Appendix: Matlab Functions For Circular Convolution and Prime Length FFTs*

Ivan Selesnick
C. Sidney Burrus

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License †

1 Programs for Reduction Operations

The reduction matrix of Equation 44 in Preliminaries[1] is implemented by `KRED` which calls `RED` . Its transpose and inverse transpose are implemented by `tRED` , `tRED` , `itKRED` and `itRED` .

```
    function x = KRED(P,E,K,x)
% x = KRED(P,E,K,x);
% P : P = [P(1),...,P(K)];
% E : E = [E(K),...,E(K)];
for i = 1:K
   a = prod(P(1:i-1).^E(1:i-1));
   c = prod(P(i+1:K).^E(i+1:K));
   p = P(i);
   e = E(i);
   for j = e-1:-1:0
      x(1:a*c*(p^(j+1))) = RED(p,a,c*(p^j),x(1:a*c*(p^(j+1))));
   end
end
```

```
    function y = RED(p,a,c,x)
% y = RED(p,a,c,x);
y = zeros(a*c*p,1);
for i = 0:c:(a-1)*c
   for j = 0:c-1
      y(i+j+1) = x(i*p+j+1);
      for k = 0:c:c*(p-2)
         y(i+j+1) = y(i+j+1) + x(i*p+j+k+c+1);
```

*Version 1.3: Sep 14, 2009 6:15 pm -0500
†http://creativecommons.org/licenses/by/2.0/
[1]"Preliminaries", (44) <http://cnx.org/content/m18132/latest/#uid20>

```
         y(i*(p-1)+j+k+a*c+1) = x(i*p+j+k+1) - x(i*p+j+c*(p-1)+1);
      end
   end
end

    function x = tKRED(P,E,K,x)
% x = tKRED(P,E,K,x);
% (transpose)
% P : P = [P(1),...,P(K)];
% E : E = [E(K),...,E(K)];
for i = K:-1:1
   a = prod(P(1:i-1).^E(1:i-1));
   c = prod(P(i+1:K).^E(i+1:K));
   p = P(i);
   e = E(i);
   for j = 0:e-1
      x(1:a*c*(p^(j+1))) = tRED(p,a,c*(p^j),x(1:a*c*(p^(j+1))));
   end
end

    function y = tRED(p,a,c,x)
% y = tRED(p,a,c,x);
% (transpose)
y = zeros(a*c*p,1);
for i = 0:c:(a-1)*c
   for j = 0:c-1
      y(i*p+j+c*(p-1)+1) = x(i+j+1);
      for k = 0:c:c*(p-2)
         y(i*p+j+k+1) = x(i+j+1) + x(i*(p-1)+j+k+a*c+1);
         y(i*p+j+c*(p-1)+1) = y(i*p+j+c*(p-1)+1) - x(i*(p-1)+j+k+a*c+1);
      end
   end
end
```

2 Programs for I ⊗ Dk ⊗ I

The operations of $I_m \otimes D_2 \otimes I_n$ and $I_m \otimes D_3 \otimes I_n$ are carried out by `ID2I` and `ID3I` . Their transposes by `ID2tI` and `ID3tI` . The functions `D2` and `D3` are listed in the appendix, 'Bilinear Forms for Linear Convolution.' Two versions of `ID2I` are listed here. One of them calls `D2` in a loop, while the other version puts the `D2` code in the loop instead of using a function call. There are several ways to implement the form $I \otimes D_2 \otimes I$. But this is a simple and straightforward method. It is modeled after `IAI` in the text.

```
    function y = ID2I(m,n,x)
y = zeros(m*n*3,1);
v = 0:n:2*n;
u = 0:n:n;
for i = 0:m-1
   for j = 0:n-1
      y(v+i*3*n+j+1) = D2(x(u+i*2*n+j+1));
   end
end
```

```
    function y = ID2I(m,n,x)
y = zeros(m*n*3,1);
for i = 0:n:n*(m-1)
   i2 = 2*i;
   i3 = 3*i;
   for j = 1:n
      j2 = i2 + j;
      j3 = i3 + j;
      y(j3) = x(j2);
      y(n+j3) = x(n+j2);
      y(2*n+j3) = x(j2) + x(n+j2);
   end
end

    function y = ID2tI(m,n,x)
y = zeros(m*n*2,1);
v = 0:n:n;
u = 0:n:2*n;
for i = 0:m-1
   for j = 0:n-1
      y(v+i*2*n+j+1) = D2t(x(u+i*3*n+j+1));
   end
end

    function y = ID3I(m,n,x)
y = zeros(m*n*5,1);
v = 0:n:4*n;
u = 0:n:2*n;
for i = 0:m-1
   for j = 0:n-1
      y(v+i*5*n+j+1) = D3(x(u+i*3*n+j+1));
   end
end

    function y = ID3tI(m,n,x)
y = zeros(m*n*3,1);
v = 0:n:2*n;
u = 0:n:4*n;
for i = 0:m-1
   for j = 0:n-1
      y(v+i*3*n+j+1) = D3t(x(u+i*5*n+j+1));
   end
end
```

References