Subject: Contribution: Implementations of DCT and IDCT using length-N FFT
Posted by tom.grydeland on Fri, 14 Feb 2014 12:09:56 GMT
View Forum Message <> Reply to Message

Hello all,

I had use for DCTs, and since these are not provided in IDL, I have rolled my own, based on the answers found here:
http://dsp.stackexchange.com/questions/2807/fast-cosine-tran sform-via-fft
and the references therein.

The code is 1D only.

I benchmarked the different approaches and found that for shorter inputs, the choice of equivalence (length-N, length-2N pad/mirror, length 4N) didn't make much difference, but for longer inputs (> 1000) the shorter FFT would win big.

The routine DCT implements type-II DCT, while IDCT implements type-III (the common nomenclature). Scaling is the same as used in scipy.fftpack (which is *2 the scaling used in the Wikipedia article on DCT). When keyword /ORTHO is set, orthogonal weighting is used, making these routines equivalent to the (i)dct routines from That Other Vectorized Language[tm].

((The function EXPIDOUBLE, if you don't have it already, simply returns DCOMPLEX(COS(arg), SIN(arg)), which is faster than calling the complex exponential but works only for real inputs.))

I don't claim copyright on this code even if I have written it from scratch. I place it in the public domain. If anyone wants to include it in a code collection (MGUTIL, Coyote's offerings or anything else vaguely useful; or even in IDL itself), feel free. If anyone wants to extend it to multiple dimensions, it should be straightforward, and I don't want any legalese to stand in their way. An acknowledgement would be nice, though.

Regards,

Tom Grydeland, Norut AS

```
;;;;;;;;;;;;;;;;;;;;;;;; dct.pro ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; doctype='rst'

;+
;  :Author: Tom Grydeland <tom.grydeland@norut.no>
;-

;+
; Discrete cosine transform computed using FFT of length N (Makhoul)
;
; Without /ortho keyword reproduces `dct` from scipy.fftpack
;  (corresponding to 2*DCT-II from Wikipedia)
;
```

```
; With /ortho keyword reproduces `dct` from Matlab
;
;  http://dsp.stackexchange.com/questions/2807/fast-cosine-tran sform-via-fft
;
; Type 2 DCT using length N FFT
;
; Signal [a, b, c, d, e, f] becomes
;   [a, c, e, f, b, d]
; i.e. the first half of the input (including the middle point,
; for odd-length input) occupies the odd positions, while the second half,
; reversed, occupies the even positions.

;   [A, B, C, D, E, F] - j*[0, F, E, D, C, B]
; then take the real part to get the DCT
;
; :Params:
;   s: required, in, type='1D array'
;
; :Keywords:
;   ortho: optional, in, type=boolean
;     if set, use orthogonal normalization (like Matlab's DCT)
;-
function dct, s, ortho=ortho

  sdim = size(s, /dim)
  ;; For now, 1D only
  if n_elements(sdim) ne 1 then message, '1D only for now'
  N = sdim[0]

  N2 = N/2 + (N mod 2)
  stype = size(s, /type)
  u = [s[0:*:2], reverse(s[1:*:2])]

  ; du = 2*N*fft(u)

  du = 2 * N*real_part(fft(u) * expidouble(-!pi/(2*N) * dindgen(N)))

  if keyword_set(ortho) then begin
    du[0] *= sqrt(1/2.)
    du /= sqrt(2*N)
  endif

  return, du
end

;;;;;;;;;;;;;;;;;;;;; idct.pro ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; doctype='rst'
```

```
;+
;  :Author: Tom Grydeland <tom.grydeland@norut.no>
;-

;+
; Discrete cosine transform computed using FFT of length 2N and mirroring
;
; Without /ortho keyword reproduces `idct` from scipy.fftpack
;  (corresponding to 2*DCT-III from Wikipedia)
;
; With /ortho keyword reproduces `idct` from Matlab
;
;
;  http://dsp.stackexchange.com/questions/2807/fast-cosine-tran sform-via-fft
;
; Type 3 DCT using length N FFT
;
; Use DCT, [A, B, C, D, E, F], to form
;   [A, B, C, D, E, F] - j[0, F, E, D, C, B]

; take the inverse FFT of that to get
;   [a, c, e, f, b, d]
; which you rearrange to form the IDCT [a, b, c, d, e, f]
;
; :Params:
;   s: required, in, type='1D array'
;
; :Keywords:
;   ortho: optional, in, type=boolean
;     if set, use orthogonal normalization (like Matlab's IDCT)
;-
function idct, s, ortho=ortho

  sdim = size(s, /dim)
  ;; For now, 1D only
  if n_elements(sdim) ne 1 then message, '1D only for now'
  N = sdim[0]

  t = double(s)

  if keyword_set(ortho) then begin
    t[0] *= sqrt(2.)
    t /= sqrt(2*N)
  endif

  j = complex(0, 1)
  u = (t - j*reverse([t[1:*], 0])) * expidouble(!pi/(2*N) * dindgen(N))
```

```
  u = real_part(fft(u, /inverse))

  N2 = N/2 + (N mod 2)
  du = dblarr(N)

  du[0:*:2] = u[0:N2-1]
  du[1:*:2] = reverse(u[N2:*])

  return, du
end
```