% Carlos Lazo

% ECE 503

% Homework #6

% Due: 3/9/10

%% 1) The FFT and Frequency-Domain Representation

clear all; close all; clc;

% b. P&M 8.36

% Part (1)

% Initialize all sequences and N values:

x\_na = ones(1,16);

Na = 64;

x\_nb = ones(1,8);

Nb = 64;

x\_nc = x\_na;

Nc = 128;

x\_nd = 10.\*exp(j\*(pi/8)\*[1:64]);

Nd = 64;

% Plot the N-point DFTs (using zero-padding):

X\_ka = fft(x\_na, Na);

X\_kb = fft(x\_nb, Nb);

X\_kc = fft(x\_nc, Nc);

X\_kd = fft(x\_nd, Nd);

% Plot the phase response of all DFTs:

figure;

plot (abs(X\_ka));

ylabel('Magnitude');

title('|X(k)|, part a)');

figure;

plot (abs(X\_kb));

ylabel('Magnitude');

title('|X(k)|, part b)');

figure;

plot (abs(X\_kc));

ylabel('Magnitude');

title('|X(k)|, part c)');

figure;

plot (abs(X\_kd));

ylabel('Magnitude');

title('|X(k)|, part d)');

% Part (2)

fprintf('Xk\_a(0) is: '); real(X\_ka(1))

fprintf('Xk\_b(0) is: '); real(X\_kb(1))

fprintf('Xk\_c(0) is: '); real(X\_kc(1))

fprintf('Xk\_d(0) is: '); real(X\_kd(1))

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*

% \*\*\* OUTPUT \*\*\*

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Xk\_a(0) is:

% ans =

%

% 16

%

% Xk\_b(0) is:

% ans =

%

% 8

%

% Xk\_c(0) is:

% ans =

%

% 16

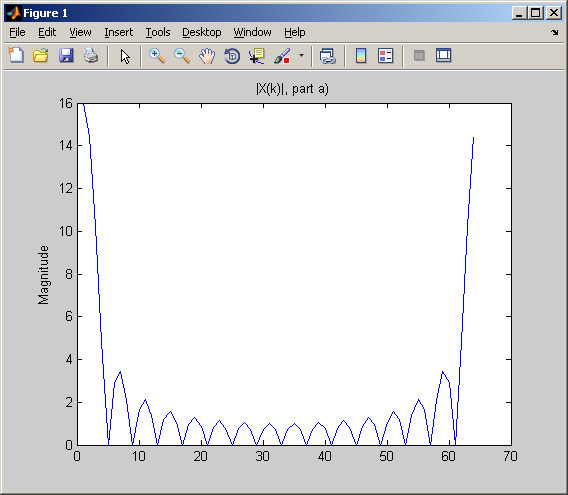
%

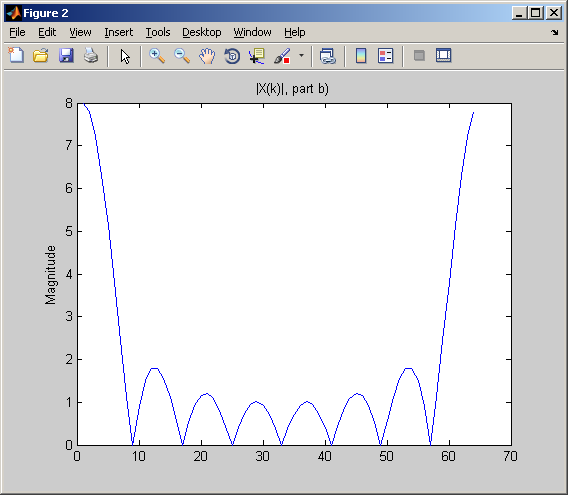
% Xk\_d(0) is:

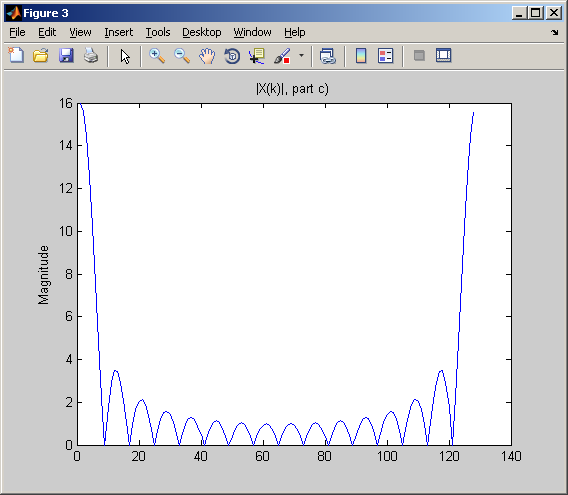
% ans =

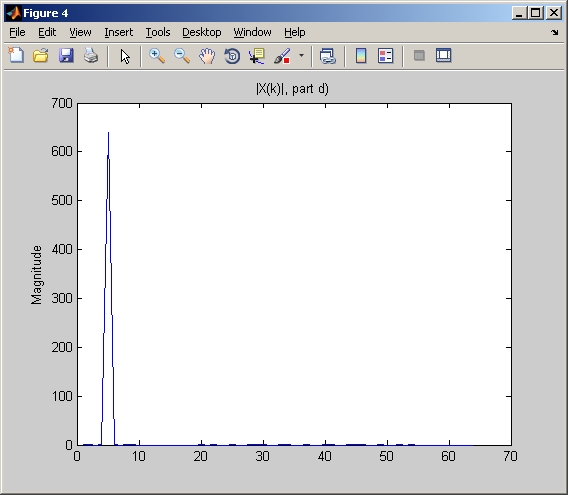
%

% -7.7729e-014









%% 1) The FFT and Frequency-Domain Representation

clear all; close all; clc;

% e. DFT Arithmetic Resolution

% First, create the given input sequence and its DFT:

x\_n = [1:64];

N = length(x\_n);

X\_k = fft(x\_n);

format long;

% Now, to implement a precision reduction algorithm:

RMS\_reg = sqrt ( (sum(x\_n.^2)) / length (x\_n));

x\_nerr = zeros(1,15);

err = zeros(1,15);

% My precision reduction algorithm takes advantage of the

% MATLAB Symbolic Toolbox. Using the VPA function (Variable

% Point Arithmetic), I'm setting the default format for all

% variable to be 16 bits. With VPA, I'm reducing the # of

% decimal points (bits) that will be used in each value of

% X\_k. I then reconvert the value from a symbolic variable

% back to a reduced-precision double, and find it's IFFT.

% Finally, the RMS error is calculated.

% The overall algorithm reduces bit size by 15 bits, the

% maximum number allowed so you still have somewhat decent

% value representation.

for i = 1:15

X\_kerr = vpa(X\_k, 16 - i);

x\_nerr = ifft(double(X\_kerr));

RMS\_err = sqrt ( (sum(x\_nerr.^2)) / length (x\_nerr));

err(i) = abs(RMS\_reg - RMS\_err);

fprintf (['X(k) reduction by' num2str(i) ' bits yield an RMS error of: ' num2str(err(i)) '\n']);

end

figure;

plot(err, '-ob');

xlabel('Variable point arithmetic, Bit Reduction Quantity');

ylabel('RMS Error');

title('Bit reduction on X(k) ==> RMS effects on x(n)');

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*

% \*\*\* OUTPUT \*\*\*

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*

% X(k) reduction by 1 bits yield an RMS error of: 0

% X(k) reduction by 2 bits yield an RMS error of: 7.1054e-015

% X(k) reduction by 3 bits yield an RMS error of: 6.3949e-014

% X(k) reduction by 4 bits yield an RMS error of: 1.2008e-012

% X(k) reduction by 5 bits yield an RMS error of: 1.09e-011

% X(k) reduction by 6 bits yield an RMS error of: 2.3419e-011

% X(k) reduction by 7 bits yield an RMS error of: 8.5807e-010

% X(k) reduction by 8 bits yield an RMS error of: 4.9924e-008

% X(k) reduction by 9 bits yield an RMS error of: 7.6377e-007

% X(k) reduction by 10 bits yield an RMS error of: 2.764e-006

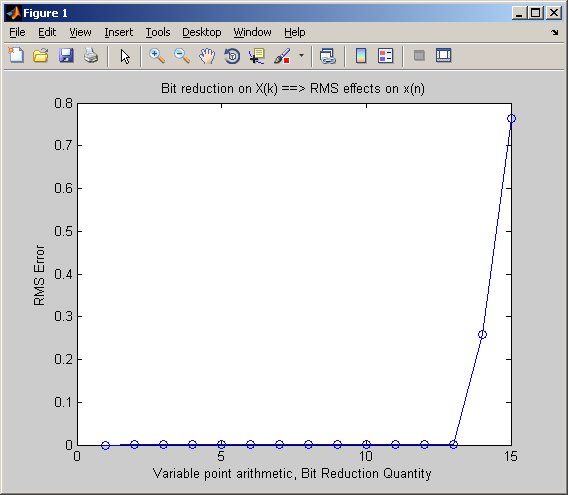
% X(k) reduction by 11 bits yield an RMS error of: 5.0148e-005

% X(k) reduction by 12 bits yield an RMS error of: 0.00027055

% X(k) reduction by 13 bits yield an RMS error of: 0.0019331

% X(k) reduction by 14 bits yield an RMS error of: 0.25912

% X(k) reduction by 15 bits yield an RMS error of: 0.76402



%% 2) DFT Implementation

clear all; close all; clc;

format long;

% d. Filter Arithmetic Resolution

% First, create the given input sequence:

x\_n = ones(1,32);

% Create the linear filter:

[b, a] = butter(4, 0.5);

% Filter to produce the output sequence:

y\_n = filter(b, a, x\_n);

% Now, to implement a precision reduction algorithm:

RMS\_reg = sqrt ( (sum(y\_n.^2)) / length (y\_n));

% This implementation uses bit reduction, reducing the

% total amount of possible significant figures used to

% represent the values return into a & b to 4, 2, and 1

% significant digits, respectively.

% The new coefficients are computed below:

a\_err4 = vpa(a,4);

b\_err4 = vpa(b,4);

a\_err2 = vpa(a,2);

b\_err2 = vpa(b,2);

a\_err1 = vpa(a,1);

b\_err1 = vpa(b,1);

% Now filter each of the sequences and find their RMS errors.

% Must divide by 10e4 to account for retransmission:

y\_nerr4 = filter(double(b\_err4), double(a\_err4), x\_n);

y\_nerr2 = filter(double(b\_err2), double(a\_err2), x\_n);

y\_nerr1 = filter(double(b\_err1), double(a\_err1), x\_n);

RMS\_err4 = sqrt ( (sum(y\_nerr4.^2)) / length (y\_nerr4));

RMS\_err2 = sqrt ( (sum(y\_nerr2.^2)) / length (y\_nerr2));

RMS\_err1 = sqrt ( (sum(y\_nerr1.^2)) / length (y\_nerr1));

err(1) = RMS\_reg - RMS\_err4;

err(2) = RMS\_reg - RMS\_err2;

err(3) = RMS\_reg - RMS\_err1;

figure;

subplot(2,1,1);

plot(y\_n, '-k'); hold on;

plot(y\_nerr4, '--or'); hold on;

plot(y\_nerr2, ':xb'); hold on;

plot(y\_nerr1, '-.^g'); hold off;

xlabel('n');

ylabel('y(n)');

title ('Bit Reduction on a & b Filter Coefficients');

legend('Original y(n)', '4 Sig Fig Reduction', '2 Sig Fig Reduction', '1 Sig Fig Reduction');

subplot(2,1,2);

stem (err);

xlabel('RMS Reduction: err(1) = 4 bits, err(2) = 2 bits, err(3) = 1 bit');

ylabel('RMS Error');

title ('RMS Error caused by Bit Reduction on a & b Filter Coefficients');

