API Collection

In-Place Functions for 2D Real FFT

Perform fast Fourier transforms in place on 2D real data.

Overview

The functions in this group use the following operation for a forward real-to-complex transform:

```
N0 = 1 << Log2N0;
N1 = 1 << Log2N1;

if (IC1 == 0) IC1 = IC0*N0/2;

scale = 2;

// Define a real matrix, h:
for (j1 = 0; j1 < N1  ; ++j1)
for (j0 = 0; j0 < N0/2; ++j0)
{
    h[j1][2*j0+0] = C->realp[j1*IC1 + j0*IC0]
              + i * C->imagp[j1*IC1 + j0*IC0];
    h[j1][2*j0+1] = C->realp[j1*IC1 + j0*IC0]
              + i * C->imagp[j1*IC1 + j0*IC0];
}

// Perform Discrete Fourier Transform.
for (k1 = 0; k1 < N1; ++k1)
for (k0 = 0; k0 < N0; ++k0)
    H[k1][k0] = scale * sum(sum(h[j1][j0]
        * e**(-Direction*2*pi*i*j0*k0/N0), 0 <= j0 < N0)
```

```
        * e**(-Direction*2*pi*i*j1*k1/N1), 0 <= j1 < N1);

// Pack special pure-real elements into output matrix:
C->realp[0*IC1][0*IC0] = H[0   ][0   ].
C->imagp[0*IC1][0*IC0] = H[0   ][N0/2]
C->realp[1*IC1][0*IC0] = H[N1/2][0   ].
C->imagp[1*IC1][0*IC0] = H[N1/2][N0/2]

// Pack two vectors into output matrix "vertically":
// (This awkward format is due to a legacy implementation.)
for (k1 = 1; k1 < N1/2; ++k1)
{
    C->realp[(2*k1+0)*IC1][0*IC0] = Re(H[k1][0   ]);
    C->realp[(2*k1+1)*IC1][0*IC0] = Im(H[k1][0   ]);
    C->imagp[(2*k1+0)*IC1][0*IC0] = Re(H[k1][N0/2]);
    C->imagp[(2*k1+1)*IC1][0*IC0] = Im(H[k1][N0/2]);
}

// Store regular elements:
for (k1 = 0; k1 < N1  ; ++k1)
for (k0 = 1; k0 < N0/2; ++k0)
{
    C->realp[k1*IC1 + k0*IC0] = Re(H[k1][k0]);
    C->imagp[k1*IC1 + k0*IC0] = Im(H[k1][k0]);
}
```

The functions in this group use the following operation for an inverse complex-to-real transform:

```
N0 = 1 << Log2N0;
N1 = 1 << Log2N1;

if (IC1 == 0) IC1 = IC0*N0/2;

scale = 1. / (N1*N0);

// Define a complex matrix, h, in multiple steps:

// Unpack the special elements:
h[0   ][0   ] = C->realp[0*IC1][0*IC0];
h[0   ][N0/2] = C->imagp[0*IC1][0*IC0];
h[N1/2][0   ] = C->realp[1*IC1][0*IC0];
```

```
h[N1/2][N0/2] = C->imagp[1*IC1][0*IC0];

// Unpack the two vectors from "vertical" storage:
for (j1 = 1; j1 < N1/2; ++j1)
{
    h[j1][0   ] = C->realp[(2*j1+0)*IC1][0*IC0]
            + i * C->realp[(2*j1+1)*IC1][0*IC0]
    h[j1][N0/2] = C->imagp[(2*j1+0)*IC1][0*IC0]
            + i * C->imagp[(2*j1+1)*IC1][0*IC0]
}

// Take regular elements:
for (j1 = 0; j1 < N1  ; ++j1)
for (j0 = 1; j0 < N0/2; ++j0)
{
    h[j1][j0   ] = C->realp[j1*IC1 + j0*IC0]
             + i * C->imagp[j1*IC1 + j0*IC0];
    h[j1][N0-j0] = conj(h[j1][j0]);
}

// Perform Discrete Fourier Transform.
for (k1 = 0; k1 < N1; ++k1)
for (k0 = 0; k0 < N0; ++k0)
    H[k1][k0] = scale * sum(sum(h[j1][j0]
        * e**(-Direction*2*pi*i*j0*k0/N0), 0 <= j0 < N0)
        * e**(-Direction*2*pi*i*j1*k1/N1), 0 <= j1 < N1);

// Store result.
for (k1 = 0; k1 < N1  ; ++k1)
for (k0 = 0; k0 < N0/2; ++k0)
{
    C->realp[k1*IC1 + k0*IC0] = Re(H[k1][2*k0+0]);
    C->imagp[k1*IC1 + k0*IC0] = Im(H[k1][2*k0+1]);
}
```

The temporary buffer versions perform the same operation but use a temporary buffer for improved performance.

Topics

In-Place FFT Functions

`vDSP_fft2d_zrip`

Computes a 2D forward or inverse in-place, single-precision real FFT.

`vDSP_fft2d_zripD`

Computes a 2D forward or inverse in-place, double-precision real FFT.

In-Place FFT Functions with Temporary Buffer

`vDSP_fft2d_zript`

Computes a 2D forward or inverse in-place, single-precision real FFT using a temporary buffer.

`vDSP_fft2d_zriptD`

Computes a 2D forward or inverse in-place, double-precision real FFT using a temporary buffer.

See Also

Functions for 2D Real FFT

Out-of-Place Functions for 2D Real FFT

Perform fast Fourier transforms out of place on 2D real data.