TEXAS TECH UNIVERSITY

David S. Gilliam
Department of Mathematics
Texas Tech University
Lubbock, TX 79409

806 742-2566
gilliam@texas.math.ttu.edu
http://texas.math.ttu.edu/~gilliam

Mathematics 5342
Discrete Fourier Transform

1 Introductory Remarks

There are many ways that the Discrete Fourier Transform (DFT) arises in practice but generally one somehow arrives at a periodic sequence numbers. These numbers may arise, for example, as a discretely sampled values of an analog function sampled over some period window and then extended periodically. They may also arise as a discrete set of values from the measurements in an experiment. Once again we would assume that they are extended periodically. In any case, the DFT of the sequence is a new periodic sequence and is related to the original sequence via a DFT inversion transform similar to the Inverse Fourier Transform. The DFT turns out to be very useful in spectral analysis of both Fourier series and Fourier transforms.

Unfortunately, there are many different definitions of the DFT just as there are for the Fourier transform. They are all related but the difference makes reading different books a bit of a chore. In these notes we will adopt the definition used in the Matlab software since Matlab will use for the most part in the numerical examples.

It is common to assume that we are given a periodic sequence of numbers $\{f_k\}_{k=0}^{N-1}$ of period N. Then the DFT of the sequence is a sequence F_n for $n = 0, \cdots, (N-1)$ defined by

$$F_n = \sum_{k=0}^{N-1} f_k \, e^{-2\pi i \, nk/N}, \text{ where } i = \sqrt{-1}. \tag{1}$$

But in Matlab you cannot use a zero or negative indices, so the sequences are $\{f_k\}_{k=1}^N$ and the DFT is computed as

$$F_n = \sum_{k=1}^{N} f_k \, e^{-2\pi i \, (n-1)(k-1)/N}, \; n = 1, 2, \cdots, N.$$

In this case we have the formula for the Inverse Discrete Fourier Transform (IDFT) which gives

$$f_k = \sum_{n=1}^{N} F_n \, e^{2\pi i \, (n-1)(k-1)/N}, \; k = 1, 2, \cdots, N.$$

We can also express the DFT and IDFT in terms of matrix multiplication as follows: Let $\vec{f} = \begin{bmatrix} f_1 & \cdots & f_N \end{bmatrix}^T$ and $\vec{F} = \begin{bmatrix} F_1 & \cdots & F_N \end{bmatrix}^T$. If we now denote the primitive N-th root of unity by $w_N = \exp(-2\pi i/N)$ and the $N \times N$ matrix $\mathcal{F}_N$ with $k\,n$th entry

$$[\mathcal{F}_N]_{k,n} = \left[w_N^{k\,n}\right]_{k,n}.$$

Then the DFT is

$$\vec{F} = \mathcal{F}_N \vec{f}$$

and the IDFT is

$$\vec{f} = \frac{1}{N} \mathcal{F}_N \vec{F}.$$

2 Relation to Fourier Transform

This definition is related to the FT of a causal signal $f(t)$, which is effectively zero for $t > T$, as follows:

1. For window size T and number of sampling points N define the *sample spacing* $T_s = \dfrac{T}{N}$.
2. Now define the *sample points* $t_k = kT_s$ for $k = 0, \cdots, (N-1)$, (i.e., $t_k = k\dfrac{T}{N}$).
3. Then we define $f_k = f(t_k)$.
4. Associated with this we define the frequency sampling points $w_n \equiv \dfrac{2\pi n}{T}$ where the number $\frac{2\pi}{T}$ is the fundamental frequency.

   Now we consider the problem of approximating the FT of f at the points $w_n = \dfrac{2\pi n}{T}$. The exact answer is

$$F(w_n) = \int_{-\infty}^{\infty} e^{-iw_n t} \, f(t) \, dt, \; n = 0, \cdots, (N-1).$$

5. For $f \sim 0$ for $|x| > T$

$$F(w_n) \approx \int_0^T e^{-iw_n t} f(t)\, dt,\ n = 0, \cdots, (N-1).$$

6. Let us approximate this integral by a left-endpoint Riemann sum approximation using the points t_k defined above:

$$F(w_n) \approx T_s \sum_{k=0}^{N-1} e^{-iw_n t_k} f(t_k)\ n = 0, \cdots, (N-1).$$

7. Substituting the definitions of w_n, t_k and T_s in terms of T and N we have

$$F(w_n) \approx \frac{T}{N} \sum_{k=0}^{N-1} e^{-2\pi ink/N} f(t_k)\ n = 0, \cdots, (N-1).$$

8. Thus we see that

$$F\left(\frac{2\pi n}{T}\right) \approx \frac{T}{N} F_n$$

where

$$F_n = \sum_{k=1}^{N} f_k\, e^{-2\pi i\,(n-1)(k-1)/N},\ n = 1, 2, \cdots, N$$

is the DFT defined above.

Thus we see the relationship between the FT and the DFT.

The above definition of the DFT is good for signal processing where we assume the the original analog signal is causal, i.e., $f(t) = 0$ for $t < 0$. Note is is not good for noncausal signals in which case one often uses the following definition of the DFT. Suppose that f is defined on $-T/2, \cdots, T/2$, or more realistically, $f \sim 0$ for $|t| > T/2$. Then the DFT is usually defined as a right hand endpoint Riemann sum approximation to the FT with $t_k = k\dfrac{T}{N}$, $k = -N/2+1, \cdots, N/2$ evaluated at the points $w_n = \dfrac{2\pi n}{T}$, $n = -N/2+1, \cdots, N/2$:

$$F\left(\frac{2\pi n}{T}\right) \approx \frac{T}{N} F_n$$

where

$$\widetilde{F}_n = \sum_{k=-N/2+1}^{N/2} f_k\, e^{-2\pi i\, nk/N},\ n = -N/2+1, \cdots, N/2$$

Lets consider an example taken from [3]. Let

$$f(t) = \begin{cases} 12\exp(-3t), & t \geq 0 \\ 0, & t < 0 \end{cases}$$

and let us denote the Fourier Transform (FT) of f by $F(w)$. Then $F(w) = \dfrac{12}{(3+iw)}$.

Let us assume that $f \sim 0$ for $t > 3$ and let $T = 3$. Do the following in Matlab

```
N=128;
t=linspace(0,3,N);
f=12*exp(-3*t);
Ts=t(2)-t(1);
Ws=2*pi/Ts;  % sampling frequency in rad/sec
F=fft(f);  % use matlab to compute the fft
% now compute the postive frequency components and mult by
% Ts=T/N to approximate F(w)
Fp=F(1:N/2+1)*Ts ;
% build the frequency axis from zero the the Nyquist frequency Ws/2
W=Ws*(0:N/2)/N;
F_exact = 12./(3+i*W);  % evaluate the exact FT of f
% now plot the results
plot(W,abs(F_exact),W,abs(Fp),'+')
xlabel('Frequency, Rad/sec')
ylabel('|F(w)|')
```

3 Relation to Fourier Series

Recall that if f is a T periodic function defined initially on $[-T/2, T/2]$ then

$$f \sim \sum_{-\infty}^{\infty} c_n \, e^{2\pi i n/T}$$

where

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} f(x) \, e^{-2\pi i n x/T} \, dx.$$

Now from what we have written above, for the approximation of the FT of a function which is zero outside the interval $[-T/2, T/2]$, we see that

$$c_n \approx \frac{1}{T}\frac{T}{N} \sum_{k=-N/2+1}^{N/2} f_k \, e^{-2\pi i \, nk/N}, \; n = -N/2+1, \cdots, N/2$$

Thus by our definition of the DFT we have

$$c_n = \frac{1}{N} F_n.$$

We also would like to use the Matlab version of the DFT to approximate the Fourier series coefficients. Also it is important to note that the matrix method is useless from a practical point of view since at best it is an algorithm that requires calculations on the order of N^2 operations. Rather, for $N = 2^\ell$, for a positive integer ℓ, there is a procedure for evaluating the DFT called the Fast Fourier Transform (FFT) which takes advantage of many symmetries in the matrix $\mathcal{F}_N$. This method, which can be evaluated recursively, is of the order $N \log(N)$.

Here is an example of how it works.

```
N=8;  % take N even, in fact N = 2^k
FN=F_N(N);
x=(1:N)';  % this is just  some sample data
% now rearrange the columns of F_N by putting
% the odd columns first. This can be done as follows
cols=[1:2:(N-1) 2:2:N]
FF_N=FN(:,cols);
xx=x(cols);
% then F_N x = FF_N xx
D=diag(exp(-2*pi*i*(0:(N/2-1))/N));
% then we compare the 4 (N/2 X N/2) matrices in FF_N
FF1=FF_N(1:N/2,1:N/2);
FF2=FF_N(1:N/2,(N/2+1):N);
FF3=FF_N((N/2+1):N,1:N/2);
FF4=FF_N((N/2+1):N,(N/2+1):N);

abs(F_N(N/2)-FF1)
abs(D*F_N(N/2)-FF2)
abs(F_N(N/2)-FF3)
abs(-D*F_N(N/2)-FF4)


[eye(N/2) D;eye(N/2) -D]*[F_N(4)*x(1:2:N);F_N(4)*x(2:2:N)]

F_N(8)*x
```

The point is that the high order 8×8 matrix multiplication can be replaced by several 4×4 matrix multiplications. Thus, for example, if $N = 4096$, the number of multiplications are reduced from 16 million to less than $50,000$.

The recursive FFT is given by the following simple Matlab code which must be saved as "fftrecur.m" so it can be used.

```
function y=fftrecur(x)
n=length(x);
if n==1
y=x;
else
m=n/2;
yT=fftrecur(x(1:2:n));
yB=fftrecur(x(2:2:n));
d=exp(-2*pi*sqrt(-1)/n).^(0:(m-1))';
z=d.*yB;
y=[yT+z;yT-z];
end
```

On the other hand one might as well use the builtin Matlab code "fft".

The following program, which must be saved in an m-file and then excuted, allows you to compare the matrix method with the Matlab builtin fft and uses these to compute the Fourier series approximation on an interval $[-A/2, A/2]$ for a function f. The three examples are:

1. $f(x) = \sin(x)$
2. $f(x) = x^2$
3. $f(x) =$ abs(x)

The program also computes the real version of the Fourier series representation in terms of sines and cosines.

```
clear

A=input('period T = ');
disp(' ')
disp('function defined on -T/2 to T/2')
```

```
disp(' ')
power=input('Input  an integer j<6 so that N = 2^power,  power = ');
disp(' ')
N=2^power;
ind=1:N;
x_vec=(ind-N/2)*A/N;
I=(ind-N/2);
x=linspace(-T/2,T/2,50);

%%%%%%%%%% Here are three different Functions %%%%%%%%%%
%----- prob 1
% f_exact=sin(x);
%  f_vec=sin(x_vec);

%----- prob 2
% f_exact=x.^2;
% f_vec=x_vec.^2;

%----- prob 3
 f_exact=abs(x);
 f_vec=abs(x_vec);

%%%%%%%%%%
% use matrix multiply
flops(0);
tic
M=(1/N)*exp(-2*pi*i*I'*I/N);
c_vec=M*f_vec';
f_approx=real(c_vec.'*exp(2*pi*i*I'*I/N));
t_matrix=toc
matrix_flops=flops
figure
plot(x,f_exact,x_vec,f_approx,'-o')

nf_vec=[f_vec((N/2):N) f_vec(1:(N/2-1))];
```

```
% use matlab fft
flops(0)
tic
c_ml=fft(nf_vec).*exp(2*pi*i*((-N/2+1):N/2)*(N/2-1));
I_ml=(0:(N-1));

dt=A/N;
f_approx_ml=N^(-1)*real(c_ml*exp(2*pi*i*(I_ml)'*x_vec/T));
t_ml=toc
ml_flops=flops
figure
plot(x,f_exact,x_vec,f_approx_ml,'-o')

%Now use sines and cosines
newc=fft(nf_vec);
a0=newc(1)/N;
a=2*real(newc(2:(N/2+1)))/N;
b=2*imag(newc(2:(N/2+1)))/N;
dt=T/N;
fac=2*pi*(1:(N/2))'*x_vec/(N*dt);
newf_approx=a0+(a*cos(fac)+b*sin(fac));
figure
plot(x,f_exact,x_vec,newf_approx,'-o')
```

The next example requires several files from the book [3]. I have included one of them, "fswindow.m", here to give you an idea what they look like. This function file generates a "window vector" used to do filtering, for example, to remove effects of Gibb's phenomena in Fourier series approximation of discontinuous functions.

```
function w = fswindow(N,wt,a)
%FSWINDOW Generate Window Functions.
% FSWINDOW(N,TYPE) creates a window vector of type TYPE having
% a length equal to the scalar N.
% FSWINDOW(X,TYPE) creates a window vector of type TYPE having
% a length and orientation the same as the vector X.
% FSWINDOW(.,TYPE,alpha) provides a parameter alpha as required
```

```
% for some window types.
% FSWINDOW with no input arguments returns a string matrix whose
% i-th row is the i-th TYPE given below.
%
% TYPE is a string designating the window type desired:
% 'rec' = Rectangular or Boxcar
% 'tri' = Triangular or Bartlet
% 'han' = Hann or Hanning
% 'ham' = Hamming
% 'bla' = Blackman common coefs.
% 'blx' = Blackman exact coefs.
% 'rie' = Riemann {sin(x)/x}
% 'tuk' = Tukey,  0< alpha < 1; alpha = 0.5 is default
% 'poi' = Poisson, 0< alpha < inf; alpha = 1 is default
% 'cau' = Cauchy, 1< alpha < inf; alpha = 1 is default
% 'gau' = Gaussian, 1< alpha < inf; alpha = 1 is default
%
% Reference: F.J. Harris,"On the Use of Windows for Harmonic Analy-
sis with the
% Discrete Fourier Transform," IEEE Proc., vol 66, no 1, Jan 1978, pp 51-
83.

% D.C. Hanselman, University of Maine, Orono, ME  04469
% 1/10/95
% Copyright (c) 1996 by Prentice-Hall, Inc.

if nargin==0
w=['rec';'tri';'han';'ham';'bla';'blx';'rie';'tuk';'poi';'cau';'gau'];
return
end
[r,c]=size(N);
n=max(r,c);
if n>1,N=n;end
wt=[wt(:)' '   '];
wt=wt(1:3);
if strcmp(wt,'rec') % rectangular
```

```
w = ones(1,N);
elseif strcmp(wt,’tri’) % triangular
m=(N-1)/2;
w=(0:m)/m;
w=[w w(ceil(m):-1:1)];
elseif strcmp(wt,’han’) % hanning
m=linspace(-pi,pi,N);
w=0.5*(1 + cos(m));
elseif strcmp(wt,’ham’) % hamming
m=linspace(-pi,pi,N);
w=.54 + .46*cos(m);
elseif strcmp(wt,’bla’) % blackman
m=linspace(-pi,pi,N);
w=.42 +.5*cos(m) + .08*cos(2*m);
elseif strcmp(wt,’blx’) % blackman exact
m=linspace(-pi,pi,N);
w=(7938 + 9240*cos(m) + 1430*cos(2*m))/18608;
elseif strcmp(wt,’rie’) % riemann
m=linspace(-pi,pi,N)+eps;
w=sin(m)./(m);
elseif strcmp(wt,’tuk’) % tukey
if nargin<3,a=0.5;end
m=fix((a+1)*N/2);
w=ones(1,N);
if a>0&a<1
w(m:N)=0.5*(1 + cos(pi*((m:N)-m)/(N-m)));
w(1:N-m+1)=w(N:-1:m);
end
elseif strcmp(wt,’poi’) % poisson
if nargin<3,a=1;end
m=round(N/2)-1;
w=[(m:-1:0) (rem(N,2):m)]/m;
w=exp(-abs(a)*w);
elseif strcmp(wt,’cau’) % cauchy
if nargin<3,a=1;end
m=round(N/2)-1;
```

```
w=[(m:-1:0) (rem(N,2):m)]/m;
w=(1+(a*w).^2).^(-1);
elseif strcmp(wt,'gau') % gaussian
if nargin<3,a=1;end
m=round(N/2)-1;
w=[(m:-1:0) (rem(N,2):m)]/m;
w=exp(-0.5*(a*w).^2);
else
error('Incorrect Window type requested');
end
if r>1,w=w.';end
```

There are several other mfiles that we need from [3].

1. fseval.m

```
function y=fseval(kn,t,wo)
%FSEVAL Fourier Series Function Evaluation.
% FSEVAL(Kn,t,Wo) computes values of a real valued function given
% its complex exponential Fourier series coefficients Kn, at the
% points given in t where the fundamental frequency is Wo rad/s.
% K contains the Fourier coefficients in ascending order:
% Kn = [k   k     ... k  ...  k    k ]
%        -N  -N+1      0       N-1  N
% if Wo is not given, Wo=1 is assumed.
% Note: this function creates a matrix of size:
% rows = length(t) and columns = (length(K)-1)/2

% D. Hanselman, University of Maine, Orono, ME  04469
% 2/9/95
% Copyright (c) 1996 by Prentice Hall, Inc.

if nargin==2,wo=1;end
nk=length(kn);
if rem(nk-1,2)|(nk==1)
 error('Number of elements in K must be odd and greater than 1')
end
```

```
n=0.5*(nk-1); % highest harmonic
nwo=wo*(1:n); % harmonic indices
ko=kn(n+1); % average value
kn=kn(n+2:nk).'; % positive frequency coefs
y=ko+2*(real(exp(j*t(:)*nwo)*kn))';
```

2. fsfind.m

```
function [Fn,nwo,f,t]=fsfind(fun,T,N,P)
%FSFIND Find Fourier Series Approximation.
% Fn=FSFIND(FUN,T,N) computes the Complex Exponential
% Fourier Series of a signal described by the function 'FUN'.
% FUN is the character string name of a user created M-file function.
% The function is called as f=FUN(t) where t is a vector over
% the range 0<=t<=T.
%
% The FFT is used. Choose sufficient harmonics to minimize aliasing.
%
% T is the period of the function. N is the number of harmonics.
% Fn is the vector of FS coefficients.
%
% [Fn,nWo,f,t]=FSFIND(FUN,T,N) returns the frequencies associated
% with Fn in nWo, and returns values of the function FUN
% in f evaluated at the points in t over the range 0<=t<=T.
%
% FSFIND(FUN,T,N,P) passes the data in P to the function FUN as
% f=FUN(t,P). This allows parameters to be passed to FUN.

% D.C. Hanselman, University of Maine, Orono, ME 04469
% 2/9/95
% Copyright (c) 1996 by Prentice-Hall, Inc.

n=2*N;
t=linspace(0,T,n+1);
if nargin==3
 f=feval(fun,t);
else
```

```
 f=feval(fun,t,P);
end
Fn=fft(f(1:n));
Fn=[0 conj(Fn(N:-1:2)) Fn(1:N) 0]/n;
nwo=2*pi/T*(-N:N);
```

3. fsform.m

```
function [a,b,ao]=fsform(c,d,co)
%FSFORM Fourier Series Format Conversion.
% Kn=FSFORM(An,Bn,Ao) converts the trigonometric FS with
% An being the COSINE and Bn being the SINE coefficients to
% the complex exponential FS with coefficients Kn.
% Ao is the DC component and An, Bn and Ao are assumed to be real.
%
% [Kn,i]=FSFORM(An,Bn,Ao) returns the index vector i that
% identifies the harmonic number of each element of Kn.
%
% [An,Bn,Ao]=FSFORM(Kn) does the reverse format conversion.

% D.C. Hanselman, University of Maine, Orono, ME 04469
% 1/12/95
% Copyright (c) 1996 by Prentice-Hall, Inc.

nc=length(c);
if nargin==1 % complex exp --> trig form
 if rem(nc-1,2)|(nc==1)
  error('Number of elements in K must be odd and greater than 1')
 end
 nn=(nc+3)/2;
 a=2*real(c(nn:nc));
 b=-2*imag(c(nn:nc));
 ao=real(c(nn-1));
elseif nargin==3 % trig form --> complex exp form
 nd=length(d);
 if nc~=nd,
  error('A and B must be the same length')
```

```
 end
 a=0.5*(c-sqrt(-1)*d);
 a=[conj(a(nc:-1:1)) co(1) a];
 b=-nc:nc;
else
 error('Improper number of input arguments.')
end
```

4. fsround.m

```
function K=fsround(kn,tol)
%FSROUND Round Fourier Series Coefficients.
% FSROUND(Kn) rounds the Fourier Series coefficients Kn
% to eliminate residual terms. Terms satisfying
% abs(Kn)<TOL*max(abs(Kn))        %terms of small magnitude, or
% abs(real(Kn))<TOL*abs(imag(Kn)) %terms with small real part, or
% abs(imag(Kn))<TOL*abs(real(Kn)) %terms with small imag part
% are set to zero. TOL is set equal to sqrt(eps) or can be
% can be specified by FSROUND(Kn,TOL)

% D.C. Hanselman, University of Maine, Orono, ME 04469
% 1/2/95
% Copyright (c) 1996 by Prentice-Hall, Inc.

if nargin<2,tol=sqrt(eps);end
K=kn(:).';
mkn=abs(kn);
rkn=abs(real(kn));
ikn=abs(imag(kn));

i=find(mkn<tol*max(mkn));K(i)=zeros(1,length(i));
i=find(rkn<tol*ikn);     K(i)=zeros(1,length(i));
i=find(ikn<tol*rkn);     K(i)=zeros(1,length(i));
```

5. sawtooth.m

```
function f=sawtooth(t,T)
%SAWTOOTH Sawtooth Waveform Generation.
% SAWTOOTH(t,T) computes values of a sawtooth having a
% period T at the values in t.
%
% Used in Fourier series example.

f=10*rem(t,T)/T;
```

Here is an example using these mfiles on a sawtooth function. First let us plot a sawtooth graph that repeats the tooth every $.2$ units and use 100 points to plot two teeth.

```
T=.2;
t=linspace(0,.4);
y=sawtooth(t,T);
plot(t,y)
```

Now we will use "fsfind.m" to use the matlab fft to approximate the fourier coefficients. The use "fseval.m" to evaluate the Fourier series on the same interval $[0, .4]$. We also give the exact Fourier coefficients and again use "fseval.m" to evaluate the series. Then we plot the results on the same graph.

```
T=.2;
N=25;
Fn=fsfind('sawtooth',T,N,T);

n=-N:N;
Fn_exact=i*10./((2*pi*n).^2).*(1+2*pi*n);
Fn_exact(N+1)=5;  %put in average value by hand
t=linspace(0,.4);
w0=2*pi/T;   %fundamental frequency

f=fseval(Fn,t,w0);
f_e=fseval(Fn_exact,t,w0);

plot(t,f,t,f_e)
```

Now we use a hanning window in “fswindow.m” to filter the data to reduce the effect of Gibb’s phenomenon.

```
y=sawtooth(t,T);
Fnh=Fn.*fswindow(Fn,'han'); % apply a hanning window
f=fseval(Fnh,t,w0);
plot(t,f,t,y)
```

References

[1] *The Matlab Primer,* Kermit Sigmon

[2] *Introduction to scientific computing: a matrix vector approach using Matlab,* Printice Hall, 1997, Charles Van Loan

[3] *Mastering Matlab,* Printice Hall, 1996, Duane Hanselman and Bruce Littlefield

[4] *Advanced Mathematics and Mechanics Applications Using Matlab,* CRC Press, 1994, Howard B. Wilson and Louis H. Turcotte

[5] *Engineering Problem Solving with Matlab,* Printice Hall, 1993, D.M Etter

[6] *Solving Problems in Scientific Computing Using Maple and Matlab,* Walter Gander and Jiri Hrebicek

[7] *Computer Exercises for Linear Algebra,* Printice Hall, 1996, Steven Leon, Eugene Herman, Richard Faulkenberry.

[8] *Contemporary Linear Systems using Matlab,* PWS Publishing Co., 1994, Robert D. Strum, Donald E. Kirk