User Manual for the paper titled 'A Wavelet Differentiation Matrix Suite'

MANI MEHRA
and
KAVITA GOYAL

1.

`cascade.m`
The function `cascade.m` computes the scaling function ϕ and the wavelet function ψ at dyadic rationals. The calling command for this function is

```
>>[x, phi, psi]=cascade(D,q);
```

where `phi` and `psi` stands for ϕ and ψ respectively. D and q are the wavelet genus and the desired dyadic resolution respectively. The implementation goes like this: An inbuilt function in Matlab, `wfilters.m`, computes low pass filter coefficients h_k and high pass filter coefficients g_k as

```
>>[h_k, g_k] = wfilters(['db' num2str(D/2)],'r');
```

Using these h_k, the matrices A_0 and A_1 are constructed such that

$$\Phi(0) = A_0\Phi(0) \text{ and } \Phi(\frac{1}{2}) = A_1\Phi(0),$$

where

$$\Phi(x) = [\phi(x)\phi(x+1)\cdots\phi(x+D-1)]^T$$

These matrices are initialized as zero matrices of order $(D-1)$ using the following Matlab command

```
>>A_0= A_1= zeros(D-1,D-1);
```

Next, the following loop of `cascade.m` constructs the matrices A_0 and A_1

```
>>E = hk(1:2:D-1);      %Even numbered low pass filter coefficients h0,h2,h4,...
>>O = hk(2:2:D);        %Odd numbered low pass filter coefficients h1,h3,h5,...
>>Firstrow = 1;         % Matrix A0
>>Lastrow  = D/2;
```

Author's address: M. Mehra (mmehra@maths.iitd.ac.in) and K. Goyal (goyalkavita9@gmail.com), Indian Institute of Technology Delhi, Hauz Khas, New Delhi-110 016

```
>>A0(Firstrow:Lastrow,1)= E;
>>for j=2:2:D-1
  >>Firstrow = Firstrow+1;
  >>Lastrow  = Lastrow+1;
  >>A0(Firstrow:Lastrow,j)   = 0;
  >>A0(Firstrow:Lastrow,j+1) = E;
>>end;
>>A0 = sqrt(2)*A0;
>>Firstrow = 1;        % Matrix A1
>>Lastrow  = D/2;
>>for j=1:2:D-3
  >>A1(Firstrow:Lastrow,j)   = 0;
  >>A1(Firstrow:Lastrow,j+1) = E;
  >>Firstrow = Firstrow+1;
  >>Lastrow = Lastrow+1;
>>end;
>>A1(Firstrow:Lastrow,D-1)   = 0;
>>A1 = sqrt(2)*A1;
```

Now we need to find out the eigenvector of A_0 corresponding to the eigenvalue 1 which is its maximum eigenvalue and this eigenvector is nothing but $\Phi(0)$. This is done by following code

```
>>[V,Dia]=eig(A0);                 % Get eigenvectors and values of matrix A0
>>[maxeig,index]=max(diag(Dia));  % Find largest eigenvalue (=1)
>>v = V(:,index);                  % Pick corresponding eigen vector
>>v = v/sum(v);                    % Normalize s.t. sum(v) = 1, (sum(\phi_{k})=1)
>>for k=0:D-2
  >>phi(k*2^q + 1) = v(k+1);      % Initialize phi at integers
>>end
```

The inbuilt function `eig.m` is used to obtain the eigenvalues and eigenvectors of A_0. After calculation of $\Phi(0)$, `cascade.m` calculate $\Phi\left(\frac{1}{2}\right)$. Last step of `cascade.m` is calculation of ψ using ϕ using following commands

```
>>psi = zeros(size(phi));
>>if q > 0                  % q is required resolution
  >>L2   = L/2;             % L=(D-1)*2^q
  >>phi2 = phi(1:2:L);       % Phi at even numerators
  >>first = 1;
  >>for k=1:D
    >>psi(first:first+L2-1) = psi(first:first+L2-1) + sqrt(2)*hp(k)*phi2;
    >>first = first + 2^q/2;
  >>end;
>>else                       % q = 0  (a special case)
  >>for m=0:D-1
```

```
    >>for k=0:D-1
      >>index = 2*m-k;
      >>if (index >= 0) & (index <= D-1)
     >>psi(m+1) = psi(m+1) + sqrt(2)*hp(k+1)*phi(index+1);
      >>end;
    >>end;
  >>end;
>>end;
```

2.

`dstmat.m`, `dst.m` and `idst.m`

- The `dstmat.m` computes the matrix $T_{r,j}$ required for discrete scaling function transformation and inverse discrete scaling function transformation.
- The `dst.m` and `idst.m` perform the required calculations for discrete and inverse discrete scaling function transformation respectively.

The function `dstmat.m` first initializes the required matrix $T_{r,j}$ using

```
>> T = spalloc(K*2^r,K*2^j,(D-1)*2^(r-j));
```

The `spalloc.m` is an inbuilt Matlab routine, `spalloc(M,N,Nz)` creates an $M \times N$ sparse matrix with all entries zero and with room to eventually hold Nz non zero elements. After initialization step, following is the main loop of `dstmat.m`

```
>>N=K*2^r;                    % Number of samples
>>P=K*2^j;                    % Number of coefficients
>>Q=2^q;                      % The resolution of phi
>>S = P*Q/N;                  % step between values needed in phi is 2^(j-r+q)
>>phi_s = phi(1:S:length(phi));
>>len   = length(phi_s);
>>if len < N                  % phi_s fits into a column
  >>for l=0:P-1
    >>firstrow = l*N/P;
    >>lastrow  = firstrow + len-1;
    >>wrap = lastrow - (N-1);
    >>if wrap > 0,
    >>lastrow = lastrow - wrap;
    >>T(0+shift:wrap-1+shift, l+shift) = phi_s(len-wrap+1:len);
  >>else
    >>wrap=0;
  >>end;
    >>T(firstrow+shift:lastrow+shift, l+shift) = phi_s(1:len-wrap);
  >>end;
>>else
  >>for l=0:P-1               % For each column
    >>firstrow = rem(l*N/P,N);
```

```
    >>for k=0:len-1         % For element in phi_s
      >>row = rem(firstrow + k, N);
      >>T(row+shift, l+shift) = T(row+shift, l+shift) + phi_s(k+shift);
    >>end;
  >>end;
>>end;
>>T=2^(j/2)*T;
```

Both dst.m and idst.m call dstmat.m using the command

```
>> T=dstmat(wavelet,r,j,q,K);
```

The calling commands for dst.m and idst.m are

```
>>c=dst(f, D);
>>f=idst(c, D, q);
```

Main commands for dst.m are:

```
>>T = dstmat(D,r,j,q,L);
>>c = T\f;
```

3.

moments.m and tmoments.m

- The function moments.m is called by the command

  ```
  >>moments = moments (h_k, pmax);
  ```

  where moments and h_k stands for M_0^p and vector containing the low pass filter coefficients respectively, pmax is the number of moments to be computed.
- The function tmoments.m computes the translated moments M_l^p.

4.

conn.m, gal_difmatrix_periodic.m and gal_diff_periodic.m

- The function conn.m implements the algorithm for the computation of the connection coefficients. The calling command for conn.m is

  ```
  >>Gamma = conn (d, D);
  ```

  where Gamma stands for $\mathbf{\Gamma^d}$, d is the order of differentiation and D is the wavelet genus. First of all it uses the Matlab inbuilt function wfilters to calculate low pass and high pass filter coefficients. Then using these low pass filter coefficients it constructs the matrix $\mathbf{A}$ of the equation

  $$(\mathbf{A} - 2^{-d}\mathbf{I})\Gamma^d = 0. \tag{1}$$

  Next it uses the following command

  ```
  >>A = A - eye(M)/2^(d);
  ```

The eye is an inbuilt function to construct the identity matrix. It then uses Matlab inbuilt function svd to find the solution of (1). The command

```
[U,S,V] = svd(A);
```

produces a diagonal matrix S, of the same size as that of A and with nonnegative diagonal elements in decreasing order, and unitary matrices U and V such that $A = USV'$. The $(2D-3)^{th}$ column of V will serve the purpose i.e. it is the required solution of (1). The normalization is done by following set of commands

```
>>if d == 0
  >>Mrow = ones (1,2D-3);
>>else
  >>Mom  = moments (hk, d);
  >>Mrow = zeros (1,2D-3);
  >>for c=1:2D-3,
    >>l       = c-(D-1);
    >>TMom    = tmoments (Mom,l);
    >>Mrow(c) = TMom(d);
  >>end
>>end;
>>c = (-1)^d * fac(d) / (Mrow * V(:,M));
>>Gamma = c * V(:,M);
```

- The function gal_difmatrix_periodic.m is the Matlab function in the suite which constructs the differentiation matrix for the Galerkin approach in periodic case. The calling command for it is

  ```
  >>difmatrix=gal_difmatrix_periodic(d,N,L,D);
  ```

  where difmatrix stands for $\mathcal{D}^{(d)}$, d is the order of differentiation, $N = 2^j$ is the size of differentiation matrix, L is the period of the function which is to be differentiated and D is the wavelet genus.
- The function gal_diff_periodic.m will differentiate a given function f using the differentiation matrix produced by gal_difmatrix_periodic.m. The calling command is

  ```
  >>[derivative, error] = gal_diff_periodic(j,d,D,L);
  ```

  where derivative and error stands for $\mathcal{D}^{(d)}f$ and $E^{(d)}(f,j)$ respectively. L is the period of the function to be differentiated. This function will ask for the function to be differentiated.

5.

cascade_der.m, collo_difmatrix_periodic.m and collo_diff_periodic.m

- The function cascade_der.m computes the values of $\phi^{(d)}$ and $\psi^{(d)}$ at dyadic rationals. The calling command for it is

  ```
  >>[x, phi_d, psi_d]=cascade_der(D,q,d);
  ```

where phi_d and psi_d stands for $\phi^{(d)}(x)$ and $\psi^{(d)}(x)$ respectively. The moments.m and tmoments.m are called by cascade_der.m.

- Matlab function collo_difmatrix_periodic.m calls cascade_der.m and then constructs the matrix $\mathcal{D}^{(d)}$ in case of collocation approach in periodic case. The calling command for collo_difmatrix_periodic.m is

```
T=collo_difmatrix_periodic(D,r,j,q,d);
```

- The function collo_diff_periodic.m will differentiate a given function f using the differentiation matrix produced by collo_difmatrix_periodic.m. The calling command for it is

```
>>[derivative, error] = collo_diff_periodic(j,D,L,d);
```

where derivative and error stands for $\mathcal{D}^{(d)}$ and $E^{(d)}(f,j)$ respectively. It will ask for a function to be differentiated and will return the value of derivative of the function at the nodal points and the error.

6.

L_daubfilt.m and R_daubfilt.m

- The function L_daubfilt.m computes the left low pass filter coefficients h^L. The calling command is:

```
>>[h_L]=L_daubfilt(k,D);
```

where h_L stands for h^L.

- The function R_daubfilt.m computes the right low pass filter coefficients h^R. The calling command is:

```
>>[h_R]=R_daubfilt(k,D);
```

where h_R stands for h^R.

7.

L_moments.m and R_moments.m

- The function L_moments.m calculates the moments of left hand side boundary scaling functions. The calling command is

```
>>[moments_Lp]=L_moments(D,p);
```

where moments_Lp stands for $m^{L,p} = \{m_k^{L,p}, \quad k = 0, \cdots, M-1\}$.

- The function R_moments.m calculates the moments of right hand side boundary scaling functions. The calling command for it is

```
>>[moments_Rp]=R_moments(D,p);
```

8.

`dstmat_nonper.m`
The function `dstmat_nonper.m` constructs the quadrature matrix C in non-periodic case of Galerkin approach. The calling command is

```
>>[C]=dstmat_nonper(D,N);
```

The `dstmat_nonper.m` calls the functions `L_momoments.m`, `R_moments.m`. The main Matlab commands used in `dstmat_nonper.m` are

```
>>ind=0;
>>st=0;
>>for k=0:N-1
    >>if(k>=0 & k<=M-1)
      >>for m=0:M-1
        >>for l=0:M-1
          >>a(m+1,l+1)=x_1(k+l+1)^(m);
        >>end
        >>[L_mo]=L_moments(D,k);
        >>b(m+1)=L_mo(m+1);
      >>end
      >>sol=a\b' ;
      >>for j=0:M-1
        >>C(k+1,j+1)=sol(j+1);
      >>end
    >>elseif((k>=M & k<=N-2*M+1))
      >>for m=0:M-1
        >>for l=0:M-1
          >>a(m+1,l+1)=l^m;
        >>end
        >>[L_mo]=mom1(D);
        >>b(m+1)=L_mo(m+1);
      >>end
      >>sol=a\b' ;
      >>for j=M+ind:M+ind+(M-1)
        >>C(k+1,j+1)=sol(j-(M+ind)+1);
      >>end
      >>ind=ind+1;
    >>else
      >>for m=0:M-1
        >>for l=0:M-1
          >>a(m+1,l+1)=x_1(N-M+l+1)^(m);
        >>end
        >>[R_mo]=R_mom(D,st);
        >>b(m+1)=R_mo(m+1);
      >>end
      >>if(st<M-1)
        >>st=st+1;
```

```
      >>end
      >>sol=a\b' ;
        >>for j=N-M:N-1
          >>C(k+1,j+1)=sol(j-(N-M)+1);
        >>end
    >>end
  >>end
```

9.

L_alpha.m and R_alpha.m

- The function L_alpha.m implements the algorithm for the calculation of $\alpha^L_{m,i}$. The calling command is

  ```
  >> [alpha_L_mi]=L_alpha(m,i,D,N);
  ```

  where alpha_L_mi stands for $\alpha^L_{m,i}$, D is wavelet genus and N is the dimension of the subspace i.e. 2^j. The functions L_firstsum_alpha.m and conn.m are called by L_alpha.m.
- The Matlab function R_alpha.m (which calls R_firstsum_alpha.m and conn.m) works on the same line as its left counterpart. The calling command is

  ```
  >> [alpha_R_mi]=R_alpha(m,i,D,N);
  ```

  where alpha_R_mi stands for $\alpha^R_{m,i}$.

10.

L_phi.m, L_phi_origin.m, L_ro.m, R_phi.m, R_phi_origin.m and R_ro.m

- The function L_phi.m computes the value of $\phi^L_k(x)$ for any x other than 0. The calling command is

  ```
  >>[phi_Lk]=L_phi(k,x,D,h,supp,phi);
  ```

  where phi_Lk stands for $\phi^L_k(x)$, and h, phi and supp are obtained using the function cascade.m as follows

  ```
  >>[y,phi,psi]=cascade(D,q);
  >>h=y(2)-y(1);
  >>supp=(D-1)*(1/h)+1;
  ```

  The functions L_daubfilt.m and L_firstsum_phi.m are called by L_phi.m.
- The function L_phi_origin.m calculates $\phi^L_k(0)$ and $\rho^L_{k,k}$ for $D = 4$ and $D = 8$. The calling command for this function is

  ```
  >>[phi_Lk0, rho_Lkk]=L_phi_origin(k,D,h,supp,phi);
  ```

  where phi_Lk0 and rho_Lkk stands for $\phi^L_k(0)$ and $\rho^L_{k,k}$ respectively. The function L_phi_origin.m for $D = 4$ uses following Matlab commands

```
>>x(1)=1/2;
>>x(2)=1;
>>y(1)=L_phi(k,x(1),D,h,supp,phi);
>>y(2)=L_phi(k,x(2),D,h,supp,phi);
>>p=polyfit(x,y,N-1);
>>x1=0;
>>y1=polyval(p,x1);
>>ro=-(y1^2)/2;
```

It is clear from above that `L_phi_origin.m` calls the function `L_phi.m` two times to calculate $\phi_k^L(1/2)$ and $\phi_k^L(1)$. Moreover, it uses the inbuilt Matlab function `polyfit` to interpolate the function $\phi_k^L(x)$ and then use the inbuilt function `polyval` to calculate $\phi_k^L(0)$. The Matlab function `polyfit` is called by the command

```
>>P = polyfit(X,Y,N);
```

It finds the coefficients of a polynomial `P(X)` of degree `N` that fits the data `Y` best in a least-squares sense. `P` is a row vector of length `N+1` containing the polynomial coefficients in descending powers. The Matlab function `polyval` is called by the command

```
>>Y = polyval(P,X);
```

It returns the value of a polynomial `P` evaluated at `X`. `P` is a vector of length `N+1` whose elements are the coefficients of the polynomial in descending powers.

- For the calculations of $\rho_{k,p}^L$ when $k \neq p$, the function `L_ro.m` is available in the suite. The calling command for it is

```
>>[rho_Lkp]=L_ro(D,L,h,supp,phi);
```

where `rho_Lkp` stands for the vector $\{\rho_{k,p}^L | k \neq p\}$. The functions `L_partialsum_ro.m`, `L_daubfilt.m`, `conn.m`, `L_phi_origin.m` and `L_alpha.m` are called by `L_ro.m`.

- The Matlab functions `R_phi.m`, `R_phi_origin.m` and `R_ro.m` are available in the suite which all work on the similar lines as their left counterparts.

11.

`gal_difmatrix_nonper.m` and `gal_diff_nonper.m`

- The function `gal_difmatrix_nonper.m` constructs the differentiation projection matrix $D^{(1)}$ in non-periodic case of Galerkin approach. The calling command is

```
>>[difmatrix]=gal_difmatrix_nonper(D,N);
```

where `difmatrix` stands for $\mathcal{D}^{(1)}$. The function `gal_difmatrix_nonper.m` has two `for` loops for the row index `l` and column index `k`. The main Matlab commands of the function `gal_difmatrix_nonper.m` are

```
L_sol=L_ro(D,L,h,supp,phi);
R_sol= R_ro(D,L,h,supp,phi);
con_1=conn(1,D);
>>for l=0:N-1
    >>for k=0:N-1
***********************************************************************
>>if(l>=0 & l<=M-1)
***********************************************************************
 >>if(k>=0 & k<=M-1)
   >>if(k==l)
     >>[y,ro]=L_phi2(k,D,h,supp,phi) ;
     >>D(l+1,k+1)=ro;
   >>else
     >>ind=(l)*M+k;
     >>D(l+1,k+1)=L_sol(ind);
   >>end
 >>elseif(k>=M & k<=N-2*M+1)
   >>D(l+1,k+1)=L_alpha(k,l,D,L);
 >>end
*************************************************************************
>>elseif(l>=M & l<=N-2*M+1)
*************************************************************************
 >>if(k>=0 & k<=M-1)
  >>D(l+1,k+1)=-L_alpha(l,k,D,L);
 >>elseif(k>=M & k<=(N-2*M+1))
  >>ind=l-k+(D-1);
  >>if(ind>=1 & ind<=2*(D-2)+1)
    >>D(l+1,k+1)=con_1(ind);
  >>end
 >>elseif(k>=(N-2*M+2) & k<=N-1)
  >>if(-(l-(N-2*M+1) )>=0 & -(l-(N-2*M+1))<=M-1...
       & -(k-(N-1))>=0 & -(k-(N-1))<=M-1)
   >>D(l+1,k+1)=-R_alpha(-(l-(N-1)),-(k-(N-1)),D,L);
  >>else
   >>D(l+1,k+1)=0;
  >>end
 >>end
*************************************************************************
>>elseif(l>=(N-2*M+2) & l<=N-1)
*************************************************************************
  >>if(k>=M & k<=N-2*M+1)
   >>if(-(l-(N-1) )>=0 & -(l-(N-1))<=M-1 & -(k-(N-2*M+1))>=0...
       & -(k-(N-2*M+1))<=M-1)
    >>D(l+1,k+1)=R_alpha(-(k-(N-1)),-(l-(N-1)),D,L);
   >>else
    >>D(l+1,k+1)=0;
```

```
   >>end
  >>elseif(k>=(N-2*M+2) & k<=N-1)
   >>if(k==l)
    >>[y,ro]=R_phi_origin(-(k-(N-1)),D,h,supp,phi);
    >>D(l+1,k+1)=ro;
   >>else
    >>ind=(l-(N-2*M+2))*(2*M-2)+(k-(N-2*M+2));
    >>if(ind<=length(R_sol))
     >>D(l+1,k+1)=R_sol(ind);
    >>end
   >>end
  >>end
****************************************************************************
>>end
****************************************************************************
    >>end
>>end
```

- The function gal_diff_nonper.m is included in our suite to construct the differentiation matrix $\mathcal{D}^{(1)} = C^{-1}D^{(1)}C$ in the non-periodic case of Galerkin approach. The calling command for it is

  `>>[difmatrix]=gal_diff_nonper(D,N);`

  where difmatrix stands for $\mathcal{D}^{(1)}$. The Matlab commands for gal_diff_nonper.m are

```
function [difmatrix]=gal_diff_nonper(D,N)
D_matrix=gal_difmatrix_nonper(D,N);
C=dstmat_nonper(D,N);
difmatrix=inv(C)*D_matrix*C;
```

12.

Bn_spline.m
The calling command for the function Bn_spline.m is

`>>[beta_nxk]=Bn_spline(x,n,k);`

where beta_nxk stands for $\beta^n(x-k)$. The code is

```
>>function [beta]=Bn_spline(x,n,k)
>>x=x-k;
>>sum=0;
>>for j=0:n+1
      >>if(j~=0)
          >>combos=combntns(1:n+1,j);
          >>tem=size(combos,1);
```

```
        >>else
            >>tem=1;
        >>end
     >>sum=sum+(((-1)^j)/factorial(n))*tem*pow_fun(n,x+((n+1)/2)-j);
>>end
>>beta=sum;
>>beta=beta*factorial(n);
```

This function evaluates the value of $\beta^n(x-k)$. The function `pow_fun.m` is available in the suite which computes $[x]^n_+ = \max\{0,x\}^n$.