floor(a)`

Description `y = floor(a)` rounds `fi` object `a` to the nearest integer in the direction of negative infinity and returns the result in `fi` object `y`.

`y` and `a` have the same `fimath` object and `DataType` property.

When the `DataType` property of `a` is `single`, `double`, or `boolean`, the `numericType` of `y` is the same as that of `a`.

When the fraction length of `a` is zero or negative, `a` is already an integer, and the `numericType` of `y` is the same as that of `a`.

When the fraction length of `a` is positive, the fraction length of `y` is 0, its sign is the same as that of `a`, and its word length is the difference between the word length and the fraction length of `a`. If `a` is signed, then the minimum word length of `y` is 2. If `a` is unsigned, then the minimum word length of `y` is 1.

For complex `fi` objects, the imaginary and real parts are rounded independently.

`floor` does not support `fi` objects with nontrivial slope and bias scaling. Slope and bias scaling is trivial when the slope is an integer power of 2 and the bias is 0.

Examples

Example 1

The following example demonstrates how the `floor` function affects the `numericType` properties of a signed `fi` object with a word length of 8 and a fraction length of 3.

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

```
a =
```

```
3.1250
```

```

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

```
y = floor(a)
```

```
y =
```

```
    3
```

```

        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Signed
        WordLength: 5
        FractionLength: 0
    
```

Example 2

The following example demonstrates how the floor function affects the numeric type properties of a signed `fi` object with a word length of 8 and a fraction length of 12.

```
a = fi(0.025,1,8,12)
```

```
a =
```

```
    0.0249
```

```

        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Signed
        WordLength: 8
        FractionLength: 12
    
```

```
y = floor(a)
```

```
y =
```

```
    0
```

floor

DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 2
FractionLength: 0

Example 3

The functions `ceil`, `fix`, and `floor` differ in the way they round `fi` objects:

- The `ceil` function rounds values to the nearest integer toward positive infinity
- The `fix` function rounds values toward zero
- The `floor` function rounds values to the nearest integer toward negative infinity

The following table illustrates these differences for a given `fi` object `a`.

a	ceil(a)	fix(a)	floor(a)
- 2.5	-2	-2	-3
-1.75	-1	-1	-2
-1.25	-1	-1	-2
-0.5	0	0	-1
0.5	1	0	0
1.25	2	1	1
1.75	2	1	1
2.5	3	2	2

See Also

`ceil`, `convergent`, `fix`, `nearest`, `round`

Purpose Plot function between specified limits

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

fractionlength

Purpose	Fraction length of quantizer object
Syntax	<code>fractionlength(q)</code>
Description	<code>fractionlength(q)</code> returns the fraction length of quantizer object <code>q</code> .
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	<code>fi</code> , <code>numerictype</code> , <code>quantizer</code> , <code>wordlength</code>

Purpose	Determine whether real-world value of one <code>fi</code> object is greater than or equal to another
Syntax	<code>c = ge(a,b)</code> <code>a >= b</code>
Description	<code>c = ge(a,b)</code> is called for the syntax <code>a >= b</code> when <code>a</code> or <code>b</code> is a <code>fi</code> object. <code>a</code> and <code>b</code> must have the same dimensions unless one is a scalar. A scalar can be compared with another object of any size. <code>a >= b</code> does an element-by-element comparison between <code>a</code> and <code>b</code> 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	<code>eq</code> , <code>gt</code> , <code>le</code> , <code>lt</code> , <code>ne</code>

get

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`

Purpose	Least significant bit
Syntax	<code>c = getlsb(a)</code>
Description	<code>c = getlsb(a)</code> returns the value of the least significant bit in <code>a</code> as a <code>u1,0</code> . <code>a</code> can be a scalar <code>fi</code> object or a vector <code>fi</code> object. <code>getlsb</code> only supports <code>fi</code> objects with fixed-point data types.
See Also	<code>bitand</code> , <code>bitandreduce</code> , <code>bitconcat</code> , <code>bitget</code> , <code>bitor</code> , <code>bitorreduce</code> , <code>bitset</code> , <code>bitxor</code> , <code>bitxorreduce</code> , <code>getmsb</code>

getmsb

Purpose Most significant bit

Syntax `c = getmsb(a)`

Description `c = getmsb(a)` returns the value of the most significant bit in `a` as a `u1,0`.

`a` can be a scalar `fi` object or a vector `fi` object.

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

See Also `bitand`, `bitandreduce`, `bitconcat`, `bitget`, `bitor`, `bitorreduce`, `bitset`, `bitxor`, `bitxorreduce`, `getlsb`

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`

Purpose Hankel matrix

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

hex

Purpose Hexadecimal representation of stored integer of `fi` object

Syntax `hex(a)`

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

Fixed-point numbers can be represented as

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

or, equivalently as

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

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

Examples The following code

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

returns

```
y =
    80    7f

z =
    80    7f
```

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

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`

hist

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.

horzcat

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 `fi`math 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`

Purpose Imaginary part of complex number

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

innerprodintbits

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 $\log_2(n)$ 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 $a * \text{sign}(a')$, 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:

```
n = ceil(log2(sum(abs(a)))) + number of integer bits in b + 1 sign bit
```

The extra sign bit is only added if both a and b are signed and b attains its minimum. This prevents overflow in the event of $(-1)*(-1)$.

int

Purpose Smallest built-in integer fitting stored integer value of `fi` object

Syntax `c = int(a)`

Description `c = int(a)` returns the smallest built-in integer of the data type in which the stored integer value of `fi` object `a` fits. `int(a)` is equivalent to `a.int`.

Fixed-point numbers can be represented as

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

or, equivalently as

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

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

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

Word Length	Return Type for Signed <code>fi</code>	Return Type for Unsigned <code>fi</code>
Word length \leq 8 bits	<code>int8</code>	<code>uint8</code>
8 bits $<$ word length \leq 16 bits	<code>int16</code>	<code>uint16</code>
16 bits $<$ word length \leq 32 bits	<code>int32</code>	<code>uint32</code>
32 bits $<$ word length \leq 64 bits	<code>int64</code>	<code>uint64</code>
64 $<$ word length	<code>double</code>	<code>double</code>

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`.

Examples

The following code

```
a = fi([-1 1],1,8,7);  
y = int(a)  
z = a.int
```

returns

```
y =  
  
-128 127  
  
z =  
  
-128 127
```

See Also

[int8](#), [int16](#), [int32](#), [int64](#), [uint8](#), [uint16](#), [uint32](#), [uint64](#)

int8

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

Syntax `c = int8(a)`

Description Fixed-point numbers can be represented as

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

or, equivalently as

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

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

`c = 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`, `int64`, `uint8`, `uint16`, `uint32`, `uint64`

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

Syntax `c = int16(a)`

Description Fixed-point numbers can be represented as

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

or, equivalently as

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

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

`c = 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`, `int64`, `uint8`, `uint16`, `uint32`, `uint64`

int32

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

Syntax `c = int32(a)`

Description Fixed-point numbers can be represented as

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

or, equivalently as

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

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

`c = 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`, `int64`, `uint8`, `uint16`, `uint32`, `uint64`

Purpose Stored integer value of `fi` object as built-in `int64`

Syntax `c = int64(a)`

Description Fixed-point numbers can be represented as

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

or, equivalently as

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

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

`c = int64(a)` returns the stored integer value of `fi` object `a` as a built-in `int64`. If the stored integer word length is too big for an `int64`, or if the stored integer is unsigned, the returned value saturates to an `int64`.

See Also `int`, `int8`, `int16`, `int32`, `uint8`, `uint16`, `uint32`, `uint64`

intmax

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

Syntax `x = intmax(a)`

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

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

Purpose Smallest stored integer value representable by numeric type of fi object

Syntax `x = intmin(a)`

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

Examples

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

```
x =
```

```
-32768
```

```
      DataTypeMode: Fixed-point: binary point scaling  
      Signedness: Signed  
      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 input is Boolean

Syntax
`y = isboolean(a)`
`y = isboolean(T)`

Description `y = isboolean(a)` returns 1 when the `DataType` property of `fi` object `a` is `boolean`, and 0 otherwise.

`y = isboolean(T)` returns 1 when the `DataType` property of numeric-type object `T` is `boolean`, and 0 otherwise.

See Also `isdouble`, `isfixed`, `isfloat`, `isscaleddouble`, `issingle`

iscolumn

Purpose Determine whether `fi` object is column vector

Syntax `y = iscolumn(a)`

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

See Also `isrow`

- Purpose** Determine whether input is double-precision data type
- Syntax**
`y = isdouble(a)`
`y = isdouble(T)`
- Description**
`y = isdouble(a)` returns 1 when the `DataType` property of `fi` object `a` is `double`, and 0 otherwise.
`y = isdouble(T)` returns 1 when the `DataType` property of `numericType` object `T` is `double`, and 0 otherwise.
- See Also** `isboolean`, `isdoubleisfixed`, `isfloat`, `isscaleddouble`, `isscaledtype`, `issingle`

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

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

Description

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

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

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

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

See Also `eq`, `ispropequal`

isfi

- Purpose** Determine whether variable is `fi` object
- Syntax** `y = isfi(a)`
- Description** `y = isfi(a)` returns 1 if `a` is a `fi` object, and 0 otherwise.
- See Also** `fi`, `isfimath`, `isfipref`, `isnumericitype`, `isquantizer`

Purpose Determine whether variable is `fimath` object

Syntax `y = isfimath(F)`

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

See Also `fimath`, `isfi`, `isfipref`, `isnumericitype`, `isquantizer`

isfinite

Purpose Determine whether array elements are finite

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

Purpose	Determine whether input is <code>fipref</code> object
Syntax	<code>y = isfipref(P)</code>
Description	<code>y = isfipref(P)</code> returns 1 if <code>P</code> is a <code>fipref</code> object, and 0 otherwise.
See Also	<code>fipref</code> , <code>isfi</code> , <code>isfimath</code> , <code>isnumericitype</code> , <code>isquantizer</code>

isfixed

Purpose Determine whether input is fixed-point data type

Syntax

```
y = isfixed(a)
y = isfixed(T)
y = isfixed(q)
```

Description

`y = isfixed(a)` returns 1 when the `DataType` property of `fi` object `a` is `Fixed`, and 0 otherwise.

`y = isfixed(T)` returns 1 when the `DataType` property of `numericType` object `T` is `Fixed`, and 0 otherwise.

`y = isfixed(q)` returns 1 when `q` is a fixed-point quantizer, and 0 otherwise.

See Also `isboolean`, `isdouble`, `isfloat`, `isscaleddouble`, `isscaledtype`, `issingle`

Purpose Determine whether input is floating-point data type

Syntax

```
y = isfloat(a)
y = isfloat(T)
y = isfloat(q)
```

Description

`y = isfloat(a)` returns 1 when the `DataType` property of `fi` object `a` is `single` or `double`, and 0 otherwise.

`y = isfloat(T)` returns 1 when the `DataType` property of `numericType` object `T` is `single` or `double`, and 0 otherwise.

`y = isfloat(q)` returns 1 when `q` is a floating-point quantizer, and 0 otherwise.

See Also `isboolean`, `isdouble`, `isfixed`, `isscaleddouble`, `isscaledtype`, `issingle`

isinf

Purpose Determine whether array elements are infinite

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

Purpose Determine whether array elements are NaN

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

isnumeric

Purpose Determine whether input is numeric array

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

Purpose Determine whether input is numerictype object

Syntax `y = isnumerictype(T)`

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

See Also `isfi`, `isfimath`, `isfipref`, `isquantizer`, `numerictype`

isobject

Purpose Determine whether input is MATLAB object

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

Purpose	Determine whether properties of two <code>fi</code> objects are equal
Syntax	<code>y = ispropequal(a,b,...)</code>
Description	<code>y = ispropequal(a,b,...)</code> returns 1 if all the inputs are <code>fi</code> objects and all the inputs have the same properties. Otherwise, the function returns 0. To compare the real-world values of two <code>fi</code> objects <code>a</code> and <code>b</code> , use <code>a == b</code> or <code>isequal(a,b)</code> .
See Also	<code>fi</code> , <code>isequal</code>

isquantizer

Purpose Determine whether input is quantizer object

Syntax `y = isquantizer(q)`

Description `y = isquantizer(q)` returns 1 when `q` is a quantizer object, and 0 otherwise.

See Also `quantizer`, `isfi`, `isfimath`, `isfipref`, `isnumericitype`

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 `y = isrow(a)`

Description `y = 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.

isscaleddouble

Purpose Determine whether input is scaled double data type

Syntax `y = isscaleddouble(a)`
`y = isscaleddouble(T)`

Description `y = isscaleddouble(a)` returns 1 when the `DataType` property of fi object `a` is `ScaledDouble`, and 0 otherwise.
`y = isscaleddouble(T)` returns 1 when the `DataType` property of numeric type object `T` is `ScaledDouble`, and 0 otherwise.

See Also `isboolean`, `isdouble`, `isfixed`, `isfloat`, `isscaledtype`, `issingle`

Purpose Determine whether input is fixed-point or scaled double data type

Syntax
`y = isscaledtype(a)`
`y = isscaledtype(T)`

Description `y = isscaledtype(a)` returns 1 when the `DataType` property of `fi` object `a` is `Fixed` or `ScaledDouble`, and 0 otherwise.

`y = isscaledtype(T)` returns 1 when the `DataType` property of `numericType` object `T` is `Fixed` or `ScaledDouble`, and 0 otherwise.

See Also `isboolean`, `isdouble`, `isfixed`, `isfloat`, `numericType`, `isscaleddouble`, `issingle`

issigned

Purpose Determine whether `fi` object is signed

Syntax `y = issigned(a)`

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

Purpose	Determine whether input is single-precision data type
Syntax	<pre>y = issingle(a) y = issingle(T)</pre>
Description	<p><code>y = issingle(a)</code> returns 1 when the <code>DataType</code> property of <code>fi</code> object <code>a</code> is <code>single</code>, and 0 otherwise.</p> <p><code>y = issingle(T)</code> returns 1 when the <code>DataType</code> property of <code>numericType</code> object <code>T</code> is <code>single</code>, and 0 otherwise.</p>
See Also	<code>isboolean</code> , <code>isdouble</code> , <code>isfixed</code> , <code>isfloat</code> , <code>isscaledouble</code> , <code>isscaledtype</code>

isslopebiasscaled

Purpose Determine whether `numerictype` object has nontrivial slope and bias

Syntax `y = isslopebiasscaled(T)`

Description `y = isslopebiasscaled(T)` returns 1 when `numerictype` object `T` has nontrivial slope and bias scaling, and 0 otherwise. Slope and bias scaling is trivial when the slope is an integer power of 2, and the bias is 0.

See Also `isboolean`, `isdouble`, `isfixed`, `isfloat`, `isscaleddouble`, `isscaledtype`, `issingle`, `numerictype`

Purpose Determine whether input is vector

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

le

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.

line

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 Quantization report

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

Description `logreport(a)` displays the `minlog`, `maxlog`, `lowerbound`, `upperbound`, `noverflows`, and `nunderflows` for the `fi` object `a`.
`logreport(a, b, ...)` displays the report for each `fi` object `a`, `b`,
...

Examples The following example produces a `logreport` for `fi` objects `a` and `b`:

```
fipref('LoggingMode','On');  
a = fi(pi);  
b = fi(randn(10),1,8,7);
```

```
Warning: 27 overflows occurred in the fi assignment operation.  
Warning: 1 underflow occurred in the fi assignment operation.
```

```
logreport(a,b)  
      minlog      maxlog  lowerbound  upperbound  noverflows  nunderflows  
a   3.141602   3.141602        -4     3.999878         0         0  
b        -1   0.9921875        -1     0.9921875        27         1
```

See Also `fipref`, `quantize`, `quantizer`

lowerbound

Purpose	Lower bound of range of fi object
Syntax	<code>lowerbound(a)</code>
Description	<code>lowerbound(a)</code> returns the lower bound of the range of fi object a. If $L = \text{lowerbound}(a)$ and $U = \text{upperbound}(a)$, then $[L, U] = \text{range}(a)$.
See Also	<code>eps</code> , <code>intmax</code> , <code>intmin</code> , <code>lsb</code> , <code>range</code> , <code>realmax</code> , <code>realmin</code> , <code>upperbound</code>

Purpose	Scaling of least significant bit of <code>fi</code> object, or value of least significant bit of quantizer object
Syntax	<code>b = lsb(a)</code> <code>p = lsb(q)</code>
Description	<code>b = lsb(a)</code> returns the scaling of the least significant bit of <code>fi</code> object <code>a</code> . The result is equivalent to the result given by the <code>eps</code> function. <code>p = lsb(q)</code> returns the quantization level of quantizer object <code>q</code> , or the distance from 1.0 to the next largest floating-point number if <code>q</code> is a floating-point quantizer object.
Examples	This example uses the <code>lsb</code> function to find the value of the least significant bit of the quantizer object <code>q</code> . <pre>q = quantizer('fixed',[8 7]); p = lsb(q) p = 0.0078</pre>
See Also	<code>eps</code> , <code>intmax</code> , <code>intmin</code> , <code>lowerbound</code> , <code>quantize</code> , <code>range</code> , <code>realmax</code> , <code>realmin</code> , <code>upperbound</code>

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`, `sort`

maxlog

Purpose Log maximums

Syntax `y = maxlog(a)`
`y = maxlog(q)`

Description `y = 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` object `LoggingMode` property to `on`. Reset logging for a `fi` object using the `resetlog` function.

`y = maxlog(q)` is the maximum value after quantization during a call to `quantize(q, ...)` for quantizer object `q`. This value is the maximum value encountered over successive calls to `quantize` since logging was turned on, and is reset with `resetlog(q)`. `maxlog(q)` is equivalent to `get(q, 'maxlog')` and `q.maxlog`.

Examples **Example 1: Using maxlog with fi objects**

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

ans =

    0.999969482421875
```

The largest value `maxlog` can return is the maximum representable value of its input. In this example, `a` is a signed `fi` object with word length 16, fraction length 15 and range:

$$-1 \leq x \leq 1 - 2^{-15}$$

You can obtain the numerical range of any `fi` object `a` using the `range` function:

```
format long g
r = range(a)

r =

-1          0.999969482421875
```

Example 2: Using maxlog with quantizer objects

```
q = quantizer;
warning on
format long g
x = [-20:10];
y = quantize(q,x);
maxlog(q)

Warning: 29 overflows.
> In embedded.quantizer.quantize at 74

ans =

.999969482421875
```

The largest value `maxlog` can return is the maximum representable value of its input. You can obtain the range of `x` after quantization using the `range` function:

```
format long g
r = range(q)

r =

-1          0.999969482421875
```

See Also

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

mesh

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.

meshz

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`, `sort`

minlog

Purpose Log minimums

Syntax `y = minlog(a)`
`y = minlog(q)`

Description `y = 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` object `LoggingMode` property to `on`. Reset logging for a `fi` object using the `resetlog` function.

`y = minlog(q)` is the minimum value after quantization during a call to `quantize(q, ...)` for quantizer object `q`. This value is the minimum value encountered over successive calls to `quantize` since logging was turned on, and is reset with `resetlog(q)`. `minlog(q)` is equivalent to `get(q, 'minlog')` and `q.minlog`.

Examples **Example 1: Using minlog with fi objects**

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

```
ans =
```

```
-1
```

The smallest value `minlog` can return is the minimum representable value of its input. In this example, `a` is a signed `fi` object with word length 16, fraction length 15 and range:

$$-1 \leq x \leq 1 - 2^{-15}$$

You can obtain the numerical range of any `fi` object `a` using the `range` function:

```
format long g
r = range(a)

r =

           -1           0.999969482421875
```

Example 2: Using minlog with quantizer objects

```
q = quantizer;
warning on
x = [-20:10];
y = quantize(q,x);
minlog(q)

Warning: 29 overflows.
> In embedded.quantizer.quantize at 74

ans =

           -1
```

The smallest value minlog can return is the minimum representable value of its input. You can obtain the range of x after quantization using the range function:

```
format long g
r = range(q)

r =

           -1           0.999969482421875
```

See Also

fipref, maxlog, noverflows, nunderflows, reset, resetlog

minus

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 value (a 1-by-1 matrix). A scalar value can be subtracted from any other value.

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

Note For information about the `fimath` properties involved in Fixed-Point Toolbox calculations, see “Using `fimath` Objects to Perform Fixed-Point Arithmetic” and “Using `fimath` ProductMode and SumMode” in the *Fixed-Point Toolbox User’s Guide*.

For information about calculations using Simulink® Fixed Point™ software, see the “Arithmetic Operations” chapter of the *Simulink Fixed Point User’s Guide*.

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
Signedness: Signed
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`, `mrdivide`, `numerictype`, `rdivide`, `sub`, `sum`

Purpose Forward slash (/) or right-matrix division

Syntax
`c = mrdivide(a,b)`
`c = a/b`

Description `c = mrdivide(a,b)` and `c = a/b` perform right-matrix division.

When one or both of the inputs is a `fi` object, the denominator input, `b`, must be a scalar and the output `fi` object `c` is equivalent to `c = rdivide(a,b)` or `c = a./b` (right-array division).

The numerator input `a` can be complex, but the denominator input `b` must always be real-valued. When the numerator input `a` is complex, the real and imaginary parts of `a` are independently divided by `b`.

For information on the data type rules used by the `mrdivide` function, see the `rdivide` reference page.

Examples In this example, you use the forward slash (/) to perform right matrix division on a 3-by-3 magic square of `fi` objects. Because the numerator input is a `fi` object, the denominator input `b` must be a scalar:

```
a = fi(magic(3))
b = fi(3, 1, 12, 8)
c = a/b
```

The `mrdivide` function outputs a signed 3-by-3 array of `fi` objects, each of which has a word length of 16 bits and a fraction length of 3 bits.

```
a =
```

```

8     1     6
3     5     7
4     9     2
```

```

      DataTypeMode: Fixed-point: binary point scaling
      Signedness: Signed
      WordLength: 16
      FractionLength: 11
```

mrdivide

b =

3

DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 12
FractionLength: 8

c =

2.6250	0.3750	2.0000
1.0000	1.6250	2.3750
1.3750	3.0000	0.6250

DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 16
FractionLength: 3

See Also

add, divide, fi, fimath, numerictype, rdivide, 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`. A scalar value (a 1-by-1 matrix) can multiply any other value. 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.

Note For information about the `fimath` properties involved in Fixed-Point Toolbox calculations, see “Using `fimath` Objects to Perform Fixed-Point Arithmetic” and “Using `fimath` ProductMode and SumMode” in the *Fixed-Point Toolbox User’s Guide*.

For information about calculations using Simulink Fixed Point software, see the “Arithmetic Operations” chapter of the *Simulink Fixed Point User’s Guide*.

See Also `plus`, `minus`, `times`, `uminus`

ndgrid

Purpose Generate arrays for N-D functions and interpolation

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

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	Round toward nearest integer with ties rounding toward positive infinity
Syntax	<code>y = nearest(a)</code>
Description	<p><code>y = nearest(a)</code> rounds <code>fi</code> object <code>a</code> to the nearest integer or, in case of a tie, to the nearest integer in the direction of positive infinity, and returns the result in <code>fi</code> object <code>y</code>.</p> <p><code>y</code> and <code>a</code> have the same <code>fi</code> object and <code>DataType</code> property.</p> <p>When the <code>DataType</code> property of <code>a</code> is <code>single</code>, <code>double</code>, or <code>boolean</code>, the <code>numericType</code> of <code>y</code> is the same as that of <code>a</code>.</p> <p>When the fraction length of <code>a</code> is zero or negative, <code>a</code> is already an integer, and the <code>numericType</code> of <code>y</code> is the same as that of <code>a</code>.</p> <p>When the fraction length of <code>a</code> is positive, the fraction length of <code>y</code> is 0, its sign is the same as that of <code>a</code>, and its word length is the difference between the word length and the fraction length of <code>a</code>, plus one bit. If <code>a</code> is signed, then the minimum word length of <code>y</code> is 2. If <code>a</code> is unsigned, then the minimum word length of <code>y</code> is 1.</p> <p>For complex <code>fi</code> objects, the imaginary and real parts are rounded independently.</p> <p><code>nearest</code> does not support <code>fi</code> objects with nontrivial slope and bias scaling. Slope and bias scaling is trivial when the slope is an integer power of 2 and the bias is 0.</p>

Examples

Example 1

The following example demonstrates how the `nearest` function affects the `numericType` properties of a signed `fi` object with a word length of 8 and a fraction length of 3.

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

```
a =
```

```
3.1250
```

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

```
y = nearest(a)
```

```
y =
```

```
3
```

```
    DataTypeMode: Fixed-point: binary point scaling  
    Signedness: Signed  
    WordLength: 6  
    FractionLength: 0
```

Example 2

The following example demonstrates how the `nearest` function affects the numeric type properties of a signed `fi` object with a word length of 8 and a fraction length of 12.

```
a = fi(0.025,1,8,12)
```

```
a =
```

```
0.0249
```

```
    DataTypeMode: Fixed-point: binary point scaling  
    Signedness: Signed  
    WordLength: 8  
    FractionLength: 12
```

```
y = nearest(a)
```

```
y =
```


0

```

DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 2
FractionLength: 0
    
```

Example 3

The functions `convergent`, `nearest` and `round` differ in the way they treat values whose least significant digit is 5:

- The `convergent` function rounds ties to the nearest even integer
- The `nearest` function rounds ties to the nearest integer toward positive infinity
- The `round` function rounds ties to the nearest integer with greater absolute value

The following table illustrates these differences for a given `fi` object `a`.

a	convergent(a)	nearest(a)	round(a)
-3.5	-4	-3	-4
-2.5	-2	-2	-3
-1.5	-2	-1	-2
-0.5	0	0	-1
0.5	0	1	1
1.5	2	2	2
2.5	2	3	3
3.5	4	4	4

See Also

`ceil`, `convergent`, `fix`, `floor`, `round`

noperations

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.

noverflows

Purpose Number of overflows

Syntax `y = noverflows(a)`
 `y = noverflows(q)`

Description `y = 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.

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

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

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`

num2hex

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

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`

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`

numerictype

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, Signed property value `s`, and 16-bit word length.
- `T = numerictype(s,w)` creates a numerictype object with Fixed-point: unspecified scaling, Signed property value `s`, and word length `w`.
- `T = numerictype(s,w,f)` creates a numerictype object with Fixed-point: binary point scaling, Signed property value `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, Signed property value `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, Signed property value `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 1-15.

- `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
- `Signedness` — Signed, unsigned, or auto
- `Slope` — Slope
- `WordLength` — Word length of the stored integer value, in bits

Examples

Example 1

Type

```
T = numerictype
```

numerictype

to create a default numerictype object.

```
T =
    DataType: Fixed
    Scaling: BinaryPoint
    Signedness: Signed
    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
    Signedness: Signed
    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
    Signedness: Signed
    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
        Signedness: Signed
        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
        Signedness: Signed
        WordLength: 32
        Slope: 0.25
        Bias: 4
```

Example 6

You can create a `numerictype` object with an unspecified sign by using property name/property values pairs to set the `Signedness` property to `Auto`.

```
T = numerictype('Signedness', 'Auto')
```

numerictype

T =

```
DataTypeMode: Fixed-point: binary point scaling
Signedness: Auto
WordLength: 16
FractionLength: 15
```

Note Although you can create `numerictype` objects with an unspecified sign (`Signedness: Auto`), all `fi` objects must have a `Signedness` of `Signed` or `Unsigned`. If you use a `numerictype` object with `Signedness: Auto` to construct a `fi` object, the `Signedness` property of the `fi` object automatically defaults to `Signed`.

See Also

`fi`, `fimath`, `fipref`, `quantizer`

Purpose Number of underflows

Syntax `y = nunderflows(a)`
 `y = nunderflows(q)`

Description `y = 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.

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

See Also `maxlog`, `minlog`, `noverflows`, `resetlog`

oct

Purpose Octal representation of stored integer of fi object

Syntax `oct(a)`

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

Fixed-point numbers can be represented as

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

or, equivalently as

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

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

Examples The following code

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

returns

```
y =
    200    177

z =
    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.

patch

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 value (a 1-by-1 matrix). A scalar value can be added to any other value.

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

Note For information about the `fimath` properties involved in Fixed-Point Toolbox calculations, see “Using `fimath` Objects to Perform Fixed-Point Arithmetic” and “Using `fimath` ProductMode and SumMode” in the *Fixed-Point Toolbox User’s Guide*.

For information about calculations using Simulink Fixed Point software, see the “Arithmetic Operations” chapter of the *Simulink Fixed Point User’s Guide*.

See Also `minus`, `mtimes`, `times`, `uminus`

polar

Purpose Plot polar coordinates

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

Purpose	Efficient fixed-point multiplication by 2^K
Syntax	<code>b = pow2(a,K)</code>
Description	<code>b = pow2(a,K)</code> returns the value of <code>a</code> shifted by <code>K</code> bits where <code>K</code> is an integer and <code>a</code> and <code>b</code> are <code>fi</code> objects. The output <code>b</code> always has the same word length and fraction length as the input <code>a</code> .

Note In fixed-point arithmetic, shifting by `K` bits is equivalent to, and more efficient than, computing $b = a \cdot 2^k$.

If `K` is a non-integer, the `pow2` function will round it to `floor` before performing the calculation.

The scaling of `a` must be equivalent to binary point-only scaling; in other words, it must have a power of 2 slope and a bias of 0.

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

The `pow2` function obeys the `OverflowMode` and `RoundMode` properties of `a`. If obeying the `RoundMode` property of `a` is not important, try using the `bitshift` function.

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

The function also does not support the syntax `b = pow2(a)` when `a` is a `fi` object.

Examples

Example 1

In the following example, `a` is a real-valued `fi` object, and `K` is a positive integer.

The `pow2` function shifts the bits of `a` 3 places to the left, effectively multiplying `a` by 2^3 .

```
a = fi(pi,1,16,8)
```

```
b = pow2(a,3)
binary_a = bin(a)
binary_b = bin(b)
```

MATLAB returns:

a =

3.1406

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

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

b =

25.1250

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

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

```
MaxSumWordLength: 128
CastBeforeSum: true
```

```
binary_a =
0000001100100100
```

```
binary_b =
0001100100100000
```

Example 2

In the following example, `a` is a real-valued `fi` object, and `K` is a negative integer.

The `pow2` function shifts the bits of `a` 4 places to the right, effectively multiplying `a` by 2^{-4} .

```
a = fi(pi,1,16,8)
b = pow2(a,-4)
binary_a = bin(a)
binary_b = bin(b)
```

MATLAB returns:

```
a =
3.1406
    DataTypeMode: Fixed-point: binary point scaling
    Signedness: Signed
    WordLength: 16
    FractionLength: 8
    RoundMode: nearest
    OverflowMode: saturate
    ProductMode: FullPrecision
```

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

b =

0.1953

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

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

binary_a =

0000001100100100

binary_b =

0000000000110010

Example 3

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 =
    s16,8          57 -          2i
pow2(a, 2)
ans =
    s16,8          127.99609375 -          8i
```

See Also bitshift, bitsll, bitsra, bitsrl

quantize

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. Quantize does not change nonnumeric elements or fields of `x`, nor does it issue warnings for nonnumeric values. The output `y` is a built-in double. When the input `x` is a structure or cell array, the fields of `y` are built-in doubles.

`[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');
```

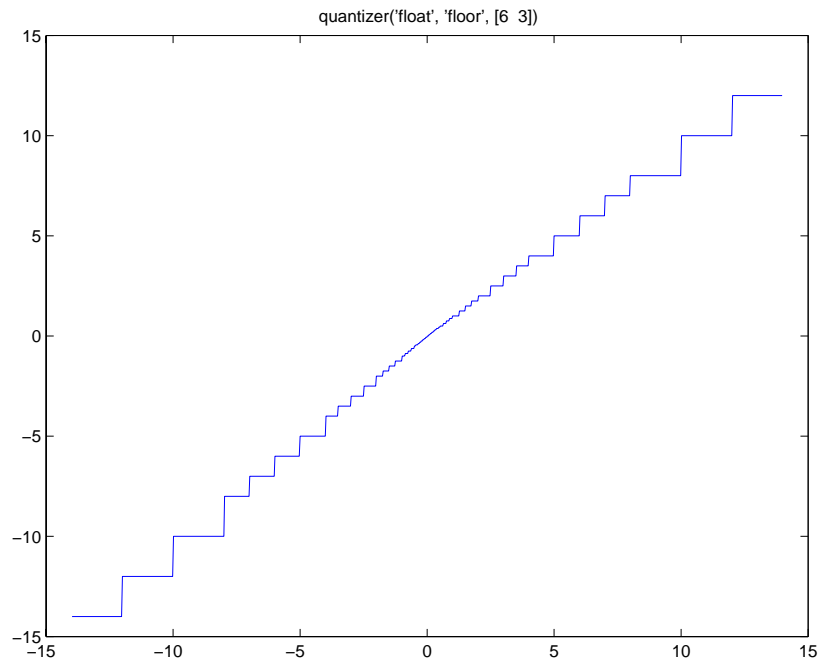


```
range(q)

ans =

    -14    14
y=quantize(q,u);
plot(u,y);title(tostring(q))
```

Warning: 68 overflows.



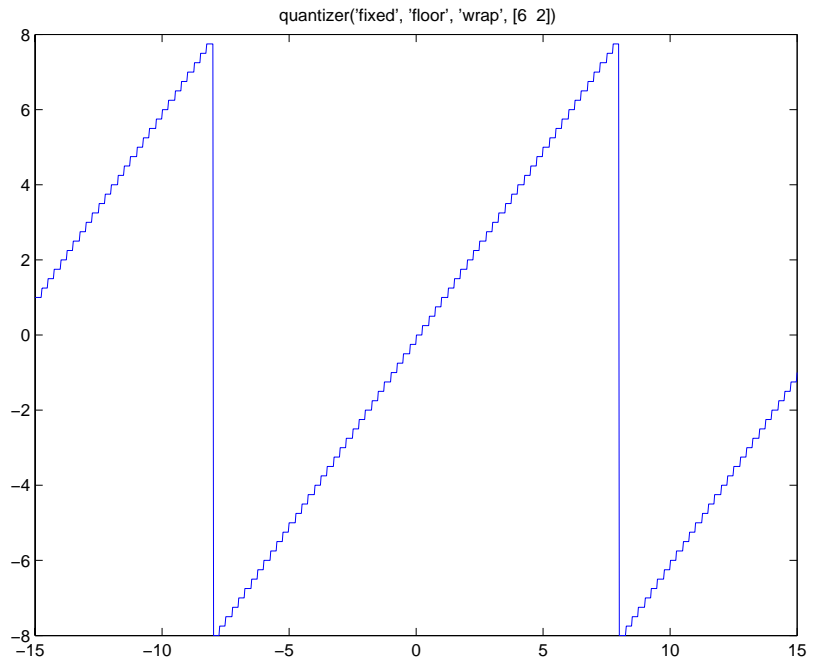
Example 2 - Fixed-Point

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

quantize

```
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

assignmentquantizer, quantizer, set, unitquantize, unitquantizer

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 1-20.

quantizer

Property Name	Property Value	Description
mode	'double'	Double-precision mode. Override all other parameters.
	'float'	Custom-precision floating-point mode.
	'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'	Round to nearest integer with ties rounding to nearest even integer.
	'fix'	Round toward zero.
	'floor'	Round toward negative infinity.
	'nearest'	Round to nearest integer with ties rounding toward positive infinity.
	'round'	Round to nearest integer with ties rounding to nearest integer with greater absolute value.

Property Name	Property Value	Description
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

`assignmentquantizer`, `fi`, `fimath`, `fipref`, `numericity`, `quantize`, `set`, `unitquantize`, `unitquantizer`

quiver

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.

randquant

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 , $y = \text{quantize}(q, x)$ returns y in the range $a \leq y \leq b$.

[*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, $a = -\text{realmax}(q)$, $b = \text{realmax}(q)$.

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

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

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

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

$$a = 0$$

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

See `realmax` for more information.

See Also

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

rdivide

Purpose Right-array division (./)

Syntax
`c = rdivide(a,b)`
`c = a./b`

Description `c = rdivide(a,b)` and `c = a./b` perform right-array division by dividing each element of `a` by the corresponding element of `b`. If inputs `a` and `b` are not the same size, one of them must be a scalar value.

The numerator input `a` can be complex, but the denominator `b` requires a real-valued input. If `a` is complex, the real and imaginary parts of `a` are independently divided by `b`.

The following table shows the rules used to assign property values to the output of the `rdivide` function.

Output Property	Rule
Signedness	If either input is Signed, the output is Signed. If both inputs are Unsigned, the output is Unsigned.
WordLength	The output word length equals the maximum of the input word lengths.
FractionLength	For <code>c = a./b</code> , the fraction length of output <code>c</code> equals the fraction length of <code>a</code> minus the fraction length of <code>b</code> .

The following table shows the rules the `rdivide` function uses to handle inputs with different data types.

Case	Rule
Interoperation of <code>fi</code> objects and built-in integers	Built-in integers are treated as fixed-point objects. For example, <code>B = int8(2)</code> is treated as an <code>s8,0 fi</code> object.

Case	Rule
Interoperation of <code>fi</code> objects and constants	The Embedded MATLAB™ subset treats constant integers as fixed-point objects with the same word length as the <code>fi</code> object and a fraction length of 0.
Interoperation of mixed data types	<p>Similar to all other <code>fi</code> object functions, when inputs <code>a</code> and <code>b</code> have different data types, the data type with the higher precedence determines the output data type. The order of precedence is as follows:</p> <ol style="list-style-type: none"> 1 ScaledDouble 2 Fixed-point 3 Built-in double 4 Built-in single <p>When both inputs are <code>fi</code> objects, the only data types that are allowed to mix are <code>ScaledDouble</code> and <code>Fixed-point</code>.</p>

Examples

In this example, you perform right-array division on a 3-by-3 magic square of `fi` objects. Each element of the 3-by-3 magic square is divided by the corresponding element in the 3-by-3 input array `b`.

```
a = fi(magic(3))
b = int8([3 3 4; 1 2 4 ; 3 1 2 ])
c = a./b
```

The `mrdivide` function outputs a 3-by-3 array of signed `fi` objects, each of which has a word length of 16 bits and fraction length of 11 bits.

```
a =
```

rdivide

```
8   1   6
3   5   7
4   9   2
```

```
DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 16
FractionLength: 11
```

b =

```
3   3   4
1   2   4
3   1   2
```

c =

```
2.6665    0.3335    1.5000
3.0000    2.5000    1.7500
1.3335    9.0000    1.0000
```

```
DataTypeMode: Fixed-point: binary point scaling
Signedness: Signed
WordLength: 16
FractionLength: 11
```

See Also

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

Purpose Real part of complex number

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

realmax

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

realmin

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]);
x = realmin(q)

x =

    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`

Purpose Convert fixed-point data types without changing underlying data

Syntax `c = reinterprecast(a, T)`

Description `c = reinterprecast(a, T)` converts the input `fi` object `a` to the data type specified by `numericType` object `T` without changing the underlying data. The result is returned in `fi` object `c`.

The data type of the input `fi` object `a` must be fixed point, and `T` must be a `numericType` object with a fully specified fixed-point data type. The word length of inputs `a` and `T` must be the same.

The `reinterprecast` function differs from the MATLAB `typecast` and `cast` functions in that it only operates on `fi` objects and it does not allow the word length of the input to change.

Examples

In the following example, `a` is a signed `fi` object with a word length of 8 bits and a fraction length of 7 bits. The `reinterprecast` function converts `a` into an unsigned `fi` object `c` with a word length of 8 bits and a fraction length of 0 bits. The real-world values of `a` and `c` are different, but their binary representations are the same.

```
a = fi([-1 pi/4], true, 8, 7)
T = numericType(false, 8, 0);
c = reinterprecast(a, T)
a =

    -1.0000    0.7891

        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Signed
        WordLength: 8
        FractionLength: 7

c =

    128    101
```

reinterprecast

```
DataTypeMode: Fixed-point: binary point scaling
Signedness: Unsigned
WordLength: 8
FractionLength: 0
```

To verify that the underlying data has not changed, compare the binary representations of a and c:

```
binary_a = bin(a)
binary_c = bin(c)
binary_a =

10000000    01100101

binary_c =

10000000    01100101
```

See Also

`cast`, `fi`, `numerictype`, `typecast`

Purpose Replicate and tile array

Description Refer to the MATLAB repmat reference page for more information.

rescale

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](#)

reset

Purpose Reset objects to initial conditions

Syntax `reset(P)`
 `reset(q)`

Description `reset(P)` resets the `fipref` object `P` to its initial conditions.
`reset(q)` resets the following quantizer object properties to their initial conditions:

- `minlog`
- `maxlog`
- `noverflows`
- `nunderflows`
- `noperations`

See Also `resetlog`

Purpose Set default fimath object to MATLAB factory default

Syntax resetdefaultfimath

Description resetdefaultfimath sets the default fimath object to the MATLAB factory setting in your current MATLAB session. The MATLAB factory default fimath object has the following properties:

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

For more information on working with the default fimath object, see “Configuring the MATLAB Default fimath Object” in the *Fixed-Point Toolbox User’s Guide*.

Examples In this example, you create your own fimath object F and set it as the MATLAB default fimath object. Then, use the resetdefaultfimath command to reset the default fimath object to the MATLAB factory setting.

```
F = fimath('RoundMode','Floor','OverflowMode','Wrap');
setdefaultfimath(F);
F1 = fimath
a = fi(pi)
```

```
F1 =
```

```
RoundMode: floor
OverflowMode: wrap
ProductMode: FullPrecision
MaxProductWordLength: 128
```

resetdefaultfimath

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

```
a =
```

```
3.1415
```

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

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

Now, set your MATLAB default `fimath` object back to the factory setting:

```
resetdefaultfimath;
F2 = fimath
a = fi(pi)
```

```
F2 =
```

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

```
CastBeforeSum: true

a =

    3.1416

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

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

All `fi` and `fimath` objects you create (without specifying a `fimath` object in the constructor) in your current MATLAB session are now assigned the MATLAB factory default `fimath` object.

To use the current MATLAB default `fimath` object in future MATLAB sessions, you must use the `savedefaultfimathpref` command.

See Also

`fimath`, `savedefaultfimathpref`, `setdefaultfimath`

resetlog

Purpose	Clear log for fi or quantizer object
Syntax	<code>resetlog(a)</code> <code>resetlog(q)</code>
Description	<code>resetlog(a)</code> clears the log for fi object a. <code>resetlog(q)</code> clears the log for quantizer object q. Turn logging on or off by setting the <code>fipref</code> property <code>LoggingMode</code> .
See Also	<code>fipref</code> , <code>maxlog</code> , <code>minlog</code> , <code>noperations</code> , <code>noverflows</code> , <code>nunderflows</code> , <code>reset</code>

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.

rose

Purpose Create angle histogram

Description Refer to the MATLAB rose reference page for more information.

Purpose	Round <code>fi</code> object toward nearest integer or round input data using quantizer object
Syntax	<code>y = round(a)</code> <code>y = round(q, x)</code>
Description	<p><code>y = round(a)</code> rounds <code>fi</code> object <code>a</code> to the nearest integer. In the case of a tie, <code>round</code> rounds values to the nearest integer with greater absolute value. The rounded value is returned in <code>fi</code> object <code>y</code>.</p> <p><code>y</code> and <code>a</code> have the same <code>fi</code> object and <code>DataType</code> property.</p> <p>When the <code>DataType</code> of <code>a</code> is <code>single</code>, <code>double</code>, or <code>boolean</code>, the <code>numericType</code> of <code>y</code> is the same as that of <code>a</code>.</p> <p>When the fraction length of <code>a</code> is zero or negative, <code>a</code> is already an integer, and the <code>numericType</code> of <code>y</code> is the same as that of <code>a</code>.</p> <p>When the fraction length of <code>a</code> is positive, the fraction length of <code>y</code> is 0, its sign is the same as that of <code>a</code>, and its word length is the difference between the word length and the fraction length of <code>a</code>, plus one bit. If <code>a</code> is signed, then the minimum word length of <code>y</code> is 2. If <code>a</code> is unsigned, then the minimum word length of <code>y</code> is 1.</p> <p>For complex <code>fi</code> objects, the imaginary and real parts are rounded independently.</p> <p><code>round</code> does not support <code>fi</code> objects with nontrivial slope and bias scaling. Slope and bias scaling is trivial when the slope is an integer power of 2 and the bias is 0.</p> <p><code>y = round(q, x)</code> uses the <code>RoundMode</code> and <code>FractionLength</code> settings of <code>q</code> to round the numeric data <code>x</code>, but does not check for overflows during the operation. Compare to <code>quantize</code>.</p>

Examples

Example 1

The following example demonstrates how the `round` function affects the `numericType` properties of a signed `fi` object with a word length of 8 and a fraction length of 3.

round

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

```
a =
```

```
3.1250
```

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

```
y = round(a)
```

```
y =
```

```
3
```

```
      DataTypeMode: Fixed-point: binary point scaling  
      Signedness: Signed  
      WordLength: 6  
      FractionLength: 0
```

Example 2

The following example demonstrates how the round function affects the numeric type properties of a signed `fi` object with a word length of 8 and a fraction length of 12.

```
a = fi(0.025, 1, 8, 12)
```

```
a =
```

```
0.0249
```

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

WordLength: 8
 FractionLength: 12

y = round(a)

y =

0

DataTypeMode: Fixed-point: binary point scaling
 Signedness: Signed
 WordLength: 2
 FractionLength: 0

Example 3

The functions `convergent`, `nearest` and `round` differ in the way they treat values whose least significant digit is 5:

- The `convergent` function rounds ties to the nearest even integer
- The `nearest` function rounds ties to the nearest integer toward positive infinity
- The `round` function rounds ties to the nearest integer with greater absolute value

The following table illustrates these differences for a given `fi` object `a`.

a	convergent(a)	nearest(a)	round(a)
-3.5	-4	-3	-4
-2.5	-2	-2	-3
-1.5	-2	-1	-2
-0.5	0	0	-1
0.5	0	1	1
1.5	2	2	2

round

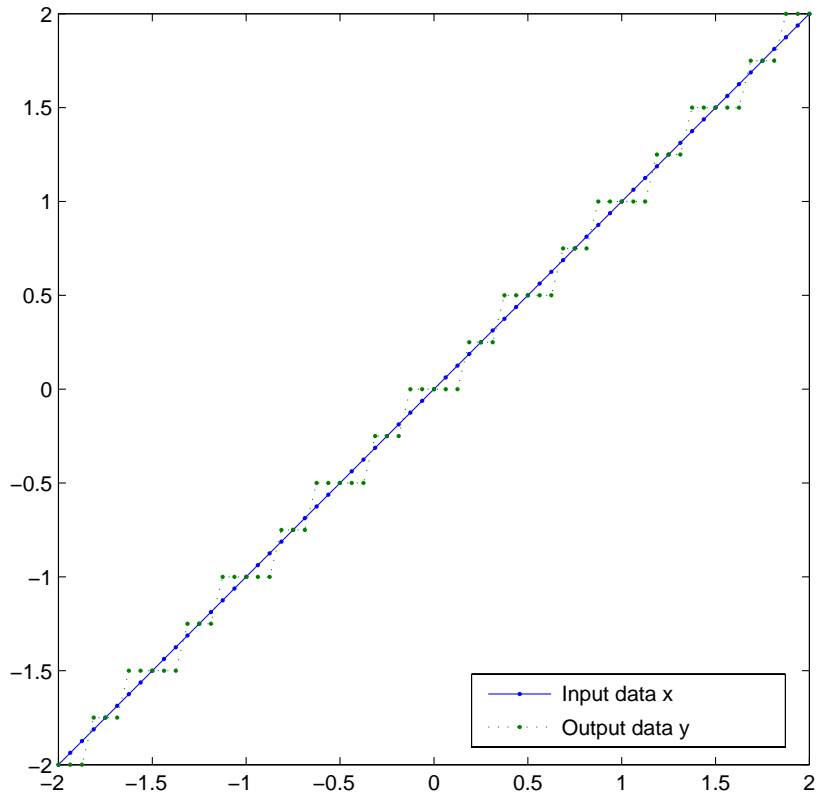
a	convergent(a)	nearest(a)	round(a)
2.5	2	3	3
3.5	4	4	4

Example 4

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 in the following figure. Linear data input results in output where `y` shows distinct quantization levels.



See Also

ceil, convergent, fix, floor, nearest, quantize, quantizer

savedefaultfimathpref

Purpose	Save default <code>fimath</code> object for next MATLAB session
Syntax	<code>savedefaultfimathpref</code>
Description	<p><code>savedefaultfimathpref</code> saves the current MATLAB default <code>fimath</code> object as the default <code>fimath</code> object to be used in all future MATLAB sessions.</p> <p>For more information on working with the default <code>fimath</code> object, see “Configuring the MATLAB Default <code>fimath</code> Object” in the <i>Fixed-Point Toolbox User’s Guide</i>.</p>
See Also	<code>fimath</code> , <code>setdefaultfimath</code> , <code>resetdefaultfimath</code>

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

scatter

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

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

or, equivalently as

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

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

`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 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`.

set

`s = set(q)` returns a structure containing the possible values for the properties of quantizer object `q`.

Note The `set` function operates on quantizer objects. To learn about setting the properties of other objects, see properties of `fi`, `fimath`, `fipref`, and `numericType` objects.

See Also

`get`

Purpose Set MATLAB default fimath object

Syntax `setdefaultfimath(F)`

Description `setdefaultfimath(F)` sets the fimath object `F` as the default fimath object for your current MATLAB session.

For more information on working with the MATLAB default fimath object, see “Configuring the MATLAB Default fimath Object” in the *Fixed-Point Toolbox User’s Guide*.

Examples If you create a `fi` or `fimath` object in the MATLAB workspace and do not specify a fimath object in the constructor, Fixed-Point Toolbox software assigns it the MATLAB default fimath object. To change the MATLAB default fimath object, you must use the `setdefaultfimath` command.

In this example, you create your own fimath object `F` and set it as the default fimath object for your current MATLAB session:

```
F = fimath('RoundMode', 'Floor', 'OverflowMode', 'Wrap')
```

```
F =
```

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

```
setdefaultfimath(F);
```

After you set `F` as your MATLAB default fimath object, any `fi` or `fimath` objects you create (without specifying a fimath object in the constructor), will be assigned the fimath object `F`.

setdefaultfimath

```
F1 = fimath
a = fi(pi)
```

```
F1 =
```

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

```
a =
```

```
3.1415
```

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

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

To use the current MATLAB default `fimath` object as your default `fimath` object in future MATLAB sessions, you must use the `savedefaultfimathpref` command.

See Also

`fimath`, `savedefaultfimathpref`, `resetdefaultfimath`

Purpose Shift data to operate on specified dimension

Syntax `[x,perm,nshifts] = shiftdata(x,dim)`

Description `[x,perm,nshifts] = shiftdata(x,dim)` shifts data `x` to permute dimension `dim` to the first column using the same permutation as the built-in `filter` function. The vector `perm` returns the permutation vector that is used.

If `dim` is missing or empty, then the first non-singleton dimension is shifted to the first column, and the number of shifts is returned in `nshifts`.

`shiftdata` is meant to be used in tandem with `unshiftdata`, which shifts the data back to its original shape. These functions are useful for creating functions that work along a certain dimension, like `filter`, `goertzel`, `sgolayfilt`, and `sosfilt`.

Examples

Example 1

This example shifts `x`, a 3-by-3 magic square, permuting dimension 2 to the first column. `unshiftdata` shifts `x` back to its original shape.

1. Create a 3-by-3 magic square:

```
x = fi(magic(3))
```

```
x =
```

```

     8     1     6
     3     5     7
     4     9     2

```

2. Shift the matrix `x` to work along the second dimension:

```
[x,perm,nshifts] = shiftdata(x,2)
```

The permutation vector, `perm`, and the number of shifts, `nshifts`, are returned along with the shifted matrix, `x`:

```
x =
```

```
    8    3    4
    1    5    9
    6    7    2
```

```
perm =
```

```
    2    1
```

```
nshifts =
```

```
    []
```

3. Shift the matrix back to its original shape:

```
y = unshiftdata(x,perm,nshifts)
```

```
y =
```

```
    8    1    6
    3    5    7
    4    9    2
```

Example 2

This example shows how `shiftdata` and `unshiftdata` work when you define `dim` as empty.

1. Define `x` as a row vector:

```
x = 1:5
```

```
x =  
    1    2    3    4    5
```

2. Define `dim` as empty to shift the first non-singleton dimension of `x` to the first column:

```
[x,perm,nshifts] = shiftdata(x,[])
```

`x` is returned as a column vector, along with `perm`, the permutation vector, and `nshifts`, the number of shifts:

```
x =  
    1  
    2  
    3  
    4  
    5  
  
perm =  
  
    []  
  
nshifts =  
  
    1
```

3. Using `unshiftdata`, restore `x` to its original shape:

```
y = unshiftdata(x,perm,nshifts)
```

shiftdata

y =

1 2 3 4 5

See Also

permute, shiftdim, unshiftdata

Purpose Shift dimensions

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

sign

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.

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

Syntax `single(a)`

Description Fixed-point numbers can be represented as

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

or, equivalently as

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

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

See Also `double`

size

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.

sort

Purpose Sort elements of real-valued fi object in ascending or descending order

Description Refer to the MATLAB sort reference page for more information.

Purpose Visualize sparsity pattern

Description Refer to the MATLAB spy reference page for more information.

sqrt

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 3-311.

`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 3-311.

`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:

$$sign_c = sign_a$$

$$WL_c = \text{ceil}\left(\frac{WL_a}{2}\right)$$

$$FL_c = WL_c - \text{ceil}\left(\frac{WL_a - FL_a}{2}\right)$$

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 staircase graph

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

stem

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.

streamribbon

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.

streamtube

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 binary-point scaling, zero fraction length 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.

```
fipref('NumericTypeDisplay','short', ...
       'FimathDisplay','none');
format long g
a = fi(0.1,true,48,47)

a =

           0.1000000000000001
           s48,47
b = stripscaling(a)

b =

           14073748835533
           s48,0
bin(a)

ans =

0000110011001100110011001100110011001100110011001100110011001101

bin(b)

ans =

0000110011001100110011001100110011001100110011001100110011001101
```

stripscaling

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
Signedness: Signed
WordLength: 32
FractionLength: 16
```

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

sub

```
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`, `mrdivide`, `numerictype`, `rdivide`

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 value.

`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 following 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

Example 1

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` while `a` assumes the value, `numerictype` object and `fi`math 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  
      Signedness: Signed  
      WordLength: 8  
      FractionLength: 7
```

```
b = fi(pi/4, 1, 16, 15)
```

```
b =
```

```
0.7854
```

```
      DataTypeMode: Fixed-point: binary point scaling  
      Signedness: Signed  
      WordLength: 16  
      FractionLength: 15
```

```
a(:) = b
```

```
a =
```

```
0.7891
```

```

        DataTypeMode: Fixed-point: binary point scaling
        Signedness: Signed
        WordLength: 8
        FractionLength: 7

```

Example 2

This example defines a variable `acc` to emulate a 40-bit accumulator of a DSP. The products and sums in this example are assigned into the accumulator using the syntax `acc(1) = ...`. Assigning values into the accumulator is like storing a value in a register.

To begin, turn the logging mode on and define the variables. In this example, `n` is the number of points in the input data `x` and output data `y`, and `t` represents time. The remaining variables are all defined as `fi` objects. The input data `x` is a high-frequency sinusoid added to a low-frequency sinusoid.

```

fipref('LoggingMode','on');
n = 100;
t = (0:n-1)/n;
x = fi(sin(2*pi*t) + 0.2*cos(2*pi*50*t));
b = fi([.5 .5]);
y = fi(zeros(size(x)), numerictype(x));
acc = fi(0.0, true, 40, 30);

```

The following loop takes a running average of the input `x` using the coefficients in `b`. Notice that `acc` is assigned into `acc(1) = ...` versus using `acc = ...`, which would overwrite and change the data type of `acc`.

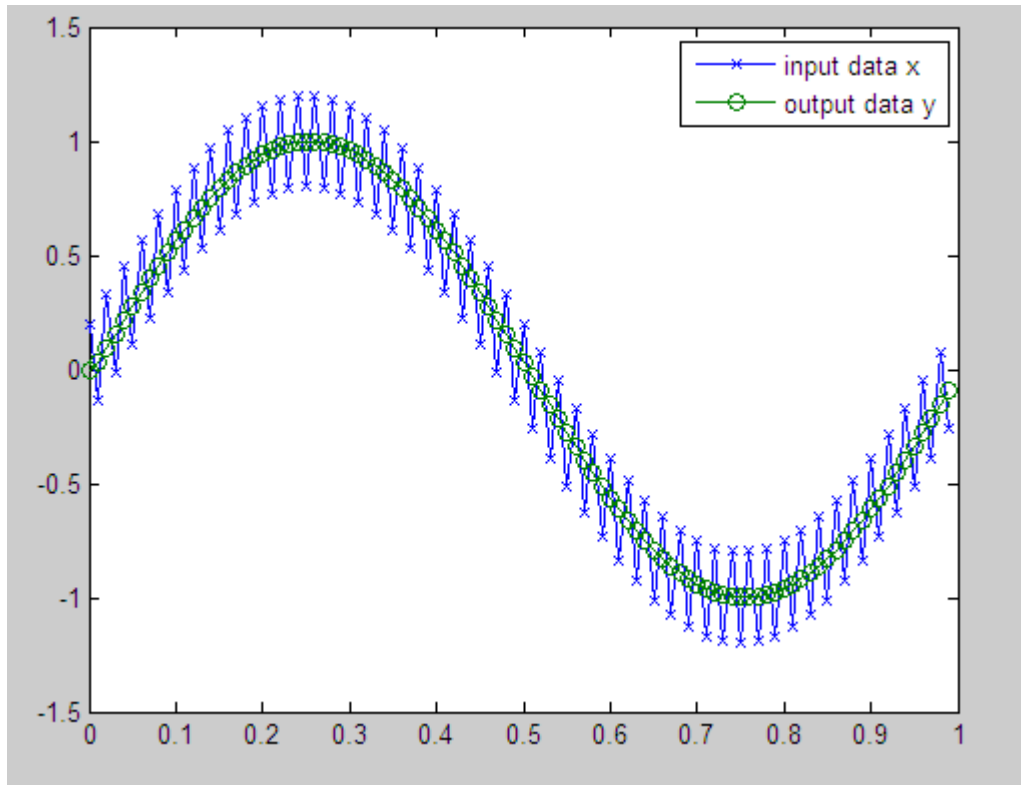
```
for k = 2:n
```

subsasgn

```
acc(1) = b(1)*x(k);  
acc(1) = acc + b(2)*x(k-1);  
y(k) = acc;  
end
```

By averaging every other sample, the loop shown above passes the low-frequency sinusoid through and attenuates the high-frequency sinusoid.

```
plot(t,x,'x-',t,y,'o-')  
legend('input data x','output data y')
```



The log report shows the minimum and maximum logged values and ranges of the variables used. Because `acc` is assigned into, rather than over written, these logs reflect the accumulated minimum and maximum values.

```
logreport(x,y,b,acc)
```

The table below shows selected output from the log report:

Value	minlog	maxlog	lowerbound	upperbound
x	-1.200012	1.197998	-2	1.999939
y	-0.9990234	0.9990234	-2	1.999939
b	0.5	0.5	-1	0.9999695
acc	-0.9990234	0.9989929	-512	512

Display `acc` to verify that its data type did not change:

```
acc
```

```
acc =
```

```
-0.0941
```

```

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

```

```

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

```

subsasgn

See Also

subsref

Purpose Subscripted reference

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

sum

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`, `mrdivide`, `numericType`, `rdivide`, `sub`

Purpose Create 3-D shaded surface plot

Description Refer to the MATLAB surf reference page for more information.

surf

Purpose Create 3-D shaded surface plot with contour plot

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

Purpose Create surface plot with colormap-based lighting

Description Refer to the MATLAB `surf1` reference page for more information.

surfnorm

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.

times

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 value. A scalar value can be multiplied by any other value.

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

Note For information about the `fimath` properties involved in Fixed-Point Toolbox calculations, see “Using `fimath` Objects to Perform Fixed-Point Arithmetic” and “Using `fimath` ProductMode and SumMode” in the *Fixed-Point Toolbox User’s Guide*.

For information about calculations using Simulink Fixed Point software, see the “Arithmetic Operations” chapter of the *Simulink Fixed Point User’s Guide*.

See Also `plus`, `minus`, `mtimes`, `uminus`

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

```
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
```

tostring

Purpose Convert numeric type or quantizer object to string

Syntax `s = tostring(T)`
`s = tostring(q)`

Description `s = tostring(T)` converts numeric type object `T` to a string `s` such that `eval(s)` would create a numeric type object with the same properties as `T`.

`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`.

Examples This example uses the `tostring` function to convert a numeric type object `T` to a string `s`

```
T = numericType(true,16,15);
s = tostring(T);
T1 = eval(s);
isequal(T,T1)
```

```
ans =
```

```
1
```

See Also `eval`, `numericTypeQuantizer`

Purpose Transpose operation

Description Refer to the MATLAB arithmetic operators reference page for more information.

treeplot

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.

trimesh

Purpose Create triangular mesh plot

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

Purpose Create 2-D triangular plot

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

trisurf

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.

uint8

Purpose Stored integer value of `fi` object as built-in `uint8`

Syntax `c = uint8(a)`

Description Fixed-point numbers can be represented as

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

or, equivalently as

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

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

`c = 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`, `int64`, `uint16`, `uint32`, `uint64`

Purpose Stored integer value of `fi` object as built-in `uint16`

Syntax `c = uint16(a)`

Description Fixed-point numbers can be represented as

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

or, equivalently as

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

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

`c = 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`, `int64`, `uint8`, `uint32`, `uint64`

uint32

Purpose Stored integer value of `fi` object as built-in `uint32`

Syntax `c = uint32(a)`

Description Fixed-point numbers can be represented as

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

or, equivalently as

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

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

`c = 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`, `int64`, `uint8`, `uint16`, `uint64`

Purpose Stored integer value of `fi` object as built-in `uint64`

Syntax `c = uint64(a)`

Description Fixed-point numbers can be represented as

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

or, equivalently as

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

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

`c = uint64(a)` returns the stored integer value of `fi` object `a` as a built-in `uint64`. If the stored integer word length is too big for a `uint64`, or if the stored integer is signed, the returned value saturates to a `uint64`.

See Also `int`, `int8`, `int16`, `int32`, `int64`, `uint8`, `uint16`, `uint32`

uminus

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, $-(-1) = 0.99\dots$:

```

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'

```

uminus

ans =

```
-1 + 0.99219i  
-1 + 0.99219i  
s8,7
```

See Also

plus, minus, mtimes, times

Purpose Quantize except numbers within eps of +1

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

Description `y = unitquantize(q, x)` works the same as `quantize` except that numbers within `eps(q)` of +1 are made exactly equal to +1 .

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

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

Examples This example demonstrates the use of `unitquantize` with a quantizer object `q` and a vector `x`.

```
q = quantizer('fixed','floor','saturate',[4 3]);
x = (0.8:.1:1.2)';
y = unitquantize(q,x);
z = [x y]
e = eps(q)
```

This quantization outputs an array containing the original values of `x` and the quantized values of `x`, followed by the value of `eps(q)`:

```
z =

    0.8000    0.7500
    0.9000    1.0000
    1.0000    1.0000
    1.1000    1.0000
    1.2000    1.0000
```

```
e =

    0.1250
```

unitquantize

See Also

eps, quantize, quantizer, unitquantizer

Purpose Constructor for unitquantizer object

Syntax `q = unitquantizer(...)`

Description `q = unitquantizer(...)` constructs a unitquantizer object, which is the same as a quantizer object in all respects except that its `quantize` method quantizes numbers within `eps(q)` of +1 to exactly +1.

See `quantizer` for parameters.

Examples In this example, a vector `x` is quantized by a unitquantizer object `u`.

```
u = unitquantizer([4 3]);
x = (0.8:.1:1.2)';
y = quantize(u,x);
z = [x y]
e = eps(u)
```

This quantization outputs an array containing the original values of `x` and the values of `x` that were quantized by the unitquantizer object `u`. The output also includes `e`, the value of `eps(u)`.

```
z =

    0.8000    0.7500
    0.9000    1.0000
    1.0000    1.0000
    1.1000    1.0000
    1.2000    1.0000
```

```
e =

    0.1250
```

See Also `quantize`, `quantizer`, `unitquantize`

unshiftdata

Purpose Inverse of shiftdata

Syntax `y = unshiftdata(x,perm,nshifts)`

Description `y = unshiftdata(x,perm,nshifts)` restores the orientation of the data that was shifted with `shiftdata`. The permutation vector is given by `perm`, and `nshifts` is the number of shifts that was returned from `shiftdata`.

`unshiftdata` is meant to be used in tandem with `shiftdata`. These functions are useful for creating functions that work along a certain dimension, like `filter`, `goertzel`, `sgolayfilt`, and `sosfilt`.

Examples

Example 1

This example shifts `x`, a 3-by-3 magic square, permuting dimension 2 to the first column. `unshiftdata` shifts `x` back to its original shape.

1. Create a 3-by-3 magic square:

```
x = fi(magic(3))
```

```
x =
```

```
     8     1     6
     3     5     7
     4     9     2
```

2. Shift the matrix `x` to work along the second dimension:

```
[x,perm,nshifts] = shiftdata(x,2)
```

This command returns the permutation vector, `perm`, and the number of shifts, `nshifts`, are returned along with the shifted matrix, `x`:

```
x =
```

```

      8     3     4
      1     5     9
      6     7     2

```

```
perm =
```

```
      2     1

```

```
nshifts =
```

```
      []

```

3. Shift the matrix back to its original shape:

```
y = unshiftdata(x,perm,nshifts)
```

```
y =
```

```

      8     1     6
      3     5     7
      4     9     2

```

Example 2

This example shows how `shiftdata` and `unshiftdata` work when you define `dim` as empty.

1. Define `x` as a row vector:

```
x = 1:5
```

```
x =
```

```
      1     2     3     4     5

```

unshiftdata

2. Define `dim` as empty to shift the first non-singleton dimension of `x` to the first column:

```
[x,perm,nshifts] = shiftdata(x,[])
```

This command returns `x` as a column vector, along with `perm`, the permutation vector, and `nshifts`, the number of shifts:

```
x =
```

```
1  
2  
3  
4  
5
```

```
perm =
```

```
[]
```

```
nshifts =
```

```
1
```

3. Using `unshiftdata`, restore `x` to its original shape:

```
y = unshiftdata(x,perm,nshifts)
```

```
y =
```

```
1    2    3    4    5
```

See Also

`ipermute`, `shiftdata`, `shiftdim`

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 <code>fi</code> objects
Syntax	<code>c = vertcat(a,b,...)</code> <code>[a; b; ...]</code> <code>[a;b]</code>
Description	<p><code>c = vertcat(a,b,...)</code> is called for the syntax <code>[a; b; ...]</code> when any of <code>a</code>, <code>b</code>, <code>...</code>, is a <code>fi</code> object.</p> <p><code>[a;b]</code> is the vertical concatenation of matrices <code>a</code> and <code>b</code>. <code>a</code> and <code>b</code> 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.</p> <p>Horizontal and vertical concatenation can be combined, as in <code>[1 2;3 4]</code>.</p> <p><code>[a b; c]</code> is allowed if the number of rows of <code>a</code> equals the number of rows of <code>b</code>, and if the number of columns of <code>a</code> plus the number of columns of <code>b</code> equals the number of columns of <code>c</code>.</p> <p>The matrices in a concatenation expression can themselves be formed via a concatenation, as in <code>[a b;[c d]]</code>.</p> <hr/> <p>Note The <code>fi</code>math and <code>numeric</code>type objects of a concatenated matrix of <code>fi</code> objects <code>c</code> are taken from the leftmost <code>fi</code> object in the list <code>(a,b,...)</code>.</p> <hr/>
See Also	<code>horzcat</code>

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 Logical exclusive-OR

Description Refer to the MATLAB xor reference page for more information.

ylim

Purpose Set or query y-axis limits

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

Purpose Set or query z-axis limits

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

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

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

where the slope can be expressed as

$$\textit{slope} = \textit{fractional slope} \times 2^{\textit{exponent}}$$

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 software.

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, signedness, 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

$$\textit{real-world value} = \textit{mantissa} \times 2^{\textit{exponent}}$$

2. Fixed-point numbers can be represented as

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

where the slope can be expressed as

$$\textit{slope} = \textit{fractional slope} \times 2^{\textit{exponent}}$$

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

$$\textit{exponent} = -1 \times \textit{fraction length}$$

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

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

where the slope can be expressed as

$$\textit{slope} = \textit{fractional slope} \times 2^{\textit{exponent}}$$

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

$$\text{real-world value} = \text{mantissa} \times 2^{\text{exponent}}$$

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

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

where the slope can be expressed as

$$\text{slope} = \text{fractional slope} \times 2^{\text{exponent}}$$

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

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

or

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

where the slope can be expressed as

$$\text{slope} = \text{fractional slope} \times 2^{\text{exponent}}$$

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

$$\text{weight of LSB} = 2^{-\text{fraction length}}$$

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

$$\text{real-world value} = \text{mantissa} \times 2^{\text{exponent}}$$

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 software.

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

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

or

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

where the slope can be expressed as

$$\text{slope} = \text{fractional slope} \times 2^{\text{exponent}}$$

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 software 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, signedness, 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, signedness, unsigned fixed-point

signedness

The signedness of a number or data type can be signed or unsigned. Signed numbers and data types can represent both positive and negative values, whereas unsigned numbers and data types can only represent values that are greater than or equal to zero.

See also data type, sign bit, sign/magnitude representation, signed fixed-point, 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

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

where the slope can be expressed as

$$\textit{slope} = \textit{fractional slope} \times 2^{\textit{exponent}}$$

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:

$$\textit{real - world value} = \textit{stored integer}$$

In [Slope Bias] representation, fixed-point numbers can be represented as

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

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:

$$\textit{real - world value} = \textit{stored integer} \times 2^{-\textit{fraction length}} = \textit{stored integer} \times 2^0$$

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, signedness

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 software.

See also ceiling (round toward), convergent rounding, floor (round toward), nearest (round toward), rounding, truncation

A

abs function 3-2
add function 3-16
all function 3-18
and function 3-19
any function 3-20
area function 3-21
assignmentquantizer function 3-22

B

bar function 3-23
barh function 3-24
Bias property 1-15
bin function 3-25
bin property 1-2
bin2num function 3-26
bitand function 3-28
bitandreduce function 3-29
bitcmp function 3-31
bitconcat function 3-32
bitget function 3-34
bitor function 3-36
bitorreduce function 3-37
bitreplicate function 3-39
bitrol function 3-40
bitror function 3-42
bitset function 3-44
bitshift function 3-45
bitsliceget function 3-48
bitsll function 3-50
bitsra function 3-52
bitsrl function 3-54
bitxor function 3-55
bitxorreduce function 3-56
buffer function 3-58

C

CastBeforeSum property 1-4

ceil function 3-59
clabel function 3-62
comet function 3-63
comet3 function 3-64
compass function 3-65
complex function 3-66
coneplot function 3-67
conj function 3-68
contour function 3-69
contour3 function 3-70
contourc function 3-71
contourf function 3-72
convergent function 3-73
copyobj function 3-77
ctranspose function 3-78

D

data property 1-2
DataType property 1-15
DataTypeMode property 1-15
DataTypeOverride property 1-12
dec function 3-79
dec property 1-2
denormalmax function 3-80
denormalmin function 3-81
diag function 3-82
disp function 3-83
div function 3-84
double function 3-89
double property 1-2

E

end function 3-90
eps function 3-91
eq function 3-92
errmean function 3-93
errorbar function 3-94
errpdf function 3-95

errvar function 3-98
etreeplot function 3-99
exponentbias function 3-100
exponentlength function 3-101
exponentmax function 3-102
exponentmin function 3-103
ezcontour function 3-104
ezcontourf function 3-105
ezmesh function 3-106
ezplot function 3-107
ezplot3 function 3-108
ezpolar function 3-109
ezsurf function 3-110
ezsurf function 3-111

F

feather function 3-112
fi function 3-113
fi objects

- properties
 - bin 1-2
 - data 1-2
 - dec 1-2
 - double 1-2
 - fimath 1-2
 - hex 1-3
 - int 1-3
 - NumericType 1-3
 - oct 1-3

fimath function 3-121
fimath objects

- properties
 - CastBeforeSum 1-4
 - MaxProductWordLength 1-4
 - MaxSumWordLength 1-4
 - OverflowMode 1-4
 - ProductBias 1-5
 - ProductFixedExponent 1-5
 - ProductFractionLength 1-5

ProductMode 1-5
ProductSlope 1-7
ProductSlopeAdjustmentFactor 1-7
ProductWordLength 1-7
RoundMode 1-8
SumBias 1-8
SumFixedExponent 1-8
SumFractionLength 1-9
SumMode 1-9
SumSlope 1-11
SumSlopeAdjustmentFactor 1-11
SumWordLength 1-11

fimath property 1-2
FimathDisplay property 1-12
fipref function 3-124
fipref objects

- properties
 - DataTypeOverride 1-12
 - FimathDisplay 1-12
 - LoggingMode 1-12
 - NumberDisplay 1-13
 - NumericTypeDisplay 1-13

fix function 3-126
FixedExponent property 1-16
flipdim function 3-129
fliplr function 3-130
flipud function 3-131
floor function 3-132
format

- rat 1-14

Format property 1-20
fplot function 3-135
fractionlength function 3-136
FractionLength property 1-17
function

- line 3-190

functions

- abs 3-2
- add 3-16
- all 3-18

and 3-19
any 3-20
area 3-21
assignmentquantizer 3-22
bar 3-23
barh 3-24
bin 3-25
bin2num 3-26
bitand 3-28
bitandreduce 3-29
bitcmp 3-31
bitconcat 3-32
bitget 3-34
bitor 3-36
bitorreduce 3-37
bitreplicate 3-39
bitrol 3-40
bitror 3-42
bitset 3-44
bitshift 3-45
bitsliceget 3-48
bitsll 3-50
bitsra 3-52
bitsrl 3-54
bitxor 3-55
bitxorreduce 3-56
buffer 3-58
ceil 3-59
clabel 3-62
comet 3-63
comet3 3-64
compass 3-65
complex 3-66
coneplot 3-67
conj 3-68
contour 3-69
contour3 3-70
contourc 3-71
contourf 3-72
convergent 3-73
copyobj 3-77
ctranspose 3-78
dec 3-79
denormalmax 3-80
denormalmin 3-81
diag 3-82
disp 3-83
div 3-84
double 3-89
end 3-90
eps 3-91
eq 3-92
errmean 3-93
errorbar 3-94
errpdf 3-95
errvar 3-98
etreeplot 3-99
exponentbias 3-100
exponentlength 3-101
exponentmax 3-102
exponentmin 3-103
ezcontour 3-104
ezcontourf 3-105
ezmesh 3-106
ezplot 3-107
ezplot3 3-108
ezpolar 3-109
ezsurf 3-110
ezsurfc 3-111
feather 3-112
fi 3-113
fimath 3-121
fipref 3-124
fix 3-126
flipdim 3-129
fliplr 3-130
flipud 3-131
floor 3-132
fplot 3-135
fractionlength 3-136

ge 3-137
get 3-138
getlsb 3-139
getmsb 3-140
gplot 3-141
gt 3-142
hankel 3-143
hex 3-144
hex2num 3-145
hist 3-146
histc 3-147
horzcat 3-148
imag 3-149
int 3-152
int16 3-155
int32 3-156
int64 3-157
int8 3-154
intmax 3-158
intmin 3-159
ipermute 3-160
isboolean 3-161
iscolumn 3-162
isdouble 3-163
isempty 3-164
isequal 3-165
isfi 3-166
isfinite 3-168
isfipref 3-169
isfixed 3-170
isfloat 3-171
isinf 3-172
isnan 3-173
isnumeric 3-174
isnumerictype 3-175
isobject 3-176
isquantizer 3-178
isreal 3-179
isrow 3-180
isscalar 3-181
isscaleddouble 3-182
isscaledtype 3-183
issigned 3-184
issingle 3-185
isslopebiasscaled 3-186
isvector 3-187
le 3-188
length 3-189
logical 3-191
loglog 3-192
logreport 3-193
lowerbound 3-194
lsb 3-195
lt 3-196
max 3-197
maxlog 3-198
mesh 3-200
meshc 3-201
meshz 3-202
min 3-203
minlog 3-204
minus 3-206
mpy 3-207
mrdivide 3-209
mtimes 3-211
ndgrid 3-212
ndims 3-213
ne 3-214
nearest 3-215
noperations 3-218
not 3-219
noverflows 3-220
num2bin 3-221
num2hex 3-222
num2int 3-224
numberofelements 3-225
numerictype 3-226
nunderflows 3-231
oct 3-232
or 3-233

patch 3-234
pcolor 3-235
permute 3-236
plot 3-237
plot3 3-238
plotmatrix 3-239
plotyy 3-240
plus 3-241
polar 3-242
pow2 3-243
quantize 3-248
quantizer 3-251
quiver 3-256
quiver3 3-257
randquant 3-258
range 3-260
rdivide 3-262
real 3-265
realmax 3-266
realmin 3-268
reinterpretcast 3-269
repmat 3-271
rescale 3-272
reset 3-274
resetdefaultfimath 3-275
resetlog 3-278
reshape 3-279
rgbplot 3-280
ribbon 3-281
rose 3-282
round 3-283
savedefaultfimathpref 3-288
savefipref 3-289
scatter 3-290
scatter3 3-291
sdec 3-292
semilogx 3-293
semilogy 3-294
set 3-295
setdefaultfimath 3-297
shiftdata 3-299
shiftdim 3-303
sign 3-304
single 3-305
size 3-306
slice 3-307
sort 3-308
spy 3-309
sqrt 3-310
squeeze 3-312
stairs 3-313
stem 3-314
stem3 3-315
streamribbon 3-316
streamslice 3-317
streamtube 3-318
stripscaling 3-319
sub 3-321
subsasgn 3-323
subsref 3-329
sum 3-330
surf 3-331
surfc 3-332
surf1 3-333
surfnorm 3-334
text 3-335
times 3-336
toeplitz 3-337
tostring 3-340
transpose 3-341
treemap 3-342
tril 3-343
trimesh 3-344
triplot 3-345
trisurf 3-346
triu 3-347
uint16 3-349
uint32 3-350
uint64 3-351
uint8 3-348

uminus 3-352
unitquantize 3-355
unitquantizer 3-357
unshiftdata 3-358
uplus 3-361
upperbound 3-362
vertcat 3-363
voronoi 3-364
voronoin 3-365
waterfall 3-366
wordlength 3-367
xlim 3-368
xor 3-369
ylim 3-370
zlim 3-371

G

ge function 3-137
get function 3-138
getlsb function 3-139
getmsb function 3-140
gplot function 3-141
gt function 3-142

H

hankel function 3-143
hex function 3-144
hex property 1-3
hex2num function 3-145
hist function 3-146
histc function 3-147
horzcat function 3-148

I

imag function 3-149
int function 3-152
int property 1-3
int16 function 3-155

int32 function 3-156
int64 function 3-157
int8 function 3-154
intmax function 3-158
intmin function 3-159
ipermute function 3-160
isboolean function 3-161
iscolumn function 3-162
isdoube function 3-163
isempty function 3-164
isequal function 3-165
isfi function 3-166
isfinite function 3-168
isfipref function 3-169
isfixed function 3-170
isfloat function 3-171
isinf function 3-172
isnan function 3-173
isnumeric function 3-174
isnumerictype function 3-175
isobject function 3-176
isquantizer function 3-178
isreal function 3-179
isrow function 3-180
isscalar function 3-181
isscaleddouble function 3-182
isscaledtype function 3-183
assigned function 3-184
issingle function 3-185
isslopebiasscaled function 3-186
isvector function 3-187

L

le function 3-188
length function 3-189
line function 3-190
LoggingMode property 1-12
logical function 3-191
loglog function 3-192

logreport function 3-193
 lowerbound function 3-194
 lsb function 3-195
 lt function 3-196

M

max function 3-197
 maxlog function 3-198
 MaxProductWordLength property 1-4
 MaxSumWordLength property 1-4
 mesh function 3-200
 meshc function 3-201
 meshz function 3-202
 min function 3-203
 minlog function 3-204
 minus function 3-206
 Mode property 1-20
 mpy function 3-207
 mrdivide function 3-209
 mtimes function 3-211

N

ndgrid function 3-212
 ndims function 3-213
 ne function 3-214
 nearest function 3-215
 nopnerations function 3-218
 not function 3-219
 noverflows function 3-220
 num2bin function 3-221
 num2hex function 3-222
 num2int function 3-224
 NumberDisplay property 1-13
 numberofelements function 3-225
 numerictype function 3-226
 numerictype objects
 properties
 Bias 1-15

DataType 1-15
 DataTypeMode 1-15
 FixedExponent 1-16
 FractionLength 1-17
 Scaling 1-17
 Signed 1-17
 Signedness 1-18
 Slope 1-18
 SlopeAdjustmentFactor 1-18
 WordLength 1-19

NumericType property 1-3
 NumericTypeDisplay property 1-13
 nunderflows function 3-231

O

oct function 3-232
 oct property 1-3
 or function 3-233
 OverflowMode property
 fimath objects 1-4
 quantizers 1-21

P

patch function 3-234
 pcolor function 3-235
 permute function 3-236
 plot function 3-237
 plot3 function 3-238
 plotmatrix function 3-239
 ploty function 3-240
 plus function 3-241
 polar function 3-242
 pow2 function 3-243
 ProductBias property 1-5
 ProductFixedExponent property 1-5
 ProductFractionLength property 1-5
 ProductMode property 1-5
 ProductSlope property 1-7

- ProductSlopeAdjustmentFactor property 1-7
 - ProductWordLength property 1-7
 - properties
 - Bias, numerictype objects 1-15
 - bin, fi objects 1-2
 - CastBeforeSum, fimath objects 1-4
 - data, fi objects 1-2
 - DataType, numerictype objects 1-15
 - DataTypeMode, numerictype objects 1-15
 - DataTypeOverride, fipref objects 1-12
 - dec, fi objects 1-2
 - double, fi objects 1-2
 - fimath, fi objects 1-2
 - FimathDisplay, fipref objects 1-12
 - FixedExponent, numerictype objects 1-16
 - Format, quantizers 1-20
 - FractionLength, numerictype objects 1-17
 - hex, fi objects 1-3
 - int, fi objects 1-3
 - LoggingMode, fipref objects 1-12
 - MaxProductWordLength, fimath objects 1-4
 - MaxSumWordLength, fimath objects 1-4
 - Mode, quantizers 1-20
 - NumberDisplay, fipref objects 1-13
 - NumericType, fi objects 1-3
 - NumericTypeDisplay, fipref objects 1-13
 - oct, fi objects 1-3
 - OverflowMode, fimath objects 1-4
 - OverflowMode, quantizers 1-21
 - ProductBias, fimath objects 1-5
 - ProductFixedExponent, fimath objects 1-5
 - ProductFractionLength, fimath objects 1-5
 - ProductMode, fimath objects 1-5
 - ProductSlope, fimath objects 1-7
 - ProductSlopeAdjustmentFactor, fimath objects 1-7
 - ProductWordLength, fimath objects 1-7
 - RoundMode, fimath objects 1-8
 - RoundMode, quantizers 1-22
 - Scaling, numerictype objects 1-17
 - Signed, numerictype objects 1-17
 - Signedness, numerictype objects 1-18
 - Slope, numerictype objects 1-18
 - SlopeAdjustmentFactor, numerictype objects 1-18
 - SumBias, fimath objects 1-8
 - SumFixedExponent, fimath objects 1-8
 - SumFractionLength, fimath objects 1-9
 - SumMode, fimath objects 1-9
 - SumSlope, fimath objects 1-11
 - SumSlopeAdjustmentFactor, fimath objects 1-11
 - SumWordLength, fimath objects 1-11
 - WordLength, numerictype objects 1-19
- Q**
- quantize function 3-248
 - quantizer function 3-251
 - quantizers
 - properties
 - Format 1-20
 - Mode 1-20
 - OverflowMode 1-21
 - RoundMode 1-22
 - quiver function 3-256
 - quiver3 function 3-257
- R**
- randquant function 3-258
 - range function 3-260
 - rat format 1-14
 - rdivide function 3-262
 - real function 3-265
 - realmax function 3-266
 - realmin function 3-268
 - reinterpretcast function 3-269
 - repmat function 3-271
 - rescale function 3-272

reset function 3-274
resetdefaultfimath function 3-275
resetlog function 3-278
reshape function 3-279
rgbplot function 3-280
ribbon function 3-281
rose function 3-282
round function 3-283
RoundMode property
 fimath objects 1-8
 quantizers 1-22

S

savedefaultfimathpref function 3-288
savefipref function 3-289
Scaling property 1-17
scatter function 3-290
scatter3 function 3-291
sdec function 3-292
semilogx function 3-293
semilogy function 3-294
set function 3-295
setdefaultfimath function 3-297
shiftdata function 3-299
shiftdim function 3-303
sign function 3-304
Signed property 1-17
Signedness property 1-18
single function 3-305
size function 3-306
slice function 3-307
Slope property 1-18
SlopeAdjustmentFactor property 1-18
sort function 3-308
spy function 3-309
sqrt function 3-310
squeeze function 3-312
stairs function 3-313
stem function 3-314

stem3 function 3-315
streamribbon function 3-316
streamslice function 3-317
streamtube function 3-318
stripscaling function 3-319
sub function 3-321
subsasgn function 3-323
subsref function 3-329
sum function 3-330
SumBias property 1-8
SumFixedExponent property 1-8
SumFractionLength property 1-9
SumMode property 1-9
SumSlope property 1-11
SumSlopeAdjustmentFactor property 1-11
SumWordLength property 1-11
surf function 3-331
surfc function 3-332
surf1 function 3-333
surfnorm function 3-334

T

text function 3-335
times function 3-336
toeplitz function 3-337
tostring function 3-340
transpose function 3-341
treeplot function 3-342
tril function 3-343
trimesh function 3-344
triplot function 3-345
trisurf function 3-346
triu function 3-347

U

uint16 function 3-349
uint32 function 3-350
uint64 function 3-351

uint8 function 3-348
uminus function 3-352
unitquantize function 3-355
unitquantizer function 3-357
unshiftdata function 3-358
uplus function 3-361
upperbound function 3-362

V

vertcat function 3-363
voronoi function 3-364
voronoin function 3-365

W

waterfall function 3-366

wordlength function 3-367
WordLength property 1-19

X

xlim function 3-368
xor function 3-369

Y

ylim function 3-370

Z

zlim function 3-371