$$\sqrt{\sum abs\left(w[i]\right)^2}$$

This function does not support text waves.

See Also
MatrixOp

NormalizeUnicode

`NormalizeUnicode(`*`sourceTextStr`*`, `*`normalizationForm`*`[, `*`options`*`])`

The NormalizeUnicode function normalizes the UTF-8-encoded text in sourceTextStr using the specified normalization form. The output text encoding is UTF-8.

NormalizeUnicode was added in Igor Pro 7.00. Most users will have no need for this function and can ignore it.

As explained under Details, in Unicode there are sometimes multiple ways to spell what appears visually to be the same word. This can cause problems when comparing text. Two strings that appear to represent the same word and which you consider equivalent may be spelled differently, causing a comparison operation to indicate that they are unequal. The NormalizeUnicode function converts *sourceTextStr* to a normalized form, which aides comparison.

Parameters

sourceTextStr is the text that you want to normalize. It must be encoded as UTF-8.

normalizationForm specifies the normalization form to use. These forms are described at http://unicode.org/reports/tr15/#Norm_Forms. The allowed values are:

0:	NFD (Canonical Decomposition)
1:	NFC (Canonical Decomposition, followed by Canonical Composition)
2:	NFKD (Compatibility Decomposition)
3:	NFKC (Compatibility Decomposition, followed by Canonical Composition)

options is a bitwise parameter, with the bits defined as follows:

Bit 0:	If cleared, in the event of an error, a null string is returned and an error is generated. Use this if you want to abort procedure execution if an error occurs. If set, in the event of an error, a null string is returned but no error is generated. Use this if you want to detect and handle an error yourself. You can test for null using strlen as shown in **String Variable Text Encoding Error Example** on page III-479.

All other bits are reserved and must be cleared.

Details

The Unicode standard specifies that some sequences of code points represent essentially the same character. There are two types of equivalence: canonical equivalence and compatibility.

Sequences of code points defined as canonically equivalent are assumed to have the same appearance and meaning when printed or displayed. For example, the code point U+006E (LATIN SMALL LETTER N) followed by U+0303 (COMBINING TILDE) is defined by Unicode to be canonically equivalent to the single code point U+00F1 (LATIN SMALL LETTER N WITH TILDE). The former is called "decomposed" while the later is called "precomposed".

Sequences that are defined as compatible are assumed to have possibly distinct appearances, but the same meaning in some contexts. Thus, for example, the code point U+FB00 (LATIN SMALL LIGATURE FF) is defined to be compatible, but not canonically equivalent, to the sequence U+0066 U+0066 (two Latin "f" letters). Sequences that are canonically equivalent are also compatible, but the opposite is not necessarily true.