Article

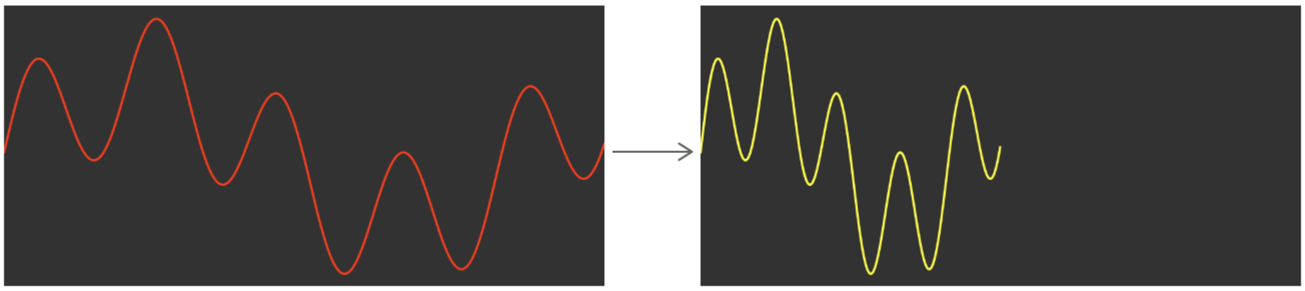
Resampling a signal with decimation

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

Overview

vDSP provides functions for decimating a signal. A decimated signal has a lower sample rate compared to its original. Decimation can be advantageous when, for example, you are transmitting a signal, creating a visual representation of a large dataset, or reducing the memory overhead when processing data.

In the following pair of images, the original signal on the left contains 1024 samples. After decimation by a factor of two, the result on the right contains 512 samples.

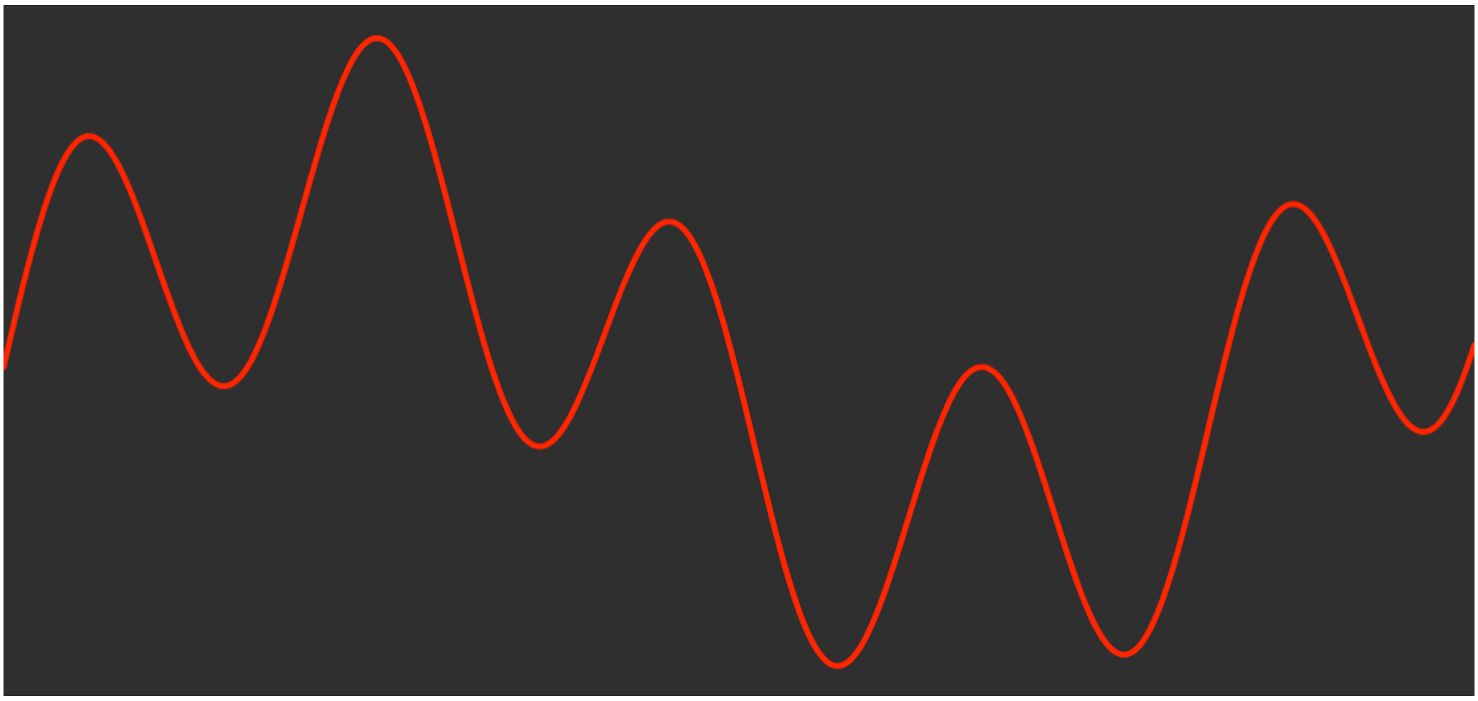


Create the input signal

The following code creates an array and populates it with a composite sine wave:

```
let inputLength = 1024
let inputSignal = (0 ..< inputLength).map {
    let x = Float($0)
    return sin(x * 0.007) + sin(x * 0.03)
}
```

The following image shows a visualization of the values in `inputSignal`:



vDSP provides the single-precision function `downsample(_:decimationFactor:filter:)` and the double-precision function `downsample(_:decimationFactor:filter:)` to decimate the elements in an array. These function wrap `vDSP_desamp` and `vDSP_desampD`, respectively.

Define the antialiasing filter

The vDSP decimation functions accept a filter that controls how adjacent samples combine. Each decimated value is the sum of the combined original values multiplied by the corresponding filter value.

The following code creates a filter that contains `[0.5, 0.5]`:

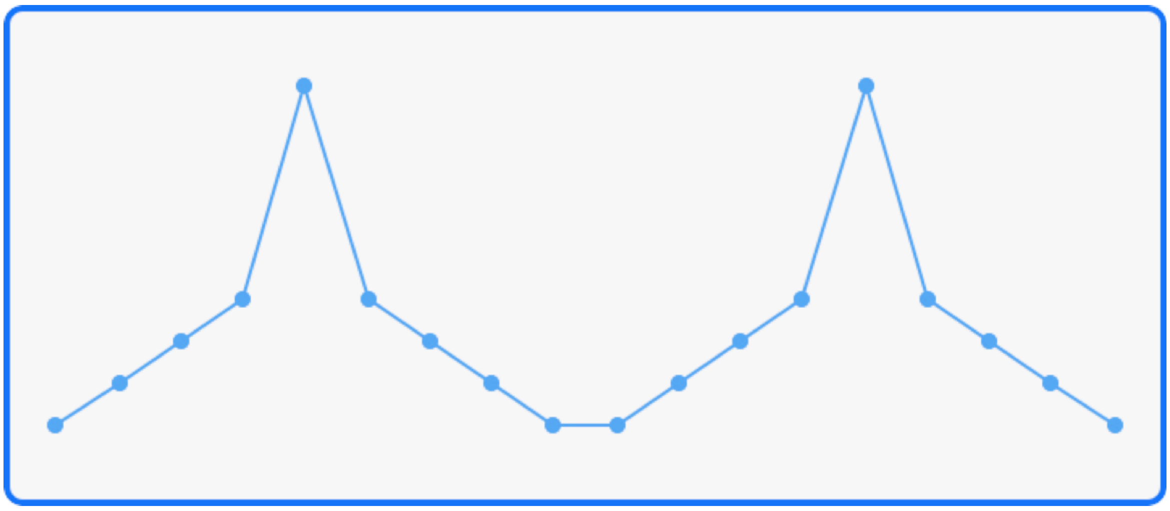
```
let filterLength = 2
let filter = [Float](repeating: 1 / Float(filterLength),
                     count: filterLength)
```

The resulting filter averages pairs of adjacent values in the original signal.

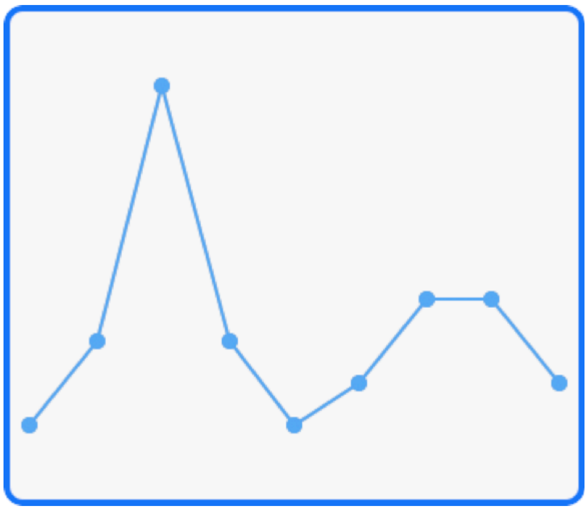
For the most complete result, set the filter length to the same value as the decimation factor, which indicates how much the original signal is decimated. For example, consider an input signal containing 18 values.

```
let originalSignal: [Float] = [10, 15, 20, 25, 50, 25, 20, 15, 10,
                               10, 15, 20, 25, 50, 25, 20, 15, 10]
```

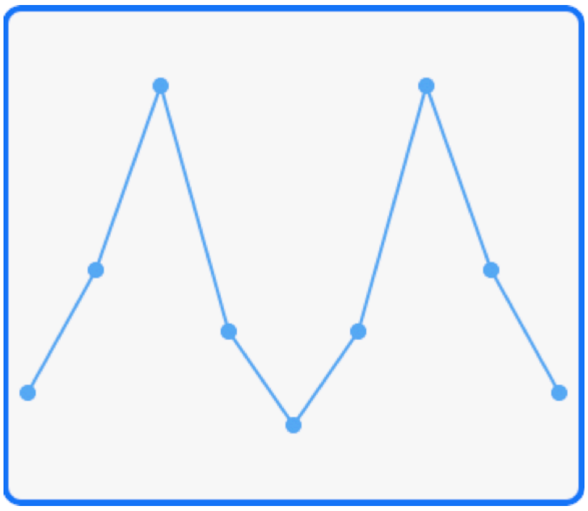
The following images visualize the original and decimated signals and illustrate the effects of different antialiasing filters. The graph below visualizes the signal.



A filter that contains a single value `[1.0]` combined with a decimation factor of 2 will sample only the even values of the original signal. The decimation functions return a result that misses the second `50` at position 13, as shown below.



However, a filter with two values, `[0.5, 0.5]` considers all values in the original signal, as illustrated below.

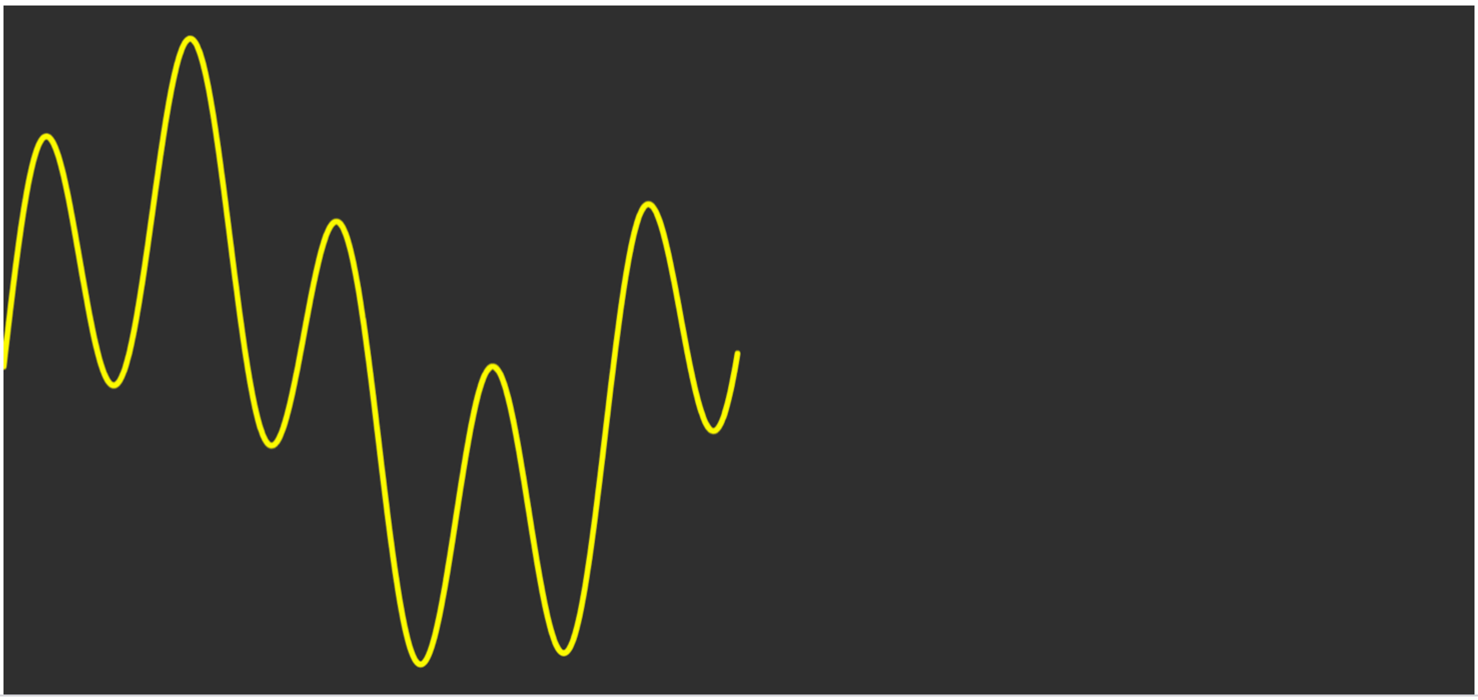


Perform the Decimation

The downsample(_:decimationFactor:filter:) function performs the decimation.

```
// The output signal contains `(source.count - filter.count) / decimationFactor + 1`
// elements.
let outputSignal = vDSP.downsample(inputSignal,
                                   decimationFactor: decimationFactor,
                                   filter: filter)
```

On return, `outputSignal` contains the result.



See Also

Signal Processing Essentials

Controlling vDSP operations with stride

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

Using linear interpolation to construct new data points

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

Using vDSP for vector-based arithmetic

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

vDSP

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