1a)
We know that because \( f(x) = \cos(6x) \) is a \( 2\pi \) periodic function of frequency 6 that its only nonzero coefficients exists at \( m=6 \) and \( m=-6 \). By definition, 
\[
\hat{f}_6 = \frac{1}{2\pi} \int_0^{2\pi} e^{-6ix} \cos(6x) = \frac{1}{2}.
\]
The only nonzero coefficient is therefore \( \hat{f}_6 = \frac{1}{2} \).
Note that for \( m=-6 \), the coefficient is the complex conjugate of the coefficient for \( m=6 \), which in this case is also \( \frac{1}{2} \).

1b)
We know that all of the Fourier coefficients of the series will be zero except for \( \hat{f}_6 \). We define the Fourier series to range from \(|m| < \frac{N}{2} = 5 \). We now apply the aliasing formula: 
\[
\tilde{f}_6 = \sum_{m=-4}^{m=4} \hat{f}_m e^{imx} = e^{-4i},
\]
which is the interpolant. Note that the error is huge here, because the frequency of the function lies outside the Nyquist frequency.

2a)
The following code was used for this problem:

cif
totalerror=[];
for N=2:2:40
    j=linspace(0,N-1,N);
x=2*pi*j/N;
    fp=@(x) cos(x).*exp((sin(x)));
f=@(x) exp((sin(x)));

    weightedsamples=f(x)*1/N; % take N weighted samples of the function f
    ftilde=fft(weightedsamples); % apply the DFT to these samples
    figure(1)
finegrid=0:1e-3:2*pi;
result=0;  % result will store our estimate of the real function
for n=1:N
    if n>(N/2)
        y=1i*(n-N-1)*ftilde(n)*exp(1i.*(n-N-1).*finegrid);  % notice the "in" multiplier to modify the normal ftilde
    else
        y=1i*(n-1)*ftilde(n)*exp(1i.*(n-1).*finegrid);
    end
    result=result+y;
end
error=abs(result-fp(finegrid));
totalerror=[totalerror max(error)];
end

scale=2:2:40;
semilogy(scale,totalerror,'+')
xlabel('N')
ylabel('Maximum Error')
title('Maximum Error vs. N for function cos(x)*e^{sin(x)}')

The resulting convergence plot appeared as follows:

![Convergence Plot](image)

The derivative function in this problem is arbitrarily smooth, which means that as proven in 1e on HW4, its Fourier coefficients should converge super-algebraically. Empirically, we observe that the coefficients converge even faster than super-algebraically, because this plot shows them converging super-exponentially.

2b)
The following code was used for this problem:

```matlab
clf
totalerror=[];
for N=2:2:100
    j=linspace(0,N-1,N);
    x=2*pi*j/N;
    
    fp=@(x) 3*sin(x).*sqrt(sin(x).^2).*cos(x);
    f=@(x) abs(sin(x)).^3

    weightedsamples=f(x)*1/N; % take N weighted samples of the function f
    ftilde=fft(weightedsamples); % apply the DFT to these samples

    figure(1)

    finegrid=0:1e-3:2*pi;
    result=0; % result will store our estimate of the real function
    for n=1:N
        if n>(N/2)
            y=1i*(n-N-1)*ftilde(n)*exp(1i.*(n-N-1).*finegrid);
        else
            y=1i*(n-1)*ftilde(n)*exp(1i.*(n-1).*finegrid);
        end
        result=result+y;
    end
    error=abs(result-fp(finegrid));
    totalerror=[totalerror max(error)];
end

scale=2:2:100;
loglog(scale,totalerror,’+’)xlabel(’N’)ylabel(’Maximum Error’)title(’Maximum Error vs. N for derivative of |sinx|^3’)
```

The resulting plot is as follows:
Because this plot has roughly slope two on a log-log plot, we can say that the Fourier coefficients of the first derivative appear to converge algebraically with order two. Now, we know that this function possesses two bounded derivatives, as shown below:

```
clf
axis1=(-10:.01:0)
axis2=(0:.01:10)

f=@(x) -3.*cos(x).*(sin(x)).^2

clf
axis1=(-10:.01:0)
axis2=(0:.01:10)

f=@(x) -3.*cos(x).*(sin(x)).^2

f=@(x) 3.*cos(x).*(sin(x)).^2

hold on
figure(1)
plot(axis1,f(axis1))
plot(axis2,g(axis2))
hold off
xlabel('x')
ylabel('y')
title('Plot of derivative of |sinx|^3')
```
Note: the code is then modified to give the graph of the second derivative of $|\sin x|^3$, which appears as follows:

Note how the function possesses a sharp point at $x=0$ which renders the function undifferentiable. So we know that $|\sin x|^3$ is only twice differentiable.

According to the theorem in 1e from HW4, the derivative of $|\sin x|^3$, which possesses only one bounded derivative, should have Fourier coefficients bounded as $\hat{f}_m = o\left(\frac{1}{m^2}\right)$—we’ve proven that they should converge algebraically with order one. In practice, as observed in the error graph, the Fourier coefficients converge algebraically with order two. Note that this doesn’t conflict with the theorem—it simply converges even faster than the theorem predicts.
The following function was developed for the problem:

```matlab
function output=cooleytukey(f)

N=length(f);

if N==2
    output=([1 1; 1 -1]*f')'; % the matrix is the DFT matrix for f length 2
else
    feven=zeros(1,N/2);
    fodd=zeros(1,N/2);
    for k=1:(N/2)
        feven(k)=(f(2*k-1)); % matlab is 1-indexed, whereas DFT is 0-indexed, so even entries in matlab are odd entries of DFT, and vice versa
        fodd(k)=(f(2*k));
    end
    e=cooleytukey(feven);
    o=cooleytukey(fodd);
    twiddlefactor=exp(-1*2*pi*1i*(1/N)*(0:(N/2-1)));
    o=twiddlefactor.*o
    % output=zeros(1,N);
    % output(1:N/2)=[e+o];
    % output((1:N/2)+N/2)=[e-o];
    output=[e+o,e-o];
end
```

The error between this implementation of the FFT and MATLAB’S, as well as the ratio of the runtimes of each method, were computed using the following code:

```matlab
N=floor(10*rand(1,16));
error=norm(cooleytukey(N)-fft(N))
tic;
cooleytukey(N);
mycodetime=toc; % stores the amount of time my function takes

tic;
fft(N);
matlabtime=toc; % stores the amount of time MATLAB takes
```
runtimeratio=mycodetime/matlabtime

A few runs of the test script created the following outputs:

error =
    4.3794e-15

eruntimeratio =
    58.9091

error =
    1.0822e-14

eruntimeratio =
    69.0000

error =
    3.9937e-15

eruntimeratio =
    63.9000

error =
    5.0814e-15

eruntimeratio =
    58.0909

So we can see that the error is very small and that my implementation of the FFT takes at least about 58 times longer than MATLAB’s does.
4a)

The following code, adapted from class code for this problem, was used:

```matlab
% Audio processing demo in Matlab. Barnett 4/23/13
% run system cmd: arecord -f S16_LE -r 44100 test.wav
% or get WAV exported from recording in audacity
[f fs nbits] = wavread('signoise.wav'); % f=data, fs=sampling freq, nbits=16
% plot(f);
%
f = f(50000:90000); % take chunk that want, discards rest
% plot(f); % but want correct time units
N = numel(f);
plot((0:N-1)/fs, f); xlabel('time (s)');
ft = fft(f);
[y,i]=max(ft) % returns one of the maxes
[y2,i2]=max(ft(1:64000)) % returns the other max
figure; plot(abs(ft), '-'); % what are correct freq axis units?
% T = N/fsam; % total time interval length
% df = 1/T; % frequency spacing
% plot((0:N-1)*df, abs(ft), '-'); xlabel('freq (Hz)');
% % now zoom in to near origin. Peaks are at around 130 Hz and multiples
% % Strengths of peaks tells you about timbre of sound, etc...
% semilogy((0:N-1)*df, abs(ft), '-'); xlabel('freq (Hz)'); % log vertical
% % plot shows more dynamic range
```

The following plots were produced:
Output for the magnitude of the dominant frequencies, as well as their MATLAB indices, are as follows:

\[ \mathbf{y} = 3.1442 \times 10^2 + 2.8471 \times 10^2 i \]

\[ \mathbf{i} = 64883 \]
\[ y_2 = 3.1442e+02 - 2.8471e+02i \]

\[ i_2 = 655 \]

Figure A and B both display some periodicity along their edges; the top and bottom in A, and the top in B, appear to assume a roughly sinusoidal pattern of rising and falling. The two dominant frequencies lie at MATLAB indices 655 and 64883. Because MATLAB is 1-indexed and the Fourier mode is 0-indexed for the first N/2 frequencies, we know that the Fourier mode index corresponding to the MATLAB index 655 is 654. To find the Fourier mode index of the other dominant frequency, we simply compute \( N-m+1 \), where \( N \) is the sample size and \( m \) is the Fourier mode index of a dominant frequency. 65536-654+1=64883, so we know that the Fourier mode index of the other dominant frequency is 64883 – what happened here is that the second half of the N frequencies, that range from \( N/2 \) to \( N \), are wrapped around, so in Fourier mode they range from \(-N/2\) to \(-1\). To sum it up, the first \( N/2 \) frequencies are 0-indexed, whereas the second \( N/2 \) frequencies are 1-indexed.

The coefficients are related by the fact that they are conjugates, as displayed in the output above. This is because the input signal had frequency \( m=654 \), so the only non-zero DFT fourier coefficients would be \( \tilde{f}_{654} \) and \( \tilde{f}_{-654} = \tilde{f}_{654}^* \).

4b)

The true frequency is given by the formula \( f = \frac{m f_s}{N} \), where \( N \) is the number of samples, \( f_s \) is the sampling frequency, and \( m \) is the observed frequency. In this case, \( f = \frac{654\times44100}{65536} \approx 440 \) Hz.

5a)

In this problem, the actual computations will be omitted for brevity’s sake. We compute \( (f * g) \) where \( f = [1, 1, 1, 0, 0, 0] \) and \( g = [1, 1, 1, 0, 0] \) to get \( (f * g) = [1, 2, 3, 3, 2, 1] \).

5b)

In this case, \( f = [1, 1, 1, 0, 0] \) and \( g = [1, 1, 1, 1, 0] \), so \( (f * g) = [2, 2, 3, 3, 2] \).
5c)

First, we pad each of the vectors with 0’s to make them of length \( N = N_1 + N_2 - 1 \). We do this because when the signal of length \( N_1 \) is convolved with the signal of length \( N_2 \), the resulting signal extends out \( N_1 - 1 \) in one direction, \( N_2 - 1 \) in the other direction, but then we add 1 to account for having subtracted one of the elements one too many times. Letting \( N_1 = 6 \) and \( N_2 = 6 \), we get \( N = 11 \). Therefore, \( f = [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0] \) and \( g = [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0] \). Convolving these two returns, excluding zero entries, \([1, 2, 3, 3, 2, 1]\). So the non-periodic convolution returns the same value as that of (a).

5d)

The DFT of \((f \ast g)\) yields \( \tilde{f} \ast \tilde{g} = \tilde{g} \ast \tilde{f} = \tilde{g} \ast f \). Performing the inverse DFT on \( g \ast f \) yields \((g \ast f)\). Therefore we know that \((f \ast g) = (g \ast f)\).

5e)

We observe that in a, the sum of the entries of \( f \) (3) multiplied by the sum of the entries of \( g \) (4) yields the sum of the entries of \( f \ast g \), 12. So we conjecture that \( \sum_{k=0}^{N-1} (f \ast g)_k = \sum_{j=0}^{N-1} f_j \sum_{i=0}^{N-1} g_i \). Applying the definition of convolution, we find that \( \sum_{k=0}^{N-1} (f \ast g)_k = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f_i g_{j-i} (modN) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} g_{j-i} (modN) \). To prove the conjecture, all we need to do is show that \( \sum_{j=0}^{N-1} g_{j-i} (modN) = \sum_{i=0}^{N-1} g_i \). However, this immediately follows from the fact that if we fix an \( i \), the sum becomes \( g_{0-i} (modN) + g_{1-i} (modN) + g_{2-i} (modN) + ... + g_{N-1-i} (modN) \)–that sum is equivalent to simply adding up all the \( g_i \), where \( i \) ranges from 0 to \( N-1 \). Thus indeed \( \sum_{k=0}^{N-1} (f \ast g)_k = \sum_{j=0}^{N-1} f_j \sum_{i=0}^{N-1} g_i \).

6)

The following code was used for this problem:

```matlab
[f fs nbits] = wavread('h5p6.wav'); % f=data, fs=sampling freq, nbits=16
[g gs gnbits] = wavread('impulseresponse.wav'); % f=data, fs=sampling freq, nbits=16
N=length(f)+length(g)-1;
f=transpose(f);
g=transpose(g);
```

11
padf=zeros(1,N-length(f));
padg=zeros(1,N-length(g));

f=[f, padf];
g=[g, padg];

ftilde=fft(f);
gtilde=fft(g);

convtilde=ftilde.*gtilde;
conv=ifft(convtilde);
conv=conv/max(conv);
wavwrite(conv,44100,'output2.wav');

The 'output2' file generated is the convolution of the two signals—the convolution is much more echoey than the original file, and it also seems to have picked up some of the beat of the clapping sound.

7)

The following code was used for the problem:

blurry=textread('blurry.txt');
blurry=reshape(blurry,512,512);

figure(1)
imagesc(blurry)
colormap(gray(256))
title('Blurry Image')
axis equal

aperture=textread('aperture.txt');
aperture=reshape(aperture,512,512);
figure(2)
imagesc(aperture)
colormap(gray(256))
title('Aperture Image')
axis equal

fgtilde=fft2(blurry);
\texttt{gtilde=fft2(aperture);}
\texttt{ftilde=gtilde./gtilde;}
\texttt{f=ifft2(ftilde);}
\texttt{figure(3)}
\texttt{imagesc(f)}
\texttt{colormap(gray(256))}
\texttt{title('Deconvolved Image')} 
\texttt{axis equal}

\texttt{N=length(blurry);}
\texttt{error=randn(N)*3*10^{-5};}
\texttt{newblurry=error+blurry;}
\texttt{newfgtilde=fft2(newblurry);}
\texttt{newftilde=newfgtilde./gtilde;}
\texttt{newf=ifft2(newftilde);}
\texttt{figure(4)}
\texttt{imagesc(newf)}
\texttt{colormap(gray(256))}
\texttt{title('Deconvolved Image With Error')} 
\texttt{axis equal}

\texttt{figure(5)}
\texttt{imagesc(real(gtilde))}
\texttt{colorbar}
\texttt{title('DFT of Aperture')} 
\texttt{axis equal}

a)

The following blurred image and aperture image were produced:
b)

Using Fast Fourier Transform techniques, the image is deconvolved to yield the following clear image:
c) After adding noise to the original blurry matrix, the image is deconvolved using the same technique as in b) to yield the following image:

The result appears as it does because the DFT of the aperture image, by which the blurry image is divided, has primarily very tiny values, as shown here:
As a result, even error on the order of $3 \times 10^{-5}$ is blown up to a huge amount.

Attribution: I worked on this homework with Matt Marcus and Michael Downs, and a little bit with Max.