Enumeration

vDSP

An enumeration that acts as a namespace for Swift overlays to vDSP.

iOS 13.0+ | iPadOS 13.0+ | Mac Catalyst | macOS 10.15+ | tvOS 13.0+ | visionOS | watchOS 6.0+

```
enum vDSP
```

Mentioned in

Performing Fourier transforms on interleaved-complex data

Topics

Type Methods

`static func absolute<U>(U) -> [Double]`

Returns the absolute value of each element in the supplied double-precision vector.

`static func absolute<U>(U) -> [Float]`

Returns the absolute value of each element in the supplied single-precision vector.

`static func absolute<V>(DSPSplitComplex, result: inout V)`

Calculates the absolute value of each element in the supplied single-precision complex vector.

`static func absolute<V>(DSPDoubleSplitComplex, result: inout V)`

Calculates the absolute value of each element in the supplied double-precision complex vector.

```
static func absolute<U, V>(U, result: inout V)
```

Calculates the absolute value of each element in the supplied double-precision vector.

```
static func absolute<U, V>(U, result: inout V)
```

Calculates the absolute value of each element in the supplied single-precision vector.

```
static func add<U>(Double, U) -> [Double]
```

Returns the double-precision element-wise sum of a vector and a scalar value.

```
static func add<T, U>(T, U) -> [Double]
```

Returns the double-precision element-wise sum of two vectors.

```
static func add<U>(Float, U) -> [Float]
```

Returns the single-precision element-wise sum of a vector and a scalar value.

```
static func add<T, U>(T, U) -> [Float]
```

Returns the single-precision element-wise sum of two vectors.

```
static func add<U, V>(Double, U, result: inout V)
```

Calculates the single-precision element-wise sum of a vector and a scalar value.

```
static func add<U, V>(Float, U, result: inout V)
```

Calculates the single-precision element-wise sum of a vector and a scalar value.

```
static func add<T, U, V>(T, U, result: inout V)
```

Calculates the double-precision element-wise sum of two vectors.

```
static func add<T, U, V>(T, U, result: inout V)
```

Calculates the single-precision element-wise sum of two vectors.

```
static func add(DSPSplitComplex, to: DSPSplitComplex, count: Int,
result: inout DSPSplitComplex)
```

Calculates the single-precision elementwise sum of the supplied complex vectors.

```
static func add(DSPDoubleSplitComplex, to: DSPDoubleSplitComplex, count
: Int, result: inout DSPDoubleSplitComplex)
```

Calculates the double-precision elementwise sum of the supplied complex vectors.

```
static func add<U>(multiplication: (a: U, b: Double), Double) -> [
Double]
```

Returns the double-precision element-wise addition of the product of a vector and a scalar value, and a vector.

`static func add<T, U>(multiplication: (a: T, b: Double), U) -> [Double]`

Returns the double-precision element-wise addition of the product of a vector and a scalar value, and a vector.

`static func add<T, U>(multiplication: (a: T, b: U), Double) -> [Double]`

Returns the double-precision element-wise sum of the product of two vectors, and a scalar value.

`static func add<S, T, U>(multiplication: (a: S, b: T), U) -> [Double]`

Returns the double-precision element-wise sum of a vector and the product of two vectors.

`static func add<U>(multiplication: (a: U, b: Float), Float) -> [Float]`

Returns the single-precision element-wise addition of the product of a vector and a scalar value, and a vector.

`static func add<T, U>(multiplication: (a: T, b: Float), U) -> [Float]`

Returns the single-precision element-wise addition of the product of a vector and a scalar value, and a vector.

`static func add<T, U>(multiplication: (a: T, b: U), Float) -> [Float]`

Returns the single-precision element-wise sum of the product of two vectors, and a scalar value.

`static func add<S, T, U>(multiplication: (a: S, b: T), U) -> [Float]`

Returns the single-precision element-wise sum of a vector and the product of two vectors.

`static func add<U, V>(multiplication: (a: U, b: Double), Double, result: inout V)`

Calculates the double-precision element-wise addition of the product of a vector and a scalar value, and a scalar.

`static func add<T, U, V>(multiplication: (a: T, b: Double), U, result: inout V)`

Calculates the double-precision element-wise addition of the product of a vector and a scalar value, and a vector.

`static func add<U, V>(multiplication: (a: U, b: Float), Float, result: inout V)`

Calculates the single-precision element-wise addition of the product of a vector and a scalar value, and a scalar.

```
static func add<T, U, V>(multiplication: (a: T, b: Float), U, result: inout V)
```

Calculates the single-precision element-wise addition of the product of a vector and a scalar value, and a vector.

```
static func add<T, U, V>(multiplication: (a: T, b: U), Double, result: inout V)
```

Calculates the double-precision element-wise sum of the product of two vectors, and a scalar value.

```
static func add<T, U, V>(multiplication: (a: T, b: U), Float, result: inout V)
```

Calculates the single-precision element-wise sum of the product of two vectors, and a scalar value.

```
static func add<S, T, U, V>(multiplication: (a: S, b: T), U, result: inout V)
```

Calculates the double-precision element-wise sum of a vector and the product of two vectors.

```
static func add<S, T, U, V>(multiplication: (a: S, b: T), U, result: inout V)
```

Calculates the single-precision element-wise sum of a vector and the product of two vectors.

```
static func add<T, U>(multiplication: (a: T, b: Double), multiplication: (c: U, d: Double)) -> [Double]
```

Returns the double-precision element-wise addition of two vector-scalar products.

```
static func add<R, S, T, U>(multiplication: (a: R, b: S), multiplication: (c: T, d: U)) -> [Double]
```

Returns the double-precision elementwise product of a vector and a vector, added to a second product of a vector and a vector.

```
static func add<T, U>(multiplication: (a: T, b: Float), multiplication: (c: U, d: Float)) -> [Float]
```

Returns the single-precision element-wise addition of two vector-scalar products.

```
static func add<R, S, T, U>(multiplication: (a: R, b: S), multiplication: (c: T, d: U)) -> [Float]
```

Returns the single-precision elementwise product of a vector and a vector, added to a second product of a vector and a vector.

```
static func add<T, U, V>(multiplication: (a: T, b: Double),
multiplication: (c: U, d: Double), result: inout V)
```

Calculates the double-precision element-wise addition of two vector-scalar products.

```
static func add<T, U, V>(multiplication: (a: T, b: Float),
multiplication: (c: U, d: Float), result: inout V)
```

Calculates the single-precision element-wise addition of two vector-scalar products.

```
static func add<R, S, T, U, V>(multiplication: (a: R, b: S),
multiplication: (c: T, d: U), result: inout V)
```

Calculates the double-precision elementwise product of a vector and a vector, added to a second product of a vector and a vector.

```
static func add<R, S, T, U, V>(multiplication: (a: R, b: S),
multiplication: (c: T, d: U), result: inout V)
```

Calculates the single-precision elementwise product of a vector and a vector, added to a second product of a vector and a vector.

```
static func addSubtract<S, T, U, V>(S, T, addResult: inout U, subtract
Result: inout V)
```

Calculates the double-precision element-wise sum and subtraction of two vectors.

```
static func addSubtract<S, T, U, V>(S, T, addResult: inout U, subtract
Result: inout V)
```

Calculates the single-precision element-wise sum and subtraction of two vectors.

```
static func amplitudeToDecibels<U>(U, zeroReference: Double) -> [Double
]
```

Returns double-precision amplitude values converted to decibel values.

```
static func amplitudeToDecibels<U>(U, zeroReference: Float) -> [Float]
```

Returns single-precision amplitude values converted to decibels.

```
static func clear<V>(inout V)
```

Populates a double-precision vector with zeros.

```
static func clear<V>(inout V)
```

Populates a single-precision vector with zeros.

```
static func clip<U>(U, to: ClosedRange<Double>) -> [Double]
```

Returns the elements of a double-precision vector clipped to the specified range.

```
static func clip<U>(U, to: ClosedRange<Float>) -> [Float]
```

Returns the elements of a single-precision vector clipped to the specified range.

```
static func clip<U, V>(U, to: ClosedRange<Double>, result: inout V)
```

Calculates the elements of a double-precision vector clipped to the specified range.

```
static func clip<U, V>(U, to: ClosedRange<Float>, result: inout V)
```

Calculates the elements of a single-precision vector clipped to the specified range.

```
static func conjugate(DSPSplitComplex, count: Int, result: inout DSPSplitComplex)
```

Calculates the complex conjugate of the values in a single-precision vector.

```
static func conjugate(DSPDoubleSplitComplex, count: Int, result: inout DSPDoubleSplitComplex)
```

Calculates the complex conjugate of the values in a double-precision vector.

```
static func convert<U, V>(amplitude: U, toDecibels: inout V, zeroReference: Double)
```

Converts double-precision amplitude values to decibel values.

```
static func convert<U, V>(amplitude: U, toDecibels: inout V, zeroReference: Float)
```

Converts single-precision amplitude values to decibel values.

```
static func convert(interleavedComplexVector: [DSPDoubleComplex], toSplitComplexVector: inout DSPDoubleSplitComplex)
```

Converts the contents of an interleaved double-precision complex vector to a split complex vector.

```
static func convert(interleavedComplexVector: [DSPComplex], toSplitComplexVector: inout DSPSplitComplex)
```

Converts the contents of an interleaved single-precision complex vector to a split complex vector.

```
static func convert<U, V>(polarCoordinates: U, toRectangularCoordinates: inout V)
```

Converts double-precision polar coordinates to rectangular coordinates.

```
static func convert<U, V>(polarCoordinates: U, toRectangularCoordinates: inout V)
```

Converts single-precision polar coordinates to rectangular coordinates.

```
static func convert<U, V>(power: U, toDecibels: inout V, zeroReference: Double)
```

Converts double-precision power values to decibel values.

```
static func convert<U, V>(power: U, toDecibels: inout V, zeroReference: Float)
```

Converts single-precision power values to decibel values.

```
static func convert<U, V>(rectangularCoordinates: U, toPolarCoordinates: inout V)
```

Converts double-precision rectangular coordinates to polar coordinates.

```
static func convert<U, V>(rectangularCoordinates: U, toPolarCoordinates: inout V)
```

Converts single-precision rectangular coordinates to polar coordinates.

```
static func convert(splitComplexVector: DSPDoubleSplitComplex, toInterleavedComplexVector: inout [DSPDoubleComplex])
```

Converts the contents of a split double-precision complex vector to an interleaved vector.

```
static func convert(splitComplexVector: DSPSplitComplex, toInterleavedComplexVector: inout [DSPComplex])
```

Converts the contents of a split single-precision complex vector to an interleaved vector.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts double-precision values to single-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts single-precision values to double-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 8-bit signed integers to double-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 8-bit signed integers to single-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 16-bit signed integers to double-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 16-bit signed integers to single-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 32-bit signed integers to double-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 32-bit signed integers to single-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 8-bit unsigned integers to double-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 8-bit unsigned integers to single-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 16-bit unsigned integers to double-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 16-bit unsigned integers to single-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 32-bit unsigned integers to double-precision values.

```
static func convertElements<U, V>(of: U, to: inout V)
```

Converts 32-bit unsigned integers to single-precision values.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.
RoundingMode)
```

Converts double-precision values to 8-bit signed integers.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.
RoundingMode)
```

Converts double-precision values to 16-bit signed integers.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.
RoundingMode)
```

Converts double-precision values to 32-bit signed integers.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.
RoundingMode)
```

Converts double-precision values to 8-bit unsigned integers.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.
RoundingMode)
```

Converts double-precision values to 16-bit unsigned integers.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.RoundingMode)
```

Converts double-precision values to 32-bit unsigned integers.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.RoundingMode)
```

Converts single-precision values to 8-bit signed integers.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.RoundingMode)
```

Converts single-precision values to 16-bit unsigned integers.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.RoundingMode)
```

Converts single-precision values to 32-bit signed integers.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.RoundingMode)
```

Converts single-precision values to 8-bit unsigned integers.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.RoundingMode)
```

Converts single-precision values to 16-bit signed integers.

```
static func convertElements<U, V>(of: U, to: inout V, rounding: vDSP.RoundingMode)
```

Converts single-precision values to 32-bit unsigned integers.

```
static func convolve<T, U>(T, rowCount: Int, columnCount: Int, with3x3Kernel: U) -> [Double]
```

Returns the 2D convolution of a double-precision vector with a 3 x 3 kernel.

```
static func convolve<T, U>(T, rowCount: Int, columnCount: Int, with3x3Kernel: U) -> [Float]
```

Returns the 2D convolution of a single-precision vector with a 3 x 3 kernel.

```
static func convolve<T, U, V>(T, rowCount: Int, columnCount: Int, with3x3Kernel: U, result: inout V)
```

Calculates the 2D convolution of a double-precision vector with a 3 x 3 kernel.

```
static func convolve<T, U, V>(T, rowCount: Int, columnCount: Int, with3x3Kernel: U, result: inout V)
```

Calculates the 2D convolution of a single-precision vector with a 3 x 3 kernel.

```
static func convolve<T, U>(T, rowCount: Int, columnCount: Int, with5x5Kernel: U) -> [Double]
```

Returns the 2D convolution of a double-precision vector with a 5 x 5 kernel.

```
static func convolve<T, U>(T, rowCount: Int, columnCount: Int, with5x5Kernel: U) -> [Float]
```

Returns the 2D convolution of a single-precision vector with a 5 x 5 kernel.

```
static func convolve<T, U, V>(T, rowCount: Int, columnCount: Int, with5x5Kernel: U, result: inout V)
```

Calculates the 2D convolution of a double-precision vector with a 5 x 5 kernel.

```
static func convolve<T, U, V>(T, rowCount: Int, columnCount: Int, with5x5Kernel: U, result: inout V)
```

Calculates the 2D convolution of a single-precision vector with a 5 x 5 kernel.

```
static func convolve<T, U>(T, rowCount: Int, columnCount: Int, withKernel: U, kernelRowCount: Int, kernelColumnCount: Int) -> [Double]
```

Returns the 2D convolution of a double-precision vector with an arbitrarily sized kernel.

```
static func convolve<T, U>(T, rowCount: Int, columnCount: Int, withKernel: U, kernelRowCount: Int, kernelColumnCount: Int) -> [Float]
```

Returns the 2D convolution of a single-precision vector with an arbitrarily sized kernel.

```
static func convolve<T, U, V>(T, rowCount: Int, columnCount: Int, withKernel: U, kernelRowCount: Int, kernelColumnCount: Int, result: inout V)
```

Calculates the 2D convolution of a double-precision vector with an arbitrarily sized kernel.

```
static func convolve<T, U, V>(T, rowCount: Int, columnCount: Int, withKernel: U, kernelRowCount: Int, kernelColumnCount: Int, result: inout V)
```

Calculates the 2D convolution of a single-precision vector with an arbitrarily sized kernel.

```
static func convolve<T, U>(T, withKernel: U) -> [Double]
```

Returns the 1D convolution of a double-precision vector.

```
static func convolve<T, U>(T, withKernel: U) -> [Float]
```

Returns the 1D convolution of a single-precision vector.

```
static func convolve<T, U, V>(T, withKernel: U, result: inout V)
```

Calculates the 1D convolution of a double-precision vector.

```
static func convolve<T, U, V>(T, withKernel: U, result: inout V)
```

Calculates the 1D convolution of a single-precision vector.

```
static func copy(DSPSplitComplex, to: inout DSPSplitComplex, count: Int
)
```

Copies a complex single-precision vector.

```
static func copy(DSPDoubleSplitComplex, to: inout DSPDoubleSplitComplex
, count: Int)
```

Copies a complex double-precision vector.

```
static func correlate<T, U>(T, withKernel: U) -> [Double]
```

Returns the correlation of a double-precision signal vector and a filter vector.

```
static func correlate<T, U>(T, withKernel: U) -> [Float]
```

Returns the correlation of a single-precision signal vector and a filter vector.

```
static func correlate<T, U, V>(T, withKernel: U, result: inout V)
```

Calculates the correlation of a double-precision signal vector and a filter vector.

```
static func correlate<T, U, V>(T, withKernel: U, result: inout V)
```

Calculates the correlation of a single-precision signal vector and a filter vector.

```
static func countZeroCrossings<U>(U) -> UInt
```

Returns the number of zero crossings in a double-precision vector.

```
static func countZeroCrossings<U>(U) -> UInt
```

Returns the number of zero crossings in a single-precision vector.

```
static func distanceSquared<U, V>(U, V) -> Double
```

Returns the double-precision distance squared between two points in n-dimensional space.

```
static func distanceSquared<U, V>(U, V) -> Float
```

Returns the single-precision distance squared between two points in n-dimensional space.

```
static func divide<U>(Double, U) -> [Double]
```

Returns the double-precision element-wise division of a scalar value and a vector.

`static func divide<U>(U, Double) -> [Double]`

Calculates the double-precision element-wise division of a vector and a scalar value.

`static func divide<T, U>(T, U) -> [Double]`

Returns the double-precision element-wise division of two vectors.

`static func divide<U>(Float, U) -> [Float]`

Returns the single-precision element-wise division of a scalar value and a vector.

`static func divide<U>(U, Float) -> [Float]`

Calculates the single-precision element-wise division of a vector and a scalar value.

`static func divide<T, U>(T, U) -> [Float]`

Returns the single-precision element-wise division of two vectors.

`static func divide<U, V>(Double, U, result: inout V)`

Calculates the double-precision element-wise division of a scalar value and a vector.

`static func divide<U, V>(Float, U, result: inout V)`

Calculates the single-precision element-wise division of a scalar value and a vector.

`static func divide<U, V>(U, Double, result: inout V)`

Calculates the double-precision element-wise division of a vector and a scalar value.

`static func divide<U, V>(U, Float, result: inout V)`

Calculates the single-precision element-wise division of a vector and a scalar value.

`static func divide<T, U, V>(T, U, result: inout V)`

Calculates the double-precision element-wise division of two vectors.

`static func divide<T, U, V>(T, U, result: inout V)`

Calculates the single-precision element-wise division of two vectors.

`static func divide(DSPSplitComplex, by: DSPSplitComplex, count: Int, result: inout DSPSplitComplex)`

Calculates the single-precision elementwise division of a complex vector by a complex vector.

`static func divide(DSPDoubleSplitComplex, by: DSPDoubleSplitComplex, count: Int, result: inout DSPDoubleSplitComplex)`

Calculates the double-precision elementwise division of a complex vector by a complex vector.

```
static func divide<U>(DSPSplitComplex, by: U, result: inout DSPSplit
Complex)
```

Calculates the single-precision elementwise division of a complex vector by a real vector.

```
static func divide<U>(DSPDoubleSplitComplex, by: U, result: inout
DSPDoubleSplitComplex)
```

Calculates the double-precision elementwise division of a complex vector by a complex vector.

```
static func doubleToFloat<U>(U) -> [Float]
```

Returns single-precision values converted from a double-precision source.

```
static func downsample<T, U>(U, decimationFactor: Int, filter: T) -> [
Double]
```

Returns the downsampled double-precision vector.

```
static func downsample<T, U>(U, decimationFactor: Int, filter: T) -> [
Float]
```

Returns the downsampled single-precision vector.

```
static func downsample<T, U, V>(U, decimationFactor: Int, filter: T,
result: inout V)
```

Calculates the downsampled double-precision vector.

```
static func downsample<T, U, V>(U, decimationFactor: Int, filter: T,
result: inout V)
```

Calculates the downsampled single-precision vector.

```
static func evaluatePolynomial<U>(usingCoefficients: [Double], with
Variables: U) -> [Double]
```

Returns a double-precision evaluated polynomial using specified coefficients and variables.

```
static func evaluatePolynomial<U>(usingCoefficients: [Float], with
Variables: U) -> [Float]
```

Returns a single-precision evaluated polynomial using specified coefficients and variables.

```
static func evaluatePolynomial<U, V>(usingCoefficients: [Double], with
Variables: U, result: inout V)
```

Evaluates a double-precision polynomial using specified coefficients and variables.

```
static func evaluatePolynomial<U, V>(usingCoefficients: [Float], with
Variables: U, result: inout V)
```

Evaluates a single-precision polynomial using specified coefficients and variables.

```
static func fill<V>(inout V, with: Double)
```

Populates a double-precision vector with a specified scalar value.

```
static func fill<V>(inout V, with: Float)
```

Populates a single-precision vector with a specified scalar value.

```
static func floatToDouble<U>(U) -> [Double]
```

Returns double-precision values converted from a single-precision source.

```
static func floatingPointToInteger<T, U>(T, integerType: U.Type,
rounding: vDSP.RoundingMode) -> [U]
```

Returns double-precision values converted to integer values.

```
static func floatingPointToInteger<T, U>(T, integerType: U.Type,
rounding: vDSP.RoundingMode) -> [U]
```

Returns single-precision values converted to integer values.

```
static func formRamp<V>(in: ClosedRange<Double>, result: inout V)
```

Populates a single-precision vector with monotonically incrementing or decrementing values within a range.

```
static func formRamp<V>(in: ClosedRange<Float>, result: inout V)
```

Populates a double-precision vector with monotonically incrementing or decrementing values within a range.

```
static func formRamp<V>(withInitialValue: Double, increment: Double,
result: inout V)
```

Populates a double-precision vector with monotonically incrementing or decrementing values using an initial value and increment.

```
static func formRamp<V>(withInitialValue: Float, increment: Float,
result: inout V)
```

Populates a single-precision vector with monotonically incrementing or decrementing values using an initial value and increment.

```
static func formRamp<U, V>(withInitialValue: inout Double, multiplying
By: U, increment: Double, result: inout V)
```

Populates a double-precision vector that contains monotonically incrementing or decrementing values, and multiplies that vector by a source vector.

```
static func formRamp<U, V>(withInitialValue: inout Float, multiplyingBy
: U, increment: Float, result: inout V)
```

Populates a single-precision vector that contains monotonically incrementing or decrementing values, and multiplies that vector by a source vector.

```
static func formStereoRamp<U, V>(withInitialValue: inout Double,
multiplyingBy: U, U, increment: Double, results: inout V, inout V)
```

Populates two single-precision vectors that contain stereo monotonically incrementing or decrementing values multiplied by two source vectors.

```
static func formStereoRamp<U, V>(withInitialValue: inout Float,
multiplyingBy: U, U, increment: Float, results: inout V, inout V)
```

Populates two single-precision vectors that contain stereo monotonically incrementing or decrementing values multiplied by two source vectors.

```
static func formWindow<V>(usingSequence: vDSP.WindowSequence, result:
inout V, isHalfWindow: Bool)
```

Populates a double-precision vector with a specified window.

```
static func formWindow<V>(usingSequence: vDSP.WindowSequence, result:
inout V, isHalfWindow: Bool)
```

Populates a single-precision vector with a specified window.

```
static func hypot<U, V>(U, V) -> [Double]
```

Returns the double-precision hypotenuses of right triangles with legs that are the lengths of corresponding elements of the two input vectors.

```
static func hypot<U, V>(U, V) -> [Float]
```

Returns the single-precision hypotenuses of right triangles with legs that are the lengths of corresponding elements of the two input vectors.

```
static func hypot<T, U, V>(T, U, result: inout V)
```

Calculates the double-precision hypotenuses of right triangles with legs that are the lengths of corresponding elements of the two input vectors.

```
static func hypot<T, U, V>(T, U, result: inout V)
```

Calculates the single-precision hypotenuses of right triangles with legs that are the lengths of corresponding elements of the two input vectors.

```
static func hypot<R, S, T, U>(x0: R, x1: S, y0: T, y1: U) -> [Double]
```

Returns the double-precision hypotenuses of right triangles with legs that are the differences of corresponding elements of two pairs of vectors.

```
static func hypot<R, S, T, U>(x0: R, x1: S, y0: T, y1: U) -> [Float]
```

Returns the single-precision hypotenuses of right triangles with legs that are the differences of corresponding elements of two pairs of vectors.

```
static func hypot<R, S, T, U, V>(x0: R, x1: S, y0: T, y1: U, result: inout V)
```

Calculates the double-precision hypotenuses of right triangles with legs that are the differences of corresponding elements of two pairs of vectors.

```
static func hypot<R, S, T, U, V>(x0: R, x1: S, y0: T, y1: U, result: inout V)
```

Calculates the single-precision hypotenuses of right triangles with legs that are the differences of corresponding elements of two pairs of vectors.

```
static func indexOfMaximum<U>(U) -> (UInt, Double)
```

Returns the maximum value and corresponding index in a double-precision vector.

```
static func indexOfMaximum<U>(U) -> (UInt, Float)
```

Returns the maximum value and corresponding index in a single-precision vector.

```
static func indexOfMaximumMagnitude<U>(U) -> (UInt, Double)
```

Returns the maximum magnitude and corresponding index in a double-precision vector.

```
static func indexOfMaximumMagnitude<U>(U) -> (UInt, Float)
```

Returns the maximum magnitude and corresponding index in a single-precision vector.

```
static func indexOfMinimum<U>(U) -> (UInt, Double)
```

Returns the maximum value and corresponding index in a double-precision vector.

```
static func indexOfMinimum<U>(U) -> (UInt, Float)
```

Returns the maximum value and corresponding index in a single-precision vector.

```
static func integerToFloatingPoint<T, U>(T, floatingPointType: U.Type) -> [U]
```

Returns a vector of floating-point values converted from signed 8-bit integer values.

```
static func integerToFloatingPoint<T, U>(T, floatingPointType: U.Type) -> [U]
```

Returns a vector of floating-point values converted from signed 16-bit integer values.

```
static func integerToFloatingPoint<T, U>(T, floatingPointType: U.Type) -> [U]
```

Returns a vector of floating-point values converted from signed 32-bit integer values.

```
static func integerToFloatingPoint<T, U>(T, floatingPointType: U.Type)
-> [U]
```

Returns a vector of floating-point values converted from unsigned 8-bit integer values.

```
static func integerToFloatingPoint<T, U>(T, floatingPointType: U.Type)
-> [U]
```

Returns a vector of floating-point values converted from unsigned 16-bit integer values.

```
static func integerToFloatingPoint<T, U>(T, floatingPointType: U.Type)
-> [U]
```

Returns a vector of floating-point values converted from unsigned 32-bit integer values.

```
static func integrate<U>(U, using: vDSP.IntegrationRule, stepSize:
Double) -> [Double]
```

Returns the integration of a double-precision vector using the specified rule.

```
static func integrate<U>(U, using: vDSP.IntegrationRule, stepSize:
Float) -> [Float]
```

Returns the integration of a single-precision vector using the specified rule.

```
static func integrate<U, V>(U, using: vDSP.IntegrationRule, stepSize:
Double, result: inout V)
```

Performs the integration of a double-precision using the specified rule.

```
static func integrate<U, V>(U, using: vDSP.IntegrationRule, stepSize:
Float, result: inout V)
```

Performs the integration of a single-precision using the specified rule.

```
static func invertedClip<U>(U, to: ClosedRange<Double>) -> [Double]
```

Returns a double-precision vector that's inverted-clipped to the specified range.

```
static func invertedClip<U>(U, to: ClosedRange<Float>) -> [Float]
```

Returns a single-precision vector that's inverted-clipped to the specified range.

```
static func invertedClip<U, V>(U, to: ClosedRange<Double>, result:
inout V)
```

Calculates a double-precision vector that's inverted-clipped to the specified range.

```
static func invertedClip<U, V>(U, to: ClosedRange<Float>, result: inout
 V)
```

Calculates a single-precision vector that's inverted-clipped to the specified range.

```
static func limit<U>(U, limit: Double, withOutputConstant: Double) -> [Double]
```

Returns the double-precision vector test limit.

```
static func limit<U>(U, limit: Float, withOutputConstant: Float) -> [Float]
```

Returns the single-precision vector test limit.

```
static func limit<U, V>(U, limit: Double, withOutputConstant: Double, result: inout V)
```

Calculates the double-precision vector test limit.

```
static func limit<U, V>(U, limit: Float, withOutputConstant: Float, result: inout V)
```

Calculates the single-precision vector test limit.

```
static func linearInterpolate<T, U>(T, U, using: Double) -> [Double]
```

Returns the linear interpolation between the supplied double-precision vectors.

```
static func linearInterpolate<T, U>(T, U, using: Float) -> [Float]
```

Returns the linear interpolation between the supplied single-precision vectors.

```
static func linearInterpolate<T, U, V>(T, U, using: Double, result: inout V)
```

Calculates the linear interpolation between the supplied double-precision vectors.

```
static func linearInterpolate<T, U, V>(T, U, using: Float, result: inout V)
```

Calculates the linear interpolation between the supplied single-precision vectors.

```
static func linearInterpolate<T, U>(elementsOf: T, using: U) -> [Double]
```

Returns the interpolation between the neighboring elements of a double-precision vector.

```
static func linearInterpolate<T, U>(elementsOf: T, using: U) -> [Float]
```

Returns the interpolation between the neighboring elements of a single-precision vector.

```
static func linearInterpolate<T, U, V>(elementsOf: T, using: U, result: inout V)
```

Calculates the interpolation between the neighboring elements of a double-precision vector.

`static func linearInterpolate<T, U, V>(elementsOf: T, using: U, result: inout V)`

Calculates the interpolation between the neighboring elements of a single-precision vector.

`static func maximum<U>(U) -> Double`

Returns the double-precision maximum value of a vector.

`static func maximum<U>(U) -> Float`

Returns the single-precision maximum value of a vector.

`static func maximum<U>(U, U) -> [Double]`

Returns a double-precision array containing the maximum of the corresponding values of two vectors.

`static func maximum<U>(U, U) -> [Float]`

Returns a single-precision array containing the maximum of the corresponding values of two vectors.

`static func maximum<U, V>(U, U, result: inout V)`

Calculates the maximum of the corresponding double-precision values of two vectors.

`static func maximum<U, V>(U, U, result: inout V)`

Calculates the maximum of the corresponding single-precision values of two vectors.

`static func maximumMagnitude<U>(U) -> Double`

Returns the double-precision maximum magnitude of a vector.

`static func maximumMagnitude<U>(U) -> Float`

Returns the single-precision maximum magnitude of a vector.

`static func mean<U>(U) -> Double`

Returns the mean value of a double-precision vector.

`static func mean<U>(U) -> Float`

Returns the mean value of a single-precision vector.

`static func meanMagnitude<U>(U) -> Double`

Returns the mean of magnitudes of a double-precision vector.

`static func meanMagnitude<U>(U) -> Float`

Returns the mean of magnitudes of a single-precision vector.

`static func meanSquare<U>(U) -> Double`

Returns the mean of squares of a double-precision vector.

`static func meanSquare<U>(U) -> Float`

Returns the mean of squares of a single-precision vector.

`static func minimum<U>(U) -> Double`

Returns the double-precision minimum value of a vector.

`static func minimum<U>(U) -> Float`

Returns the single-precision minimum value of a vector.

`static func minimum<U>(U, U) -> [Double]`

Returns a double-precision array containing the minimum of the corresponding values of two vectors.

`static func minimum<U>(U, U) -> [Float]`

Returns a single-precision array containing the minimum of the corresponding values of two vectors.

`static func minimum<U, V>(U, U, result: inout V)`

Calculates the double-precision minimum of the corresponding values of two vectors.

`static func minimum<U, V>(U, U, result: inout V)`

Calculates the single-precision minimum of the corresponding values of two vectors.

`static func multiply<U>(Double, U) -> [Double]`

Returns the double-precision element-wise product of a vector and a scalar value.

`static func multiply<T, U>(T, U) -> [Double]`

Returns the double-precision element-wise product of two vectors.

`static func multiply<U>(Float, U) -> [Float]`

Returns the single-precision element-wise product of a vector and a scalar value.

`static func multiply<T, U>(T, U) -> [Float]`

Returns the single-precision element-wise product of two vectors.

`static func multiply<U, V>(Double, U, result: inout V)`

Calculates the double-precision element-wise product of a vector and a scalar value.

`static func multiply<U, V>(Float, U, result: inout V)`

Calculates the single-precision element-wise product of a vector and a scalar value.

```
static func multiply<T, U, V>(T, U, result: inout V)
```

Calculates the double-precision element-wise product of two vectors.

```
static func multiply<T, U, V>(T, U, result: inout V)
```

Calculates the single-precision element-wise product of two vectors.

```
static func multiply(DSPSplitComplex, by: DSPSplitComplex, count: Int, useConjugate: Bool, result: inout DSPSplitComplex)
```

Calculates the product of two complex single-precision vectors, optionally conjugating one of them.

```
static func multiply(DSPDoubleSplitComplex, by: DSPDoubleSplitComplex, count: Int, useConjugate: Bool, result: inout DSPDoubleSplitComplex)
```

Calculates the elementwise product of two complex double-precision vectors, optionally conjugating one of them.

```
static func multiply<U>(DSPSplitComplex, by: U, result: inout DSPSplitComplex)
```

Calculates the double-precision elementwise product of a complex vector and a real vector.

```
static func multiply<U>(DSPDoubleSplitComplex, by: U, result: inout DSPDoubleSplitComplex)
```

Calculates the single-precision elementwise product of a complex vector and a real vector.

```
static func multiply<T, U>(addition: (a: T, b: U), Double) -> [Double]
```

Returns the double-precision element-wise product of the sum of two vectors and a scalar value.

```
static func multiply<S, T, U>(addition: (a: S, b: T), U) -> [Double]
```

Returns the double-precision element-wise product of a vector and the sum of two vectors.

```
static func multiply<T, U>(addition: (a: T, b: U), Float) -> [Float]
```

Returns the single-precision element-wise product of the sum of two vectors and a scalar value.

```
static func multiply<S, T, U>(addition: (a: S, b: T), U) -> [Float]
```

Returns the single-precision element-wise product of a vector and the sum of two vectors.

```
static func multiply<T, U, V>(addition: (a: T, b: U), Double, result: inout V)
```

Calculates the double-precision element-wise product of the sum of two vectors and a scalar value.

```
static func multiply<T, U, V>(addition: (a: T, b: U), Float, result: inout V)
```

Calculates the single-precision element-wise product of the sum of two vectors and a scalar value.

```
static func multiply<S, T, U, V>(addition: (a: S, b: T), U, result: inout V)
```

Calculates the double-precision element-wise product of a vector and the sum of two vectors.

```
static func multiply<S, T, U, V>(addition: (a: S, b: T), U, result: inout V)
```

Calculates the single-precision element-wise product of a vector and the sum of two vectors.

```
static func multiply<S, T, U>(addition: (a: S, b: T), addition: (c: U, d: U)) -> [Double]
```

Returns the double-precision element-wise product of the sums of two pairs of vectors.

```
static func multiply<S, T, U>(addition: (a: S, b: T), addition: (c: U, d: U)) -> [Float]
```

Returns the single-precision element-wise product of the sums of two pairs of vectors.

```
static func multiply<S, T, U, V>(addition: (a: S, b: T), addition: (c: U, d: U), result: inout V)
```

Calculates the double-precision element-wise product of the sums of two pairs of vectors.

```
static func multiply<S, T, U, V>(addition: (a: S, b: T), addition: (c: U, d: U), result: inout V)
```

Calculates the single-precision element-wise product of the sums of two pairs of vectors.

```
static func multiply<R, S, T, U>(addition: (a: R, b: S), subtraction: (c: T, d: U)) -> [Double]
```

Returns the double-precision element-wise product of the sum of two vectors and the difference of two vectors.

```
static func multiply<R, S, T, U>(addition: (a: R, b: S), subtraction: (c: T, d: U)) -> [Float]
```

Returns the single-precision element-wise product of the sum of two vectors and the difference of two vectors.

```
static func multiply<R, S, T, U, V>(addition: (a: R, b: S), subtraction
: (c: T, d: U), result: inout V)
```

Calculates the double-precision element-wise product of the sum of two vectors and the difference of two vectors.

```
static func multiply<R, S, T, U, V>(addition: (a: R, b: S), subtraction
: (c: T, d: U), result: inout V)
```

Calculates the double-precision element-wise product of the sum of two vectors and the difference of two vectors.

```
static func multiply<T, U>(subtraction: (a: T, b: U), Double) -> [
Double]
```

Returns the double-precision element-wise product of the difference of two vectors and a scalar value.

```
static func multiply<S, T, U>(subtraction: (a: S, b: T), U) -> [Double]
```

Returns the double-precision element-wise product of a vector and the differences of two vectors.

```
static func multiply<T, U>(subtraction: (a: T, b: U), Float) -> [Float]
```

Returns the single-precision element-wise product of the difference of two vectors and a scalar value.

```
static func multiply<S, T, U>(subtraction: (a: S, b: T), U) -> [Float]
```

Returns the single-precision element-wise product of a vector and the differences of two vectors.

```
static func multiply<T, U, V>(subtraction: (a: T, b: U), Double, result
: inout V)
```

Calculates the double-precision element-wise product of the difference of two vectors and a scalar value.

```
static func multiply<T, U, V>(subtraction: (a: T, b: U), Float, result:
inout V)
```

Calculates the single-precision element-wise product of the difference of two vectors and a scalar value.

```
static func multiply<S, T, U, V>(subtraction: (a: S, b: T), U, result:
inout V)
```

Calculates the double-precision element-wise product of a vector and the differences of two vectors.

```
static func multiply<S, T, U, V>(subtraction: (a: S, b: T), U, result: inout V)
```

Calculates the single-precision element-wise product of a vector and the differences of two vectors.

```
static func multiply<R, S, T, U>(subtraction: (a: R, b: S), subtraction: (c: T, d: U)) -> [Double]
```

Returns the double-precision element-wise product of the differences of two pairs of vectors.

```
static func multiply<R, S, T, U>(subtraction: (a: R, b: S), subtraction: (c: T, d: U)) -> [Float]
```

Returns the single-precision element-wise product of the differences of two pairs of vectors.

```
static func multiply<R, S, T, U, V>(subtraction: (a: R, b: S), subtraction: (c: T, d: U), result: inout V)
```

Calculates the double-precision element-wise product of the differences of two pairs of vectors.

```
static func multiply<R, S, T, U, V>(subtraction: (a: R, b: S), subtraction: (c: T, d: U), result: inout V)
```

Calculates the single-precision element-wise product of the differences of two pairs of vectors.

```
static func negative<U>(U) -> [Double]
```

Returns the negative value of each element in the supplied double-precision vector.

```
static func negative<U>(U) -> [Float]
```

Returns the negative value of each element in the supplied single-precision vector.

```
static func negative<U, V>(U, result: inout V)
```

Calculates the negative value of each element in the supplied double-precision vector.

```
static func negative<U, V>(U, result: inout V)
```

Calculates the negative value of each element in the supplied single-precision vector.

```
static func negativeAbsolute<U>(U) -> [Double]
```

Returns the negative absolute value of each element in the supplied double-precision vector.

```
static func negativeAbsolute<U>(U) -> [Float]
```

Returns the negative absolute value of each element in the supplied single-precision vector.

```
static func negativeAbsolute<U, V>(U, result: inout V)
```

Calculates the negative absolute value of each element in the supplied double-precision vector.

```
static func negativeAbsolute<U, V>(U, result: inout V)
```

Calculates the negative absolute value of each element in the supplied single-precision vector.

```
static func phase<V>(DSPSplitComplex, result: inout V)
```

Calculates the single-precision element-wise phase values, in radians, of the supplied complex vector.

```
static func phase<V>(DSPDoubleSplitComplex, result: inout V)
```

Calculates the double-precision element-wise phase values, in radians, of the supplied complex vector.

```
static func polarToRectangular<U>(U) -> [Double]
```

Returns double-precision rectangular coordinates converted from polar coordinates.

```
static func polarToRectangular<U>(U) -> [Float]
```

Returns single-precision rectangular coordinates converted from polar coordinates.

```
static func powerToDecibels<U>(U, zeroReference: Double) -> [Double]
```

Returns double-precision power values converted to decibel values.

```
static func powerToDecibels<U>(U, zeroReference: Float) -> [Float]
```

Returns single-precision power values converted to decibel values.

```
static func ramp(in: ClosedRange<Double>, count: Int) -> [Double]
```

Returns a single-precision vector that contains monotonically incrementing or decrementing values within a range.

```
static func ramp(in: ClosedRange<Float>, count: Int) -> [Float]
```

Returns a double-precision vector that contains monotonically incrementing or decrementing values within a range.

```
static func ramp(withInitialValue: Double, increment: Double, count: Int) -> [Double]
```

Returns a double-precision vector that contains monotonically incrementing or decrementing values using an initial value and increment.

```
static func ramp(withInitialValue: Float, increment: Float, count: Int) -> [Float]
```

Returns a single-precision vector that contains monotonically incrementing or decrementing values using an initial value and increment.

```
static func ramp<U>(withInitialValue: inout Double, multiplyingBy: U, increment: Double) -> [Double]
```

Returns a double-precision vector that contains monotonically incrementing or decrementing values, and multiplies that vector by a source vector.

```
static func ramp<U>(withInitialValue: inout Float, multiplyingBy: U, increment: Float) -> [Float]
```

Returns a single-precision vector that contains monotonically incrementing or decrementing values, and multiplies that vector by a source vector.

```
static func rectangularToPolar<U>(U) -> [Double]
```

Returns double-precision polar coordinates converted from rectangular coordinates.

```
static func rectangularToPolar<U>(U) -> [Float]
```

Returns single-precision polar coordinates converted from rectangular coordinates.

```
static func reverse<V>(inout V)
```

Reverses a vector of double-precision values in-place.

```
static func reverse<V>(inout V)
```

Reverses a vector of single-precision values in-place.

```
static func rootMeanSquare<U>(U) -> Double
```

Returns the root mean square of a double-precision vector.

```
static func rootMeanSquare<U>(U) -> Float
```

Returns the root mean square of a single-precision vector.

```
static func signedSquare<U>(U) -> [Double]
```

Returns a double-precision array containing the signed square of each element in the supplied vector.

```
static func signedSquare<U>(U) -> [Float]
```

Returns a single-precision array containing the signed square of each element in the supplied vector.

```
static func signedSquare<U, V>(U, result: inout V)
```

Calculates the signed square of each element in the supplied double-precision vector.

```
static func signedSquare<U, V>(U, result: inout V)
```

Calculates the signed square of each element in the supplied single-precision vector.

`static func slidingWindowSum<U>(U, usingWindowLength: Int) -> [Double]`

Returns the double-precision sliding window sum of a vector.

`static func slidingWindowSum<U>(U, usingWindowLength: Int) -> [Float]`

Returns the single-precision sliding window sum of a vector.

`static func slidingWindowSum<U, V>(U, usingWindowLength: Int, result: inout V)`

Calculates the double-precision sliding window sum of a vector.

`static func slidingWindowSum<U, V>(U, usingWindowLength: Int, result: inout V)`

Calculates the single-precision sliding window sum of a vector.

`static func sort<V>(inout V, sortOrder: vDSP.SortOrder)`

Sorts a vector of double-precision values in-place.

`static func sort<V>(inout V, sortOrder: vDSP.SortOrder)`

Sorts a vector of single-precision values in-place.

`static func square<U>(U) -> [Double]`

Returns a double-precision array containing the square of each element in the supplied vector.

`static func square<U>(U) -> [Float]`

Returns a single-precision array containing the square of each element in the supplied vector.

`static func square<U, V>(U, result: inout V)`

Calculates the square of each element in the supplied double-precision vector.

`static func square<U, V>(U, result: inout V)`

Calculates the square of each element in the supplied single-precision vector.

`static func squareMagnitudes<V>(DSPSplitComplex, result: inout V)`

Calculates the square magnitude of each element in the supplied single-precision complex vector.

`static func squareMagnitudes<V>(DSPDoubleSplitComplex, result: inout V)`

Calculates the square magnitude of each element in the supplied double-precision complex vector.

```
static func stereoRamp<U>(withInitialValue: inout Double, multiplyingBy: U, U, increment: Double) -> (firstOutput: [Double], secondOutput: [Double])
```

Returns two double-precision vectors that contain stereo monotonically incrementing or decrementing values multiplied by two source vectors.

```
static func stereoRamp<U>(withInitialValue: inout Float, multiplyingBy: U, U, increment: Float) -> (firstOutput: [Float], secondOutput: [Float])
```

Returns two single-precision vectors that contain stereo monotonically incrementing or decrementing values multiplied by two source vectors.

```
static func subtract<T, U>(U, T) -> [Double]
```

Returns the double-precision element-wise subtraction of two vectors.

```
static func subtract<T, U>(U, T) -> [Float]
```

Returns the single-precision element-wise subtraction of two vectors.

```
static func subtract<T, U, V>(U, T, result: inout V)
```

Calculates the double-precision element-wise subtraction of two vectors.

```
static func subtract<T, U, V>(U, T, result: inout V)
```

Calculates the single-precision element-wise subtraction of two vectors.

```
static func subtract(DSPSplitComplex, from: DSPSplitComplex, count: Int, result: inout DSPSplitComplex)
```

Calculates the single-precision elementwise subtraction of a complex vector from a complex vector.

```
static func subtract(DSPDoubleSplitComplex, from: DSPDoubleSplitComplex, count: Int, result: inout DSPDoubleSplitComplex)
```

Calculates the double-precision elementwise subtraction of a complex vector from a complex vector.

```
static func subtract<T, U>(multiplication: (a: U, b: Double), T) -> [Double]
```

Calculates the double-precision element-wise difference of the product of a vector and a scalar value, and a vector.

```
static func subtract<S, T, U>(multiplication: (a: T, b: U), S) -> [Double]
```

Returns the double-precision element-wise difference of a vector and the product of two vectors.

```
static func subtract<T, U>(multiplication: (a: U, b: Float), T) -> [Float]
```

Calculates the single-precision element-wise difference of the product of a vector and a scalar value, and a vector.

```
static func subtract<S, T, U>(multiplication: (a: T, b: U), S) -> [Float]
```

Returns the single-precision element-wise difference of a vector and the product of two vectors.

```
static func subtract<T, U, V>(multiplication: (a: U, b: Double), T, result: inout V)
```

Calculates the double-precision element-wise difference of the product of a vector and a scalar value, and a vector.

```
static func subtract<T, U, V>(multiplication: (a: U, b: Float), T, result: inout V)
```

Calculates the single-precision element-wise difference of the product of a vector and a scalar value, and a vector.

```
static func subtract<S, T, U, V>(multiplication: (a: T, b: U), S, result: inout V)
```

Calculates the double-precision element-wise difference of a vector and the product of two vectors.

```
static func subtract<S, T, U, V>(multiplication: (a: T, b: U), S, result: inout V)
```

Calculates the single-precision element-wise difference of a vector and the product of two vectors.

```
static func subtract<R, S, T, U>(multiplication: (a: T, b: U), multiplication: (c: R, d: S)) -> [Double]
```

Returns the double-precision element-wise difference of the products of two pairs of vectors.

```
static func subtract<R, S, T, U>(multiplication: (a: T, b: U), multiplication: (c: R, d: S)) -> [Float]
```

Returns the single-precision element-wise difference of the products of two pairs of vectors.

```
static func subtract<R, S, T, U, V>(multiplication: (a: T, b: U), multiplication: (c: R, d: S), result: inout V)
```

Calculates the double-precision element-wise difference of the products of two pairs of vectors.

```
static func subtract<R, S, T, U, V>(multiplication: (a: T, b: U), multiplication: (c: R, d: S), result: inout V)
```

Calculates the single-precision element-wise difference of the products of two pairs of vectors.

```
static func sum<U>(U) -> Double
```

Returns the double-precision vector sum.

```
static func sum<U>(U) -> Float
```

Returns the single-precision vector sum.

```
static func sumAndSumOfSquares<U>(U) -> (elementsSum: Double, squaresSum: Double)
```

Returns the double-precision vector sum and sum of squares.

```
static func sumAndSumOfSquares<U>(U) -> (elementsSum: Float, squaresSum: Float)
```

Returns the single-precision vector sum and sum of squares.

```
static func sumOfMagnitudes<U>(U) -> Double
```

Returns the double-precision vector sum of magnitudes.

```
static func sumOfMagnitudes<U>(U) -> Float
```

Returns the single-precision vector sum of magnitudes.

```
static func sumOfSquares<U>(U) -> Double
```

Returns the double-precision vector sum of squares.

```
static func sumOfSquares<U>(U) -> Float
```

Returns the single-precision vector sum of squares.

```
static func threshold<U>(U, to: Double, with: vDSP.ThresholdRule<Double>) -> [Double]
```

Returns the elements of the supplied double-precision vector after applying a specified thresholding rule.

```
static func threshold<U>(U, to: Float, with: vDSP.ThresholdRule<Float>) -> [Float]
```

Returns the elements of the supplied single-precision vector after applying a specified thresholding rule.

```
static func threshold<U, V>(U, to: Double, with: vDSP.ThresholdRule<
Double>, result: inout V)
```

Calculates the elements of the supplied double-precision vector after applying a specified thresholding rule.

```
static func threshold<U, V>(U, to: Float, with: vDSP.ThresholdRule<
Float>, result: inout V)
```

Calculates the elements of the supplied single-precision vector after applying a specified thresholding rule.

```
static func trunc<U>(U) -> [Double]
```

Returns a double-precision array containing each element in the supplied vector truncated to a fraction.

```
static func trunc<U>(U) -> [Float]
```

Returns a single-precision array containing each element in the supplied vector truncated to a fraction.

```
static func trunc<U, V>(U, result: inout V)
```

Calculates each element in the supplied double-precision vector truncated to a fraction.

```
static func trunc<U, V>(U, result: inout V)
```

Calculates each element in the supplied single-precision vector truncated to a fraction.

```
static func twoPoleTwoZeroFilter<U>(U, coefficients: (Double, Double,
Double, Double, Double)) -> [Double]
```

Returns the result of double-precision, two-pole, two-zero recursive filtering.

```
static func twoPoleTwoZeroFilter<U>(U, coefficients: (Float, Float,
Float, Float, Float)) -> [Float]
```

Returns the result of single-precision, two-pole, two-zero recursive filtering.

```
static func twoPoleTwoZeroFilter<U, V>(U, coefficients: (Double, Double
, Double, Double, Double), result: inout V)
```

Performs double-precision, two-pole, two-zero recursive filtering.

```
static func twoPoleTwoZeroFilter<U, V>(U, coefficients: (Float, Float,
Float, Float, Float), result: inout V)
```

Performs single-precision, two-pole, two-zero recursive filtering.

```
static func window<T>(ofType: T.Type, usingSequence: vDSP.Window
Sequence, count: Int, isHalfWindow: Bool) -> [T]
```

Returns an array that contains the specified window.

```
static func compress<T, U>(T, gatingVector: U, nonZeroGatingCount: Int
?) -> [Double]
```

Returns a compressed copy of the specified double-precision vector using the nonzero values in a gating vector.

```
static func compress<T, U>(T, gatingVector: U, nonZeroGatingCount: Int
?) -> [Float]
```

Returns a compressed copy of the specified single-precision vector using the nonzero values in a gating vector.

```
static func compress<T, U, V>(T, gatingVector: U, result: inout V)
```

Compresses the specified double-precision vector using the nonzero values in a gating vector.

```
static func compress<T, U, V>(T, gatingVector: U, result: inout V)
```

Compresses the specified single-precision vector using the nonzero values in a gating vector.

```
static func convertElements<U, V>(of: U, to: inout V)
```

```
static func convertElements<U, V>(of: U, to: inout V)
```

```
static func dot<T, U>(T, U) -> Double
```

```
static func dot<T, U>(T, U) -> Float
```

```
static func float16ToFloat<U>(U) -> [Float]
```

```
static func floatToFloat16<U>(U) -> [Float16]
```

```
static func formRamp<V>(from: Double, through: Double, result: inout V)
```

```
static func formRamp<V>(from: Float, through: Float, result: inout V)
```

```
static func gather<T, U>(T, indices: U) -> [Double]
```

Returns a gathered copy of the specified double-precision vector using a vector that defines the indices to keep.

```
static func gather<T, U>(T, indices: U) -> [Float]
```

Returns a gathered copy of the specified single-precision vector using a vector that defines the indices to keep.

```
static func gather<T, U, V>(T, indices: U, result: inout V)
```

Gathers the specified double-precision vector using a vector that defines the indices to keep.

```
static func gather<T, U, V>(T, indices: U, result: inout V)
```

Gathers the specified single-precision vector using a vector that defines the indices to keep.

```
static func linearInterpolate<T, U>(lookupTable: T, withOffsets: U,
scale: Double, baseOffset: Double) -> [Double]
```

Returns the double-precision linearly interpolated values of a lookup table from the specified offsets.

```
static func linearInterpolate<T, U>(lookupTable: T, withOffsets: U,
scale: Float, baseOffset: Float) -> [Float]
```

Returns the single-precision linearly interpolated values of a lookup table from the specified offsets.

```
static func linearInterpolate<T, U, V>(lookupTable: T, withOffsets: U,
scale: Double, baseOffset: Double, result: inout V)
```

Computes the double-precision linearly interpolated values of a lookup table from the specified offsets.

```
static func linearInterpolate<T, U, V>(lookupTable: T, withOffsets: U,
scale: Float, baseOffset: Float, result: inout V)
```

Computes the single-precision linearly interpolated values of a lookup table from the specified offsets.

```
static func linearInterpolate<T, U>(values: T, atIndices: U) -> [Double
]
```

Returns the double-precision linearly interpolated values of a vector at the specified indices.

```
static func linearInterpolate<T, U>(values: T, atIndices: U) -> [Float]
```

Returns the single-precision linearly interpolated values of a vector at the specified indices.

```
static func linearInterpolate<T, U, V>(values: T, atIndices: U, result:
inout V)
```

Computes the double-precision linearly interpolated values of a vector at the specified indices.

```
static func linearInterpolate<T, U, V>(values: T, atIndices: U, result:
inout V)
```

Computes the single-precision linearly interpolated values of a vector at the specified indices.

```
static func normalize(some AccelerateBuffer<Double>) -> [Double]
```

```
static func normalize(some AccelerateBuffer<Float>) -> [Float]
```

```
static func normalize(some AccelerateBuffer<Double>, result: inout some
 AccelerateMutableBuffer<Double>) -> (mean: Double, stdDev: Double)
```

```
static func normalize(some AccelerateBuffer<Float>, result: inout some
AccelerateMutableBuffer<Float>) -> (mean: Float, stdDev: Float)
```

```
static func ramp(from: Double, through: Double, count: Int) -> [Double]
```

```
static func ramp(from: Float, through: Float, count: Int) -> [Float]
```

```
static func standardDeviation(some AccelerateMutableBuffer<Double>) ->
Double
```

```
static func standardDeviation(some AccelerateMutableBuffer<Float>) ->
Float
```

```
static func swapElements<T, U>(inout T, inout U)
```

Swaps the elements of two double-precision vectors.

```
static func swapElements<T, U>(inout T, inout U)
```

Swaps the elements of two single-precision vectors.

```
static func taperedMerge<T, U>(T, U) -> [Double]
```

Returns the result of a tapered merge between two double-precision vectors.

```
static func taperedMerge<T, U>(T, U) -> [Float]
```

Returns the result of a tapered merge between two single-precision vectors.

```
static func taperedMerge<T, U, V>(T, U, result: inout V)
```

Computes the result of a tapered merge between two double-precision vectors.

```
static func taperedMerge<T, U, V>(T, U, result: inout V)
```

Computes the result of a tapered merge between two single-precision vectors.

Classes

```
class DCT
```

A single-precision discrete cosine transform.

~~`class DFT`~~

A single- and double-precision discrete Fourier transform.

Deprecated

```
class FFT
```

A 1D single- and double-precision fast Fourier transform.

`class` `FFT2D`

A 2D single- and double-precision fast Fourier transform.

`class` `DiscreteFourierTransform`

An object that provides forward and inverse discrete Fourier transforms on single- or double-precision collections of interleaved or split-complex data.

Structures

`struct` `Biquad`

A single- or double-precision biquadratic filter.

`struct` `VectorizableDouble`

A structure that represents a double-precision real value for biquadratic filtering and discrete Fourier transforms.

`struct` `VectorizableFloat`

A structure that represents a single-precision real value for biquadratic filtering and discrete Fourier transforms.

`struct` `DFTDoublePrecisionInterleavedFunctions`

`struct` `DFTDoublePrecisionSplitComplexFunctions`

`struct` `DFTSinglePrecisionInterleavedFunctions`

`struct` `DFTSinglePrecisionSplitComplexFunctions`

`struct` `vDSP_SplitComplexDouble`

`struct` `vDSP_SplitComplexFloat`

Enumerations

`enum` `DCTTransformType`

An enumeration that describes the discrete cosine transform types.

`enum` `DFTTransformType`

Discrete Fourier transform types.

`enum` `FourierTransformDirection`

Fast Fourier transform directions.

`enum IntegrationRule`

Integration rules.

`enum Radix`

Fast Fourier transform radices.

`enum RoundingMode`

Floating point to integer conversion rounding modes.

`enum SortOrder`

Constants that specify the sorting order.

`enum ThresholdRule`

Constants that specify vector threshold rules.

`enum WindowSequence`

Constants that specify window sequence functions.

`enum DFTError`

See Also

Swift overlay

vDSP Protocols

Protocols that support Swift implementations of vDSP operations.