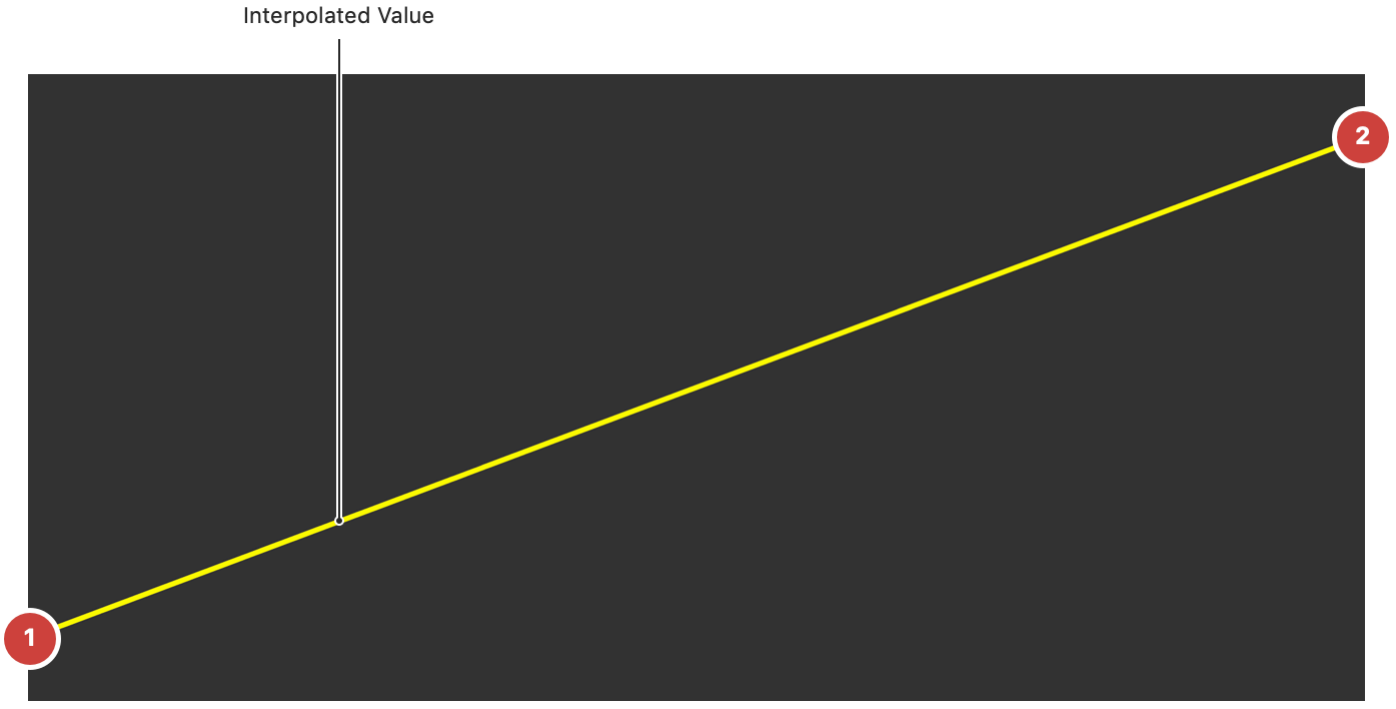
Article

# Using linear interpolation to construct new data points

Fill the gaps in arrays of numerical data using linear interpolation.

## Overview

Linear interpolation is a method of calculating intermediate data between known values by conceptually drawing a straight line between two adjacent known values. An interpolated value is any point along that line. You use linear interpolation to, for example, draw graphs or animate between keyframes. The following figure shows an example interpolated value between two points:




vDSP provides the following functions to linearly interpolate between the elements in an array:

- The single-precision function `linearInterpolate(values:atIndices:)` and the double-precision function `linearInterpolate(values:atIndices:)` that provide a simple

interface for generating interpolated data. These functions wrap `vDSP_vgenp` and `vDSP_vgenpD`, respectively.

- The single-precision function `linearInterpolate(elementsOf:using:)` and the double-precision function `linearInterpolate(elementsOf:using:)` that provide fine control over the interpolation. These functions wrap `vDSP_vlint` and `vDSP_vlintD`, respectively.

This article discusses using these functions to draw a continuous line graph based on a set of discrete data points.

## Create the data

In this example, the code interpolates the data that the `values` and `indices` arrays represent.

```
let values: [Float] = [50,  90,  55,  10,  40,  85,  65,  15,  30,   80]
let indices: [Float] = [0, 113, 227, 341, 455, 568, 682, 796, 910, 1024]
```

Each array consists of 10 elements. A pair of corresponding elements in the two arrays defines a single point in the diagram as follows:

- Elements in the `values` array denote the vertical position.
- Elements in the `indices` array denote the horizontal position.

## Define the constant that represents the number of elements

The following code defines `count` as the number of elements in the interpolation result:

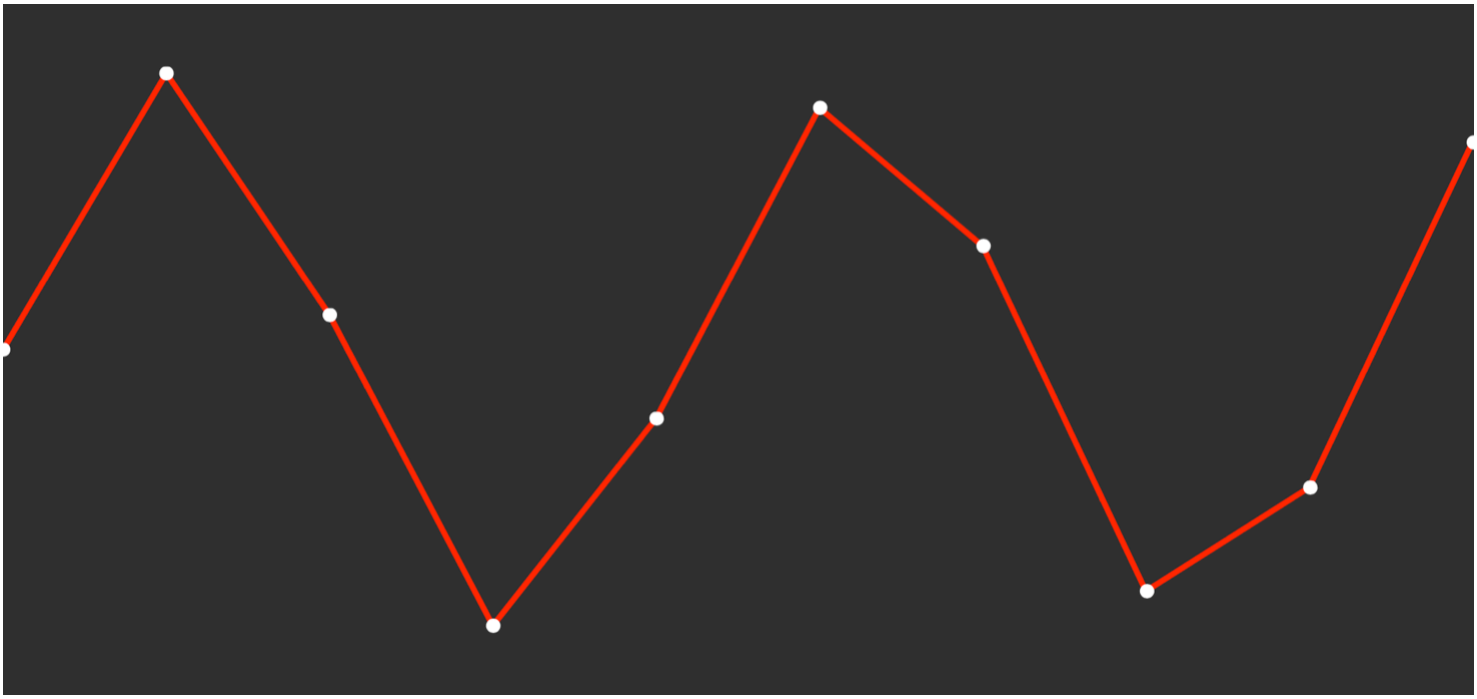
```
let count = 1024
```

## Generate a vector by interpolation

The `linearInterpolate(values:atIndices:)` function accepts the `values` and `indices` arrays and returns the interpolation result.

```
let result = vDSP.linearInterpolate(values: values,
                                    atIndices: indices)
```

On return, `result` contains the interpolated values. The following graph shows the values in `result` as a line between the known values that the `values` and `indices` arrays describe:



## Interpolate with fine control

The `linearInterpolate(elementsOf:using:)` function requires a control vector array that includes fractional parts. The fractional parts define the interpolation between the pair of values in the `values` array, starting at the index that the integer part defines.

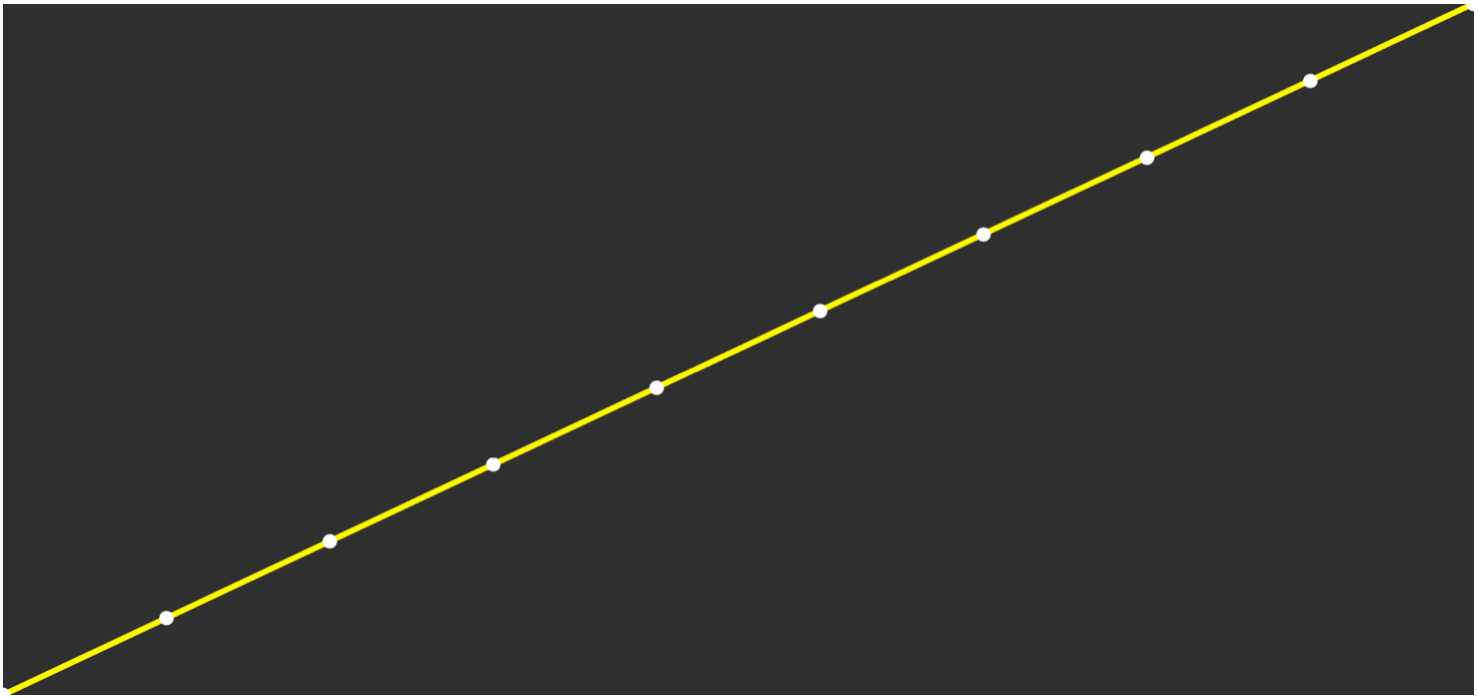
Use `ramp(in:count:)` to generate a linear ramp from `0` to the number of elements in `values`, minus 1. Note that the `ramp(in:count:)` function wraps `vDSP_vgen`.

```
let base: Float = 0
let end = Float(values.count - 1)

let control = vDSP.ramp(in: base ... end,
                        count: count)
```

On return, `control` contains the following:

```
[0] Float 0.0000

[1] Float 0.0087

[2] Float 0.0175

...

[1021] Float 8.9824

[1022] Float 8.9912

[1023] Float 9.0000
```

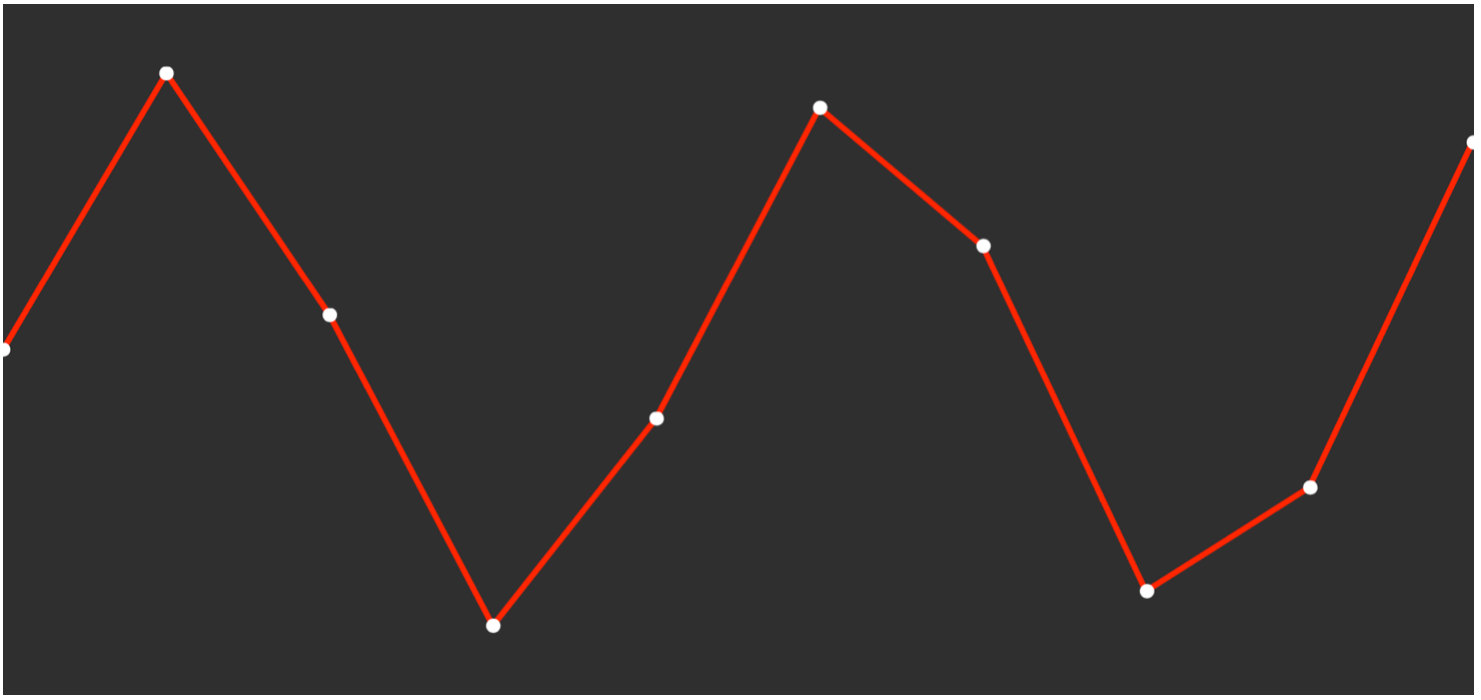
The following figure shows a visualization of the values in `control`, with small circles indicating each integer index:



The following code passes the control values to `linearInterpolate(elementsOf:using:)` to calculate the interpolated values:

```
let result = vDSP.linearInterpolate(elementsOf: values,
                                    using: control)
```

On return, `result` contains the interpolated values. The graph below shows the values in `result` as a line between the known values that the `values` and `indices` arrays describe:



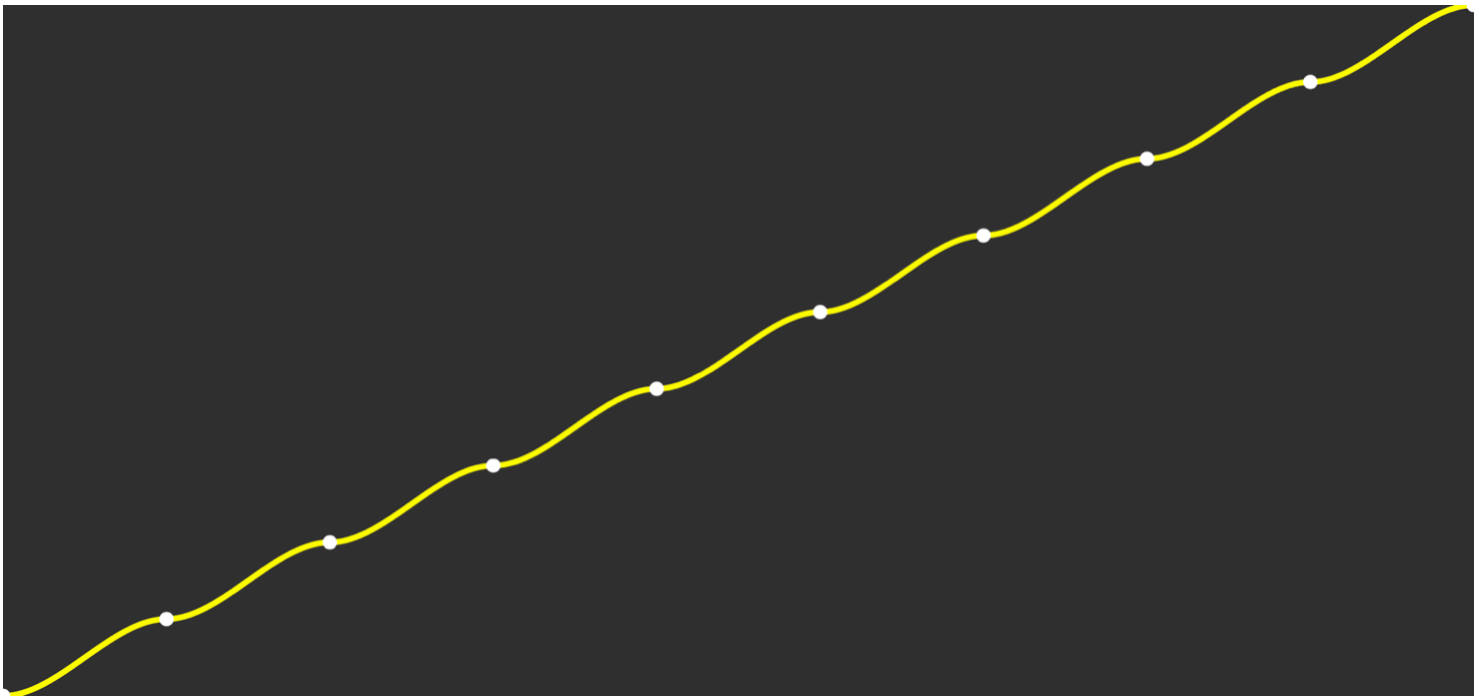
The result of `linearInterpolate(elementsOf:using:)` is equal to the result of `linearInterpolate(values:atIndices:)`.

## Add smoothing to the interpolation result

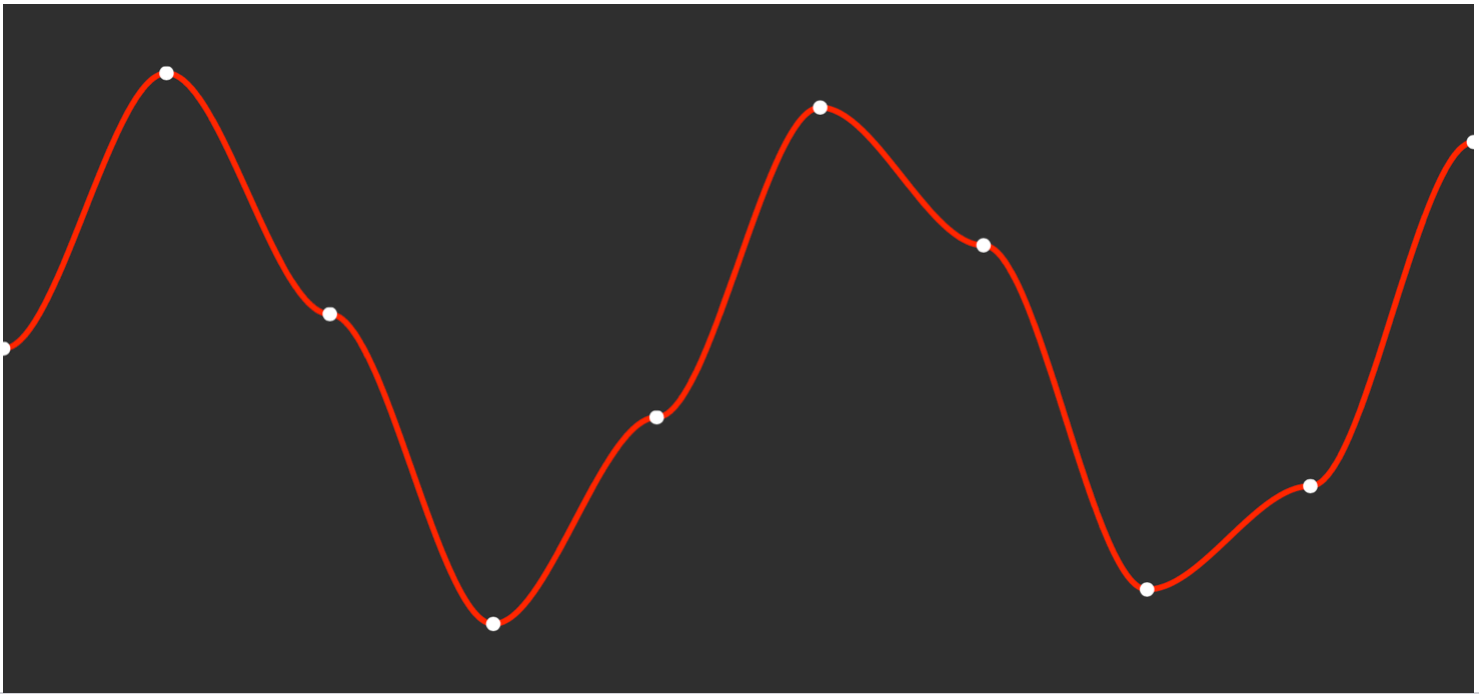
You can change the way that you generate the control array to alter the interpolated result. For example, you may want to add smoothing between the segments of a line graph or add easing to an animation. The following code uses `simd_smoothstep(_:_:_:)` to smooth the fractional parts of `control` near the increments to the integer parts:

```
let denominator = Float(n) / Float(values.count - 1)

let control: [Float] = (0 ... count).map {
    let x = Float($0) / denominator
    return floor(x) + simd_smoothstep(0, 1, simd_fract(x))
}
```

The following graph shows a visualization of the values in `control`, with small circles indicating each integer index:



Using the same call to `linearInterpolate(elementsOf:using:)` as above, the result, as shown below, shows a smoother transition between the known values.



# See Also

## Signal Processing Essentials

- Controlling vDSP operations with stride

  Operate selectively on the elements of a vector at regular intervals.

- Using vDSP for vector-based arithmetic

Increase the performance of common mathematical tasks with vDSP vector-vector and vector-scalar operations.

Resampling a signal with decimation

Reduce the sample rate of a signal by specifying a decimation factor and applying a custom antialiasing filter.

vDSP

Perform basic arithmetic operations and common digital signal processing (DSP) routines on large vectors.