API Collection

vForce

Perform transcendental and trigonometric functions on vectors of any length.

Overview

The vForce library provides a range of trigonometric and transcendental functions that work over large collections of single- and double-precision values. The collections can be of any length, and vForce supplies vectorized functions for the current architecture.

The functions declared in the vForce library have the customary mathematical names, but with the prefix `vv`, for example, `vvsqrtf(_:_:_:)`. Each mathematical function is available in two variants: one for single-precision data and one for double-precision data. The single-precision forms have the suffix `f`, whereas the double-precision forms have no suffix. For example, `vvcosf(_:_:_:)` is the single-precision cosine function, and `vvcos(_:_:_:)` is the double-precision variant.

All of the vForce library functions follow a common format:

- The return type is `void`.
- The first parameter points to an array to hold the results. The only exceptions are `vvsincosf(_:_:_:_:)` and `vvsincos(_:_:_:_:)`, which have two result arrays that the first two parameters point to.
- One or more parameters point to operand arrays that are the same length as the result array.
- The last parameter is the array length.

> **Note**
>
> Unless otherwise mentioned, vForce functions work in-place. That is, the input may exactly equal the output.

Using vForce

The vForce library provides a high-performance alternative to `for` loops and `map(_:)` when applying operations on arrays of floating-point values.

For example, given an arbitrarily sized array, `x`, that contains single-precision values, the following code uses `map(_:)` to create a second array, `y`. On return, `y` contains the square root of each array element.

```
let n = 10_000

let x = (0..<n).map { _ in
    Float.random(in: 1 ... 10_000)
}

let y = x.map {
    return sqrt($0)
}
```

The equivalent functionality implemented in vForce runs significantly faster:

```
let y = [Float](unsafeUninitializedCapacity: n) { buffer, initializedCount in
    vForce.sqrt(x,
                result: &buffer)

    initializedCount = n
}
```

Topics

Swift Overlay

`enum` **vForce**

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

Array-Oriented Arithmetic and Auxiliary Functions

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

Returns the ceiling of each element in a vector of double-precision values.

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

Returns the ceiling of each element in a vector of single-precision values.

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

Calculates the ceiling of each element in a vector of double-precision values.

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

Calculates the ceiling of each element in a vector of single-precision values.

```
static func copysign<U, V>(magnitudes: U, signs: V) -> [Double]
```

Returns each single-precision element in the magnitudes vector, setting its sign to the corresponding elements in the signs vector.

```
static func copysign<U, V>(magnitudes: U, signs: V) -> [Float]
```

Returns each single-precision element in the magnitudes vector, setting its sign to the corresponding elements in the signs vector.

```
static func copysign<T, U, V>(magnitudes: T, signs: U, result: inout V)
```

Calculates each double-precision element in the magnitudes vector, setting its sign to the corresponding elements in the signs vector.

```
static func copysign<T, U, V>(magnitudes: T, signs: U, result: inout V)
```

Calculates each single-precision element in the magnitudes vector, setting its sign to the corresponding elements in the signs vector.

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

Returns the floor of each element in a vector of double-precision values.

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

Returns the floor of each element in a vector of single-precision values.

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

Calculates the floor of each element in a vector of double-precision values.

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

Calculates the floor of each element in a vector of single-precision values.

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

Returns the nearest integer to each element in a vector of double-precision values.

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

Returns the nearest integer to each element in a vector of single-precision values.

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

Calculates the nearest integer to each element in a vector of double-precision values.

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

Calculates the nearest integer to each element in a vector of double-precision values.

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

Returns the reciprocal of each element in a vector of double-precision values.

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

Returns the reciprocal of each element in a vector of single-precision values.

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

Calculates the reciprocal of each element in a vector of double-precision values.

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

Calculates the reciprocal of each element in a vector of single-precision values.

```
static func remainder<U, V>(dividends: U, divisors: V) -> [Double]
```

Returns the remainder of the double-precision elements in `dividends` divided by the elements in `divisors`, using truncating division.

```
static func remainder<U, V>(dividends: U, divisors: V) -> [Float]
```

Returns the remainder of the single-precision elements in `dividends` divided by the elements in `divisors`, using truncating division.

```
static func remainder<T, U, V>(dividends: T, divisors: U, result: inout
 V)
```

Calculates the remainder of the double-precision elements in `dividends` divided by the elements in `divisors`, using truncating division.

```
static func remainder<T, U, V>(dividends: T, divisors: U, result: inout
 V)
```

Calculates the remainder of the single-precision elements in `dividends` divided by the elements in `divisors`, using truncating division.

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

Returns the reciprocal square root of each element in a vector of double-precision values.

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

Returns the reciprocal square root of each element in a vector of single-precision values.

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

Calculates the reciprocal square root of each element in a vector of double-precision values.

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

Calculates the reciprocal square root of each element in a vector of single-precision values.

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

Returns the square root of each element in a vector of double-precision values.

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

Returns the square root each element in a vector of single-precision values.

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

Calculates the square root of each element in a vector of double-precision values.

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

Calculates the square root of each element in a vector of single-precision values.

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

Returns the integer truncation of each element in a vector of double-precision values.

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

Returns the integer truncation of each element in a vector of single-precision values.

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

Calculates the integer truncation of each element in a vector of double-precision values.

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

Calculates the integer truncation of each element in a vector of single-precision values.

```
static func truncatingRemainder<U, V>(dividends: U, divisors: V) -> [
Double]
```

Returns the remainder of the double-precision elements in `dividends` divided by the elements in `divisors`, using truncating division.

```
static func truncatingRemainder<U, V>(dividends: U, divisors: V) -> [
Float]
```

Returns the remainder of the single-precision elements in `dividends` divided by the elements in `divisors`, using truncating division.

```
static func truncatingRemainder<T, U, V>(dividends: T, divisors: U,
result: inout V)
```

Calculates the remainder of the double-precision elements in `dividends` divided by the elements in `divisors`, using truncating division.

```
static func truncatingRemainder<T, U, V>(dividends: T, divisors: U, result: inout V)
```

Calculates the remainder of the single-precision elements in `dividends` divided by the elements in `divisors`, using truncating division.

```
func vvceil(UnsafeMutablePointer<Double>, UnsafePointer<Double>, UnsafePointer<Int32>)
```

Calculates the ceiling of each element in an array of double-precision values.

```
func vvceilf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)
```

Calculates the ceiling of each element in an array of single-precision values.

```
func vvfloor(UnsafeMutablePointer<Double>, UnsafePointer<Double>, UnsafePointer<Int32>)
```

Calculates the floor of each element in an array of double-precision values.

```
func vvfloorf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)
```

Calculates the floor of each element in an array of single-precision values.

```
func vvcopysign(UnsafeMutablePointer<Double>, UnsafePointer<Double>, UnsafePointer<Double>, UnsafePointer<Int32>)
```

Copies an array, setting the sign of each element based on a second array of double-precision values.

```
func vvcopysignf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)
```

Copies an array, setting the sign of each element based on a second array of single-precision values.

```
func vvdiv(UnsafeMutablePointer<Double>, UnsafePointer<Double>, UnsafePointer<Double>, UnsafePointer<Int32>)
```

Divides each element in an array by the corresponding value in a second array of double-precision values.

```
func vvdivf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)
```

Divides each element in an array by the corresponding value in a second array of single-precision values.

```
func vvfabs(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the absolute value for each element in an array of double-precision values.

```
func vvfabsf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the absolute value for each element in an array of single-precision values.

```
func vvfmod(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Double>, UnsafePointer<Int32>)
```

Calculates the modulus after dividing each element in an array by the corresponding element in a second array of double-precision values.

```
func vvfmodf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Float>, UnsafePointer<Int32>)
```

Calculates the modulus after dividing each element in an array by the corresponding element in a second array of single-precision values.

```
func vvremainder(UnsafeMutablePointer<Double>, UnsafePointer<Double>,
UnsafePointer<Double>, UnsafePointer<Int32>)
```

Calculates the remainder after dividing each element in an array by the corresponding element in a second array of double-precision values.

```
func vvremainderf(UnsafeMutablePointer<Float>, UnsafePointer<Float>,
UnsafePointer<Float>, UnsafePointer<Int32>)
```

Calculates the remainder after dividing each element in an array by the corresponding element in a second array of single-precision values.

```
func vvint(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the integer truncation for each element in an array of double-precision values.

```
func vvintf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the integer truncation for each element in an array of single-precision values.

```
func vvnint(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the nearest integer for each element in an array of double-precision values.

```
func vvnintf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the nearest integer for each element in an array of single-precision values.

```
func vvrsqrt(UnsafeMutablePointer<Double>, UnsafePointer<Double>,
UnsafePointer<Int32>)
```

Calculates the reciprocal square root of each element in an array of double-precision values.

```
func vvrsqrtf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the reciprocal square root of each element in an array of single-precision values.

```
func vvsqrt(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the square root of each element in an array of double-precision values.

```
func vvsqrtf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the square root of each element in an array of single-precision values.

```
func vvrec(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the reciprocal of each element in an array of double-precision values.

```
func vvrecf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the reciprocal of each element in an array of single-precision values.

```
func vvnextafter(UnsafeMutablePointer<Double>, UnsafePointer<Double>,
UnsafePointer<Double>, UnsafePointer<Int32>)
```

Calculates the next machine-representable value for each element in an array of double-precision values.

```
func vvnextafterf(UnsafeMutablePointer<Float>, UnsafePointer<Float>,
UnsafePointer<Float>, UnsafePointer<Int32>)
```

Calculates the next machine-representable value for each element in an array of single-precision values.

Array-Oriented Exponential and Logarithmic Functions

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

Returns the *e*, raised to the power of each element in a vector of double-precision values.

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

Returns the *e*, raised to the power of each element in a vector of single-precision values.

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

Calculates the *e*, raised to the power of each element in a vector of double-precision values.

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

Calculates the *e*, raised to the power of each element in a vector of single-precision values.

```
static func exp2<U>(U) -> [Double]
```

Returns the 2, raised to the power of each element in a vector of double-precision values.

```
static func exp2<U>(U) -> [Float]
```

Returns the 2, raised to the power of each element in a vector of single-precision values.

```
static func exp2<U, V>(U, result: inout V)
```

Calculates the 2, raised to the power of each element in a vector of double-precision values.

```
static func exp2<U, V>(U, result: inout V)
```

Calculates the 2, raised to the power of each element in a vector of single-precision values.

```
static func expm1<U>(U) -> [Double]
```

Returns the e^x-1 for each element in a vector of double-precision values.

```
static func expm1<U>(U) -> [Float]
```

Returns the e^x-1 for each element in a vector of single-precision values.

```
static func expm1<U, V>(U, result: inout V)
```

Calculates the e^x-1 for each element in a vector of double-precision values.

```
static func expm1<U, V>(U, result: inout V)
```

Calculates the e^x-1 for each element in a vector of single-precision values.

```
static func log10<U>(U) -> [Double]
```

Returns the base 10 logarithm of each element in a vector of double-precision values.

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

Returns the natural logarithm for each element in a vector of double-precision values.

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

Returns the natural logarithm for each element in a vector of single-precision values.

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

Calculates the natural logarithm for each element in a vector of double-precision values.

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

Calculates the natural logarithm for each element in a vector of single-precision values.

```
static func log1p<U>(U) -> [Double]
```

Returns *log(1+x)* for each element in a vector of double-precision values.

```
static func log1p<U>(U) -> [Float]
```

Returns *log(1+x)* for each element in a vector of single-precision values.

```
static func log1p<U, V>(U, result: inout V)
```

Calculates *log(1+x)* for each element in a vector of double-precision values.

```
static func log1p<U, V>(U, result: inout V)
```

Calculates *log(1+x)* for each element in a vector of single-precision values.

```
static func log10<U>(U) -> [Float]
```

Returns the base 10 logarithm of each element in a vector of single-precision values.

```
static func log10<U, V>(U, result: inout V)
```

Calculates the base 10 logarithm of each element in a vector of double-precision values.

```
static func log10<U, V>(U, result: inout V)
```

Calculates the base 10 logarithm of each element in a vector of single-precision values.

```
static func log2<U>(U) -> [Double]
```

Returns the base 2 logarithm of each element in a vector of double-precision values.

```
static func log2<U>(U) -> [Float]
```

Returns the base 2 logarithm of each element in a vector of single-precision values.

```
static func log2<U, V>(U, result: inout V)
```

Calculates the base 2 logarithm of each element in a vector of double-precision values.

```
static func log2<U, V>(U, result: inout V)
```

Calculates the base 2 logarithm of each element in a vector of single-precision values.

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

Returns the unbiased exponent of each element in a vector of double-precision values.

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

Returns the unbiased exponent of each element in a vector of double-precision values.

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

Calculates the unbiased exponent of each element in a vector of double-precision values.

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

Calculates the unbiased exponent of each element in a vector of single-precision values.

```
func vvexp(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates *e* raised to the power of each element in an array of double-precision values.

```
func vvexpf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates *e* raised to the power of each element in an array of single-precision values.

```
func vvexp2(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates 2 raised to the power of each element in an array of double-precision values.

```
func vvexp2f(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates 2 raised to the power of each element in an array of single-precision values.

```
func vvexpm1(UnsafeMutablePointer<Double>, UnsafePointer<Double>,
UnsafePointer<Int32>)
```

Calculates e^x-1 for each element in an array of double-precision values.

```
func vvexpm1f(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates e^x-1 for each element in an array of single-precision values.

```
func vvlog(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the natural logarithm for each element in an array of double-precision values.

```
func vvlogf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the natural logarithm for each element in an array of single-precision values.

```
func vvlog1p(UnsafeMutablePointer<Double>, UnsafePointer<Double>,
UnsafePointer<Int32>)
```

Calculates $log(1+x)$ for each element in an array of double-precision values.

```
func vvlog1pf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates *log(1+x)* for each element in an array of single-precision values.

```
func vvlog2(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the base 2 logarithm of each element in an array of double-precision values.

```
func vvlog2f(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the base 2 logarithm of each element in an array of single-precision values.

```
func vvlog10(UnsafeMutablePointer<Double>, UnsafePointer<Double>,
UnsafePointer<Int32>)
```

Calculates the base 10 logarithm of each element in an array of double-precision values.

```
func vvlog10f(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the base 10 logarithm of each element in an array of single-precision values.

```
func vvlogb(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the unbiased exponent of each element in an array of double-precision values.

```
func vvlogbf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the unbiased exponent of each element in an array of single-precision values.

Array-Oriented Power Functions

```
static func pow<U, V>(bases: U, exponents: V) -> [Double]
```

Returns each double-precision element in the bases vector, raised to the power of the corresponding element in the exponents vector.

```
static func pow<U, V>(bases: U, exponents: V) -> [Float]
```

Returns each single-precision element in the bases vector, raised to the power of the corresponding element in the exponents vector.

```
static func pow<T, U, V>(bases: T, exponents: U, result: inout V)
```

Calculates each double-precision element in the bases vector, raised to the power of the corresponding element in the exponents vector.

`static func pow<T, U, V>(bases: T, exponents: U, result: inout V)`

Calculates each single-precision element in the bases vector, raised to the power of the corresponding element in the exponents vector.

`func vvpow(UnsafeMutablePointer<Double>, UnsafePointer<Double>, UnsafePointer<Double>, UnsafePointer<Int32>)`

Raises each element in an array to the power of the corresponding element in a second array of double-precision values.

`func vvpowf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)`

Raises each element in an array to the power of the corresponding element in a second array of single-precision values.

Array-Oriented Trigonometric Functions

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

Returns the arccosine of each element in a vector of double-precision values.

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

Returns the arccosine of each element in a vector of single-precision values.

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

Calculates the arccosine of each element in a vector of double-precision values.

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

Calculates the arccosine of each element in a vector of single-precision values.

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

Returns the arcsine of each element in a vector of double-precision values.

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

Returns the arcsine of each element in a vector of single-precision values.

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

Calculates the arcsine of each element in a vector of double-precision values.

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

Calculates the arcsine of each element in a vector of single-precision values.

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

Returns the arctangent of each element in a vector of double-precision values.

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

Returns the arctangent of each element in a vector of single-precision values.

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

Calculates the arctangent of each element in a vector of double-precision values.

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

Calculates the arctangent of each element in a vector of single-precision values.

```
static func atan2<U, V>(x: U, y: V) -> [Double]
```

Returns the arctangent of each pair of elements in two vectors of double-precision values.

```
static func atan2<U, V>(x: U, y: V) -> [Float]
```

Returns the arctangent of each pair of elements in two vectors of single-precision values.

```
static func atan2<T, U, V>(x: T, y: U, result: inout V)
```

Calculates the arctangent of each pair of elements in two vectors of double-precision values.

```
static func atan2<T, U, V>(x: T, y: U, result: inout V)
```

Calculates the arctangent of each pair of elements in two vectors of single-precision values.

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

Returns the cosine of each element in a vector of double-precision values.

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

Returns the cosine of each element in a vector of single-precision values.

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

Calculates the cosine of each element in a vector of double-precision values.

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

Calculates the cosine of each element in a vector of single-precision values.

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

Returns the cosine of pi, multiplied by each element in a vector of double-precision values.

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

Returns the cosine of pi, multiplied by each element in a vector of single-precision values.

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

Calculates the cosine of pi, multiplied by each element in a vector of double-precision values.

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

Calculates the cosine of pi, multiplied by each element in a vector of single-precision values.

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

Returns the sine of each element in a vector of double-precision values.

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

Returns the sine of each element in a vector of single-precision values.

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

Calculates the sine of each element in a vector of double-precision values.

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

Calculates the sine of each element in a vector of single-precision values.

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

Returns the sine of pi, multiplied by each element in a vector of double-precision values.

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

Returns the sine of pi, multiplied by each element in a vector of single-precision values.

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

Calculates the sine of pi, multiplied by each element in a vector of double-precision values.

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

Calculates the sine of pi, multiplied by each element in a vector of single-precision values.

```
static func sincos<T, U, V>(T, sinResult: inout U, cosResult: inout V)
```

Calculates the sine and cosine of each element in a vector of double-precision values.

```
static func sincos<T, U, V>(T, sinResult: inout U, cosResult: inout V)
```

Calculates the sine and cosine of each element in a vector of double-precision values.

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

Returns the tangent of each element in a vector of double-precision values.

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

Returns the tangent of each element in a vector of single-precision values.

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

Calculates the tangent of each element in a vector of double-precision values.

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

Calculates the tangent of each element in a vector of single-precision values.

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

Returns the tangent of pi, multiplied by each element in a vector of double-precision values.

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

Returns the tangent of pi, multiplied by each element in a vector of single-precision values.

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

Calculates the tangent of pi, multiplied by each element in a vector of double-precision values.

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

Calculates the tangent of pi, multiplied by each element in a vector of single-precision values.

```
func vvsin(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the sine of each element in an array of double-precision values.

```
func vvsinf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the sine of each element in an array of single-precision values.

```
func vvsinpi(UnsafeMutablePointer<Double>, UnsafePointer<Double>,
UnsafePointer<Int32>)
```

Calculates the sine of pi multiplied by each element in an array of double-precision values.

```
func vvsinpif(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the sine of pi multiplied by each element in an array of single-precision values.

```
func vvcos(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the cosine of each element in an array of double-precision values.

```
func vvcosf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the cosine of each element in an array of single-precision values.

```
func vvcospi(UnsafeMutablePointer<Double>, UnsafePointer<Double>,
UnsafePointer<Int32>)
```

Calculates the cosine of pi multiplied by each element in an array of double-precision values.

```
func vvcospif(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the cosine of pi multiplied by each element in an array of single-precision values.

```
func vvcosisin(OpaquePointer, UnsafePointer<Double>, UnsafePointer<
Int32>)
```

Calculates the cosine and sine of each element in an array of double-precision values.

```
func vvcosisinf(OpaquePointer, UnsafePointer<Float>, UnsafePointer<
Int32>)
```

Calculates the cosine and sine of each element in an array of single-precision values.

```
func vvsincos(UnsafeMutablePointer<Double>, UnsafeMutablePointer<Double
>, UnsafePointer<Double>, UnsafePointer<Int32>)
```

Calculates the cosine and sine of each element in an array of double-precision values.

```
func vvsincosf(UnsafeMutablePointer<Float>, UnsafeMutablePointer<Float
>, UnsafePointer<Float>, UnsafePointer<Int32>)
```

Calculates the cosine and sine of each element in an array of single-precision values.

```
func vvtan(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the tangent of each element in an array of double-precision values.

```
func vvtanf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the tangent of each element in an array of single-precision values.

```
func vvtanpi(UnsafeMutablePointer<Double>, UnsafePointer<Double>,
UnsafePointer<Int32>)
```

Calculates the tangent of pi multiplied by each element in an array of double-precision values.

```
func vvtanpif(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the tangent of pi multiplied by each element in an array of single-precision values.

```
func vvasin(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the arcsine of each element in an array of double-precision values.

`func vvasinf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)`

Calculates the arcsine of each element in an array of single-precision values.

`func vvacos(UnsafeMutablePointer<Double>, UnsafePointer<Double>, UnsafePointer<Int32>)`

Calculates the arccosine of each element in an array of double-precision values.

`func vvacosf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)`

Calculates the arccosine of each element in an array of single-precision values.

`func vvatan(UnsafeMutablePointer<Double>, UnsafePointer<Double>, UnsafePointer<Int32>)`

Calculates the arctangent of each element in an array of double-precision values.

`func vvatanf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)`

Calculates the arctangent of each element in an array of single-precision values.

`func vvatan2(UnsafeMutablePointer<Double>, UnsafePointer<Double>, UnsafePointer<Double>, UnsafePointer<Int32>)`

Calculates the arctangent of each pair of elements in two arrays of double-precision values.

`func vvatan2f(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)`

Calculates the arctangent of each pair of elements in two arrays of single-precision values.

Array-Oriented Hyperbolic Functions

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

Returns the inverse hyperbolic cosine of each element in a vector of double-precision values.

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

Returns the inverse hyperbolic cosine of each element in a vector of single-precision values.

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

Calculates the inverse hyperbolic cosine of each element in a vector of double-precision values.

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

Calculates the inverse hyperbolic cosine of each element in a vector of single-precision values.

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

Returns the inverse hyperbolic sine of each element in a vector of double-precision values.

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

Returns the inverse hyperbolic sine of each element in a vector of single-precision values.

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

Calculates the inverse hyperbolic sine of each element in a vector of double-precision values.

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

Calculates the inverse hyperbolic sine of each element in a vector of single-precision values.

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

Returns the inverse hyperbolic tangent of each element in a vector of double-precision values.

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

Returns the inverse hyperbolic tangent of each element in a vector of single-precision values.

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

Calculates the inverse hyperbolic tangent of each element in a vector of double-precision values.

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

Calculates the inverse hyperbolic tangent of each element in a vector of single-precision values.

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

Returns the hyperbolic cosine of each element in a vector of double-precision values.

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

Returns the hyperbolic cosine of each element in a vector of single-precision values.

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

Calculates the hyperbolic cosine of each element in a vector of double-precision values.

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

Calculates the hyperbolic cosine of each element in a vector of single-precision values.

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

Returns the hyperbolic sine of each element in a vector of double-precision values.

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

Returns the hyperbolic sine of each element in a vector of single-precision values.

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

Calculates the hyperbolic sine of each element in a vector of double-precision values.

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

Calculates the hyperbolic sine of each element in a vector of single-precision values.

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

Returns the hyperbolic tangent of each element in a vector of double-precision values.

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

Returns the hyperbolic tangent of each element in a vector of single-precision values.

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

Calculates the hyperbolic tangent of each element in a vector of double-precision values.

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

Calculates the hyperbolic tangent of each element in a vector of single-precision values.

```
func vvsinh(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the hyperbolic sine of each element in an array of double-precision values.

```
func vvsinhf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the hyperbolic sine of each element in an array of single-precision values.

```
func vvcosh(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the hyperbolic cosine of each element in an array of double-precision values.

```
func vvcoshf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, Unsafe
Pointer<Int32>)
```

Calculates the hyperbolic cosine of each element in an array of single-precision values.

```
func vvtanh(UnsafeMutablePointer<Double>, UnsafePointer<Double>, Unsafe
Pointer<Int32>)
```

Calculates the hyperbolic tangent of each element in an array of double-precision values.

`func vvtanhf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)`

Calculates the hyperbolic tangent of each element in an array of single-precision values.

`func vvasinh(UnsafeMutablePointer<Double>, UnsafePointer<Double>, UnsafePointer<Int32>)`

Calculates the inverse hyperbolic sine of each element in an array of double-precision values.

`func vvasinhf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)`

Calculates the inverse hyperbolic sine of each element in an array of single-precision values.

`func vvacosh(UnsafeMutablePointer<Double>, UnsafePointer<Double>, UnsafePointer<Int32>)`

Calculates the inverse hyperbolic cosine of each element in an array of double-precision values.

`func vvacoshf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)`

Calculates the inverse hyperbolic cosine of each element in an array of single-precision values.

`func vvatanh(UnsafeMutablePointer<Double>, UnsafePointer<Double>, UnsafePointer<Int32>)`

Calculates the inverse hyperbolic tangent of each element in an array of double-precision values.

`func vvatanhf(UnsafeMutablePointer<Float>, UnsafePointer<Float>, UnsafePointer<Int32>)`

Calculates the inverse hyperbolic tangent of each element in an array of single-precision values.

Data Types

`typealias COMPLEX`

`typealias DOUBLE_COMPLEX`

See Also

Vectors, Matrices, and Quaternions

Working with Vectors

Use vectors to calculate geometric values, calculate dot products and cross products, and interpolate between values.

Working with Matrices

Solve simultaneous equations and transform points in space.

Working with Quaternions

Rotate points around the surface of a sphere, and interpolate between them.

{} Rotating a cube by transforming its vertices

Rotate a cube through a series of keyframes using quaternion interpolation to transition between them.

simd

Perform computations on small vectors and matrices.