Article

Performing Fourier transforms on interleaved-complex data

Optimize discrete Fourier transform (DFT) performance with the vDSP interleaved DFT routines.

Overview

In many cases, your code performs Fourier transforms on data that originates as interleaved-complex values. An interleaved-complex representation stores the real and imaginary parts of complex values together as collections of `DSPComplex` or `DSPDoubleComplex` structures. Many Fourier-related routines in vDSP accept complex values in a split-complex representation that stores real and imaginary parts as separate collections.

For example, the following shows a collection of four complex values in a single interleaved collection:

```
let interleaved = [DSPComplex(real: real₀, imag: imag₀),
                   DSPComplex(real: real₁, imag: imag₁),
                   DSPComplex(real: real₂, imag: imag₂),
                   DSPComplex(real: real₃, imag: imag₃)]
```

The following shows the same four complex values as two collections in a split representation:

```
let reals = [real₀, real₁, real₂, real₃]
let imaginaries = [imag₀, imag₁, imag₂, imag₃]
```

vDSP routines accept split-complex values either as `DSPSplitComplex` structures or as two separate collections.

Convert interleaved values to split-complex format

Given an array `signal` that contains 32 interleaved-complex values, the following code performs a Fourier transform on the values. Use vDSP_ctoz to populate the split collections `splitSignalReal` and `splitSignalImag` with the interleaved values from `signal`:

```
let complexValuesCount = 32

let signal: [DSPComplex] = [ ... ] // `signal.count` equals `complexValuesCount`.

var splitSignalReal = [Float](repeating: 0,
                              count: complexValuesCount)
var splitSignalImag = [Float](repeating: 0,
                              count: complexValuesCount)

signal.withUnsafeBufferPointer { signalPtr in
    splitSignalReal.withUnsafeMutableBufferPointer { signalRealPtr in
        splitSignalImag.withUnsafeMutableBufferPointer { signalImagPtr in
            var splitComplex = DSPSplitComplex(realp: signalRealPtr.baseAddress!,
                                               imagp: signalImagPtr.baseAddress!)

            vDSP_ctoz(signalPtr.baseAddress!, 2,
                      &splitComplex, 1,
                      vDSP_Length(complexValuesCount))
        }
    }
}
```

Perform a Fourier transform on split data

Use the vDSP function vDSP_DFT_zop_CreateSetup to create a setup object for complex-to-complex DFTs. The execute function, vDSP_DFT_Execute, automatically switches to a fast Fourier transform (FFT) when the specified count supports the FFT algorithm.

```
var splitOutputReal = [Float](repeating: 0,
                              count: complexValuesCount)
var splitOutputImag = [Float](repeating: 0,
                              count: complexValuesCount)

if let splitComplexSetup = vDSP_DFT_zop_CreateSetup(nil,
                                                    vDSP_Length(complexValuesCount),
```

```
                                                    .FORWARD) {

    vDSP_DFT_Execute(splitComplexSetup,
                     splitSignalReal, splitSignalImag,
                     &splitOutputReal, &splitOutputImag)

    vDSP_DFT_DestroySetup(splitComplexSetup)
}


let splitComplexDominantFrequency = vDSP.indexOfMaximum(splitOutputReal)

print("Split-complex dominant frequency",
      splitComplexDominantFrequency.0,
      splitComplexDominantFrequency.1)
```

On return,`splitOutputReal` and `splitOutputImag` contain the split format frequency-domain representation of the values in `signal`. Use `indexOfMaximum(_:)` to find the dominant frequency.

Convert split-complex values to interleaved format

Use `vDSP_ztoc` to convert the split result to the interleaved format.

```
var dftOutputInterleaved = [DSPComplex](repeating: DSPComplex(),
                                        count: complexValuesCount)

splitOutputReal.withUnsafeMutableBufferPointer { dftOutputRealPtr in
    splitOutputImag.withUnsafeMutableBufferPointer { dftOutputImagPtr in
        var splitComplex = DSPSplitComplex(realp: dftOutputRealPtr.baseAddress!,
                                           imagp: dftOutputImagPtr.baseAddress!)

        vDSP_ztoc(&splitComplex, 1,
                  &dftOutputInterleaved, 2,
                  vDSP_Length(complexValuesCount))
    }
}
```

On return, `dftOutputInterleaved` contains the DFT result in the interleaved format.

Perform a Fourier transform directly on interleaved data

vDSP provides routines for DFTs directly on interleaved data. Use these functions instead of using `vDSP_ctoz` and `vDSP_ztoc` to convert between interleaved and split formats.

The following code performs the transform from the Performing Fourier transforms on interleaved-complex data section directly on the interleaved data:

```
var interleavedOutput = [DSPComplex](repeating: DSPComplex(real: 0, imag: 0),
                                     count: complexValuesCount)

if let interleavedSetup = vDSP_DFT_Interleaved_CreateSetup(nil,
                                                           vDSP_Length(complexValues
                                                           .FORWARD,
                                                           .interleaved_ComplextoCom
    vDSP_DFT_Interleaved_Execute(interleavedSetup,
                                 signal,
                                 &interleavedOutput)

    vDSP_DFT_Interleaved_DestroySetup(interleavedSetup)
}

let interleavedDominantFrequency = interleavedOutput.enumerated().max {
    a, b in a.element.real < b.element.real
}

print("Interleaved dominant frequency",
      interleavedDominantFrequency?.offset ?? -1,
      interleavedDominantFrequency?.element.real ?? 0)
```

On return, `interleavedOutput` contains the FFT result in the interleaved format.

See Also

Fourier and Cosine Transforms

- Understanding data packing for Fourier transforms
  Format source data for the vDSP Fourier functions, and interpret the results.
- Finding the component frequencies in a composite sine wave
  Use 1D fast Fourier transform to compute the frequency components of a signal.
- Reducing spectral leakage with windowing

Multiply signal data by window sequence values when performing transforms with noninteger period signals.

{} Signal extraction from noise

Use Accelerate's discrete cosine transform to remove noise from a signal.

Performing Fourier Transforms on Multiple Signals

Use Accelerate's multiple-signal fast Fourier transform (FFT) functions to transform multiple signals with a single function call.

{} Halftone descreening with 2D fast Fourier transform

Reduce or remove periodic artifacts from images.

Fast Fourier transforms

Transform vectors and matrices of temporal and spatial domain complex values to the frequency domain, and vice versa.

Discrete Fourier transforms

Transform vectors of temporal and spatial domain complex values to the frequency domain, and vice versa.

Discrete Cosine transforms

Transform vectors of temporal and spatial domain real values to the frequency domain, and vice versa.