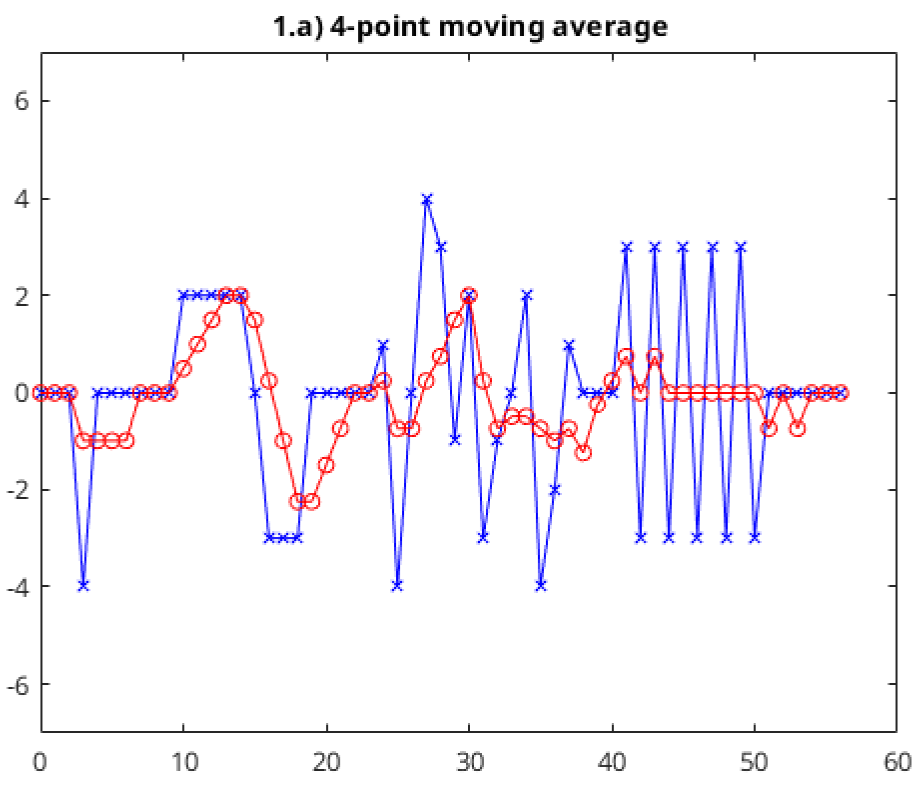
Contents

1.a

```
x = [ 0 0 0 -4 0 0 0 0 0 0 2 2 2 2 ...
2 0 -3 -3 -3 0 0 0 0 0 1 -4 0 4 ...
3 -1 2 -3 -1 0 2 -4 -2 1 0 0 0 3 ...
-3 3 -3 3 -3 3 -3 3 -3 0 0 0 0 0 0 ];
n = 0:length(x)-1;

a=[1];
b=[1 1 1 1]/4;
y = filter(b, a, x);
figure;
plot(n, x, 'bx-', n, y, 'ro-');
ylim([-7,7]);
title("1.a) 4-point moving average");
```



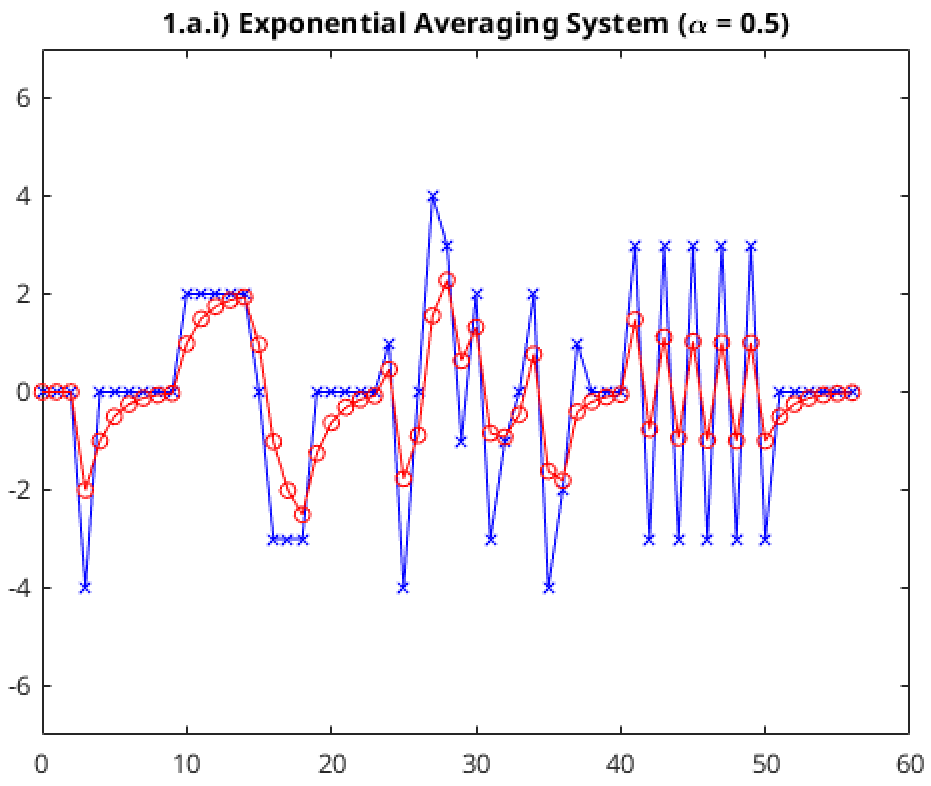
1.a.i) Exponential Averaging System

$y_n = alpha * x_n + (1 - alpha) * y_{n-1}$

$=> y_n - (1 - alpha) * y_{n-1} = alpha * x_n$

$=> [1, -(1 - alpha)].[y_n, y_{n-1}] = [alpha].[x_n]$

```
alpha = 0.5;
a=[1, -(1 - alpha)];
b=[alpha];
y = filter(b, a, x);
figure;
plot(n, x, 'bx-', n, y, 'ro-');
ylim([-7,7]);
title("1.a.i) Exponential Averaging System ({\alpha = 0.5})");
```



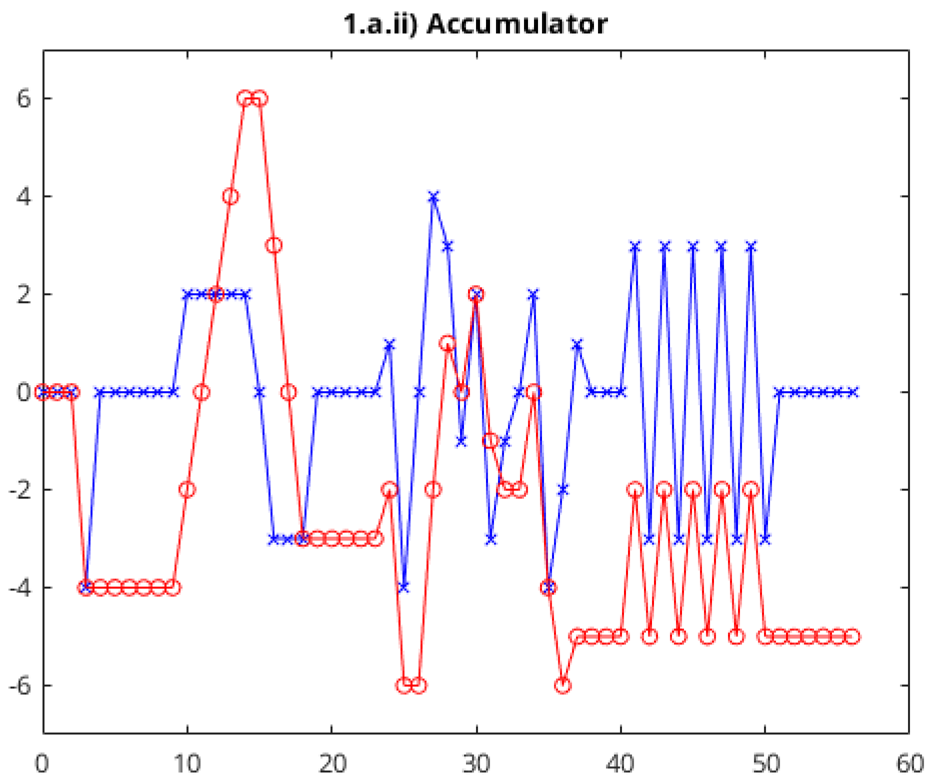
1.a.ii) Accumulator

$y_n = x_n + y_{n-1}$

$=> y_n - y_{n-1} = x_n$

$=> [1, -1].[y_n, y_{n-1}] = [1].[x_n]$

```
a=[1, -1];
b=[1];
y = filter(b, a, x);
figure;
plot(n, x, 'bx-', n, y, 'ro-');
ylim([-7,7]);
title("1.a.ii) Accumulator");
```

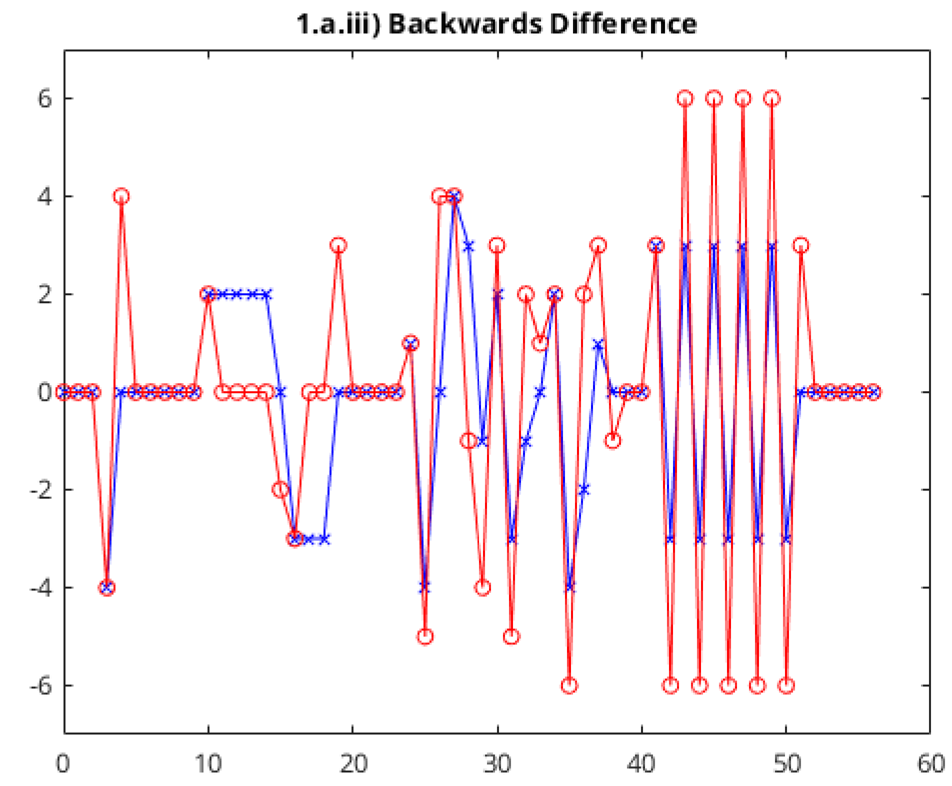


1.a.iii) Backwards Difference System

$y_n = x_n - x_{n-1}$

$[1].[y_n] = [1, -1].[x_n, x_{n-1}]$

```
a=[1];
b=[1, -1];
y = filter(b, a, x);
figure;
plot(n, x, 'bx-', n, y, 'ro-');
ylim([-7,7]);
title("1.a.iii) Backwards Difference");
```



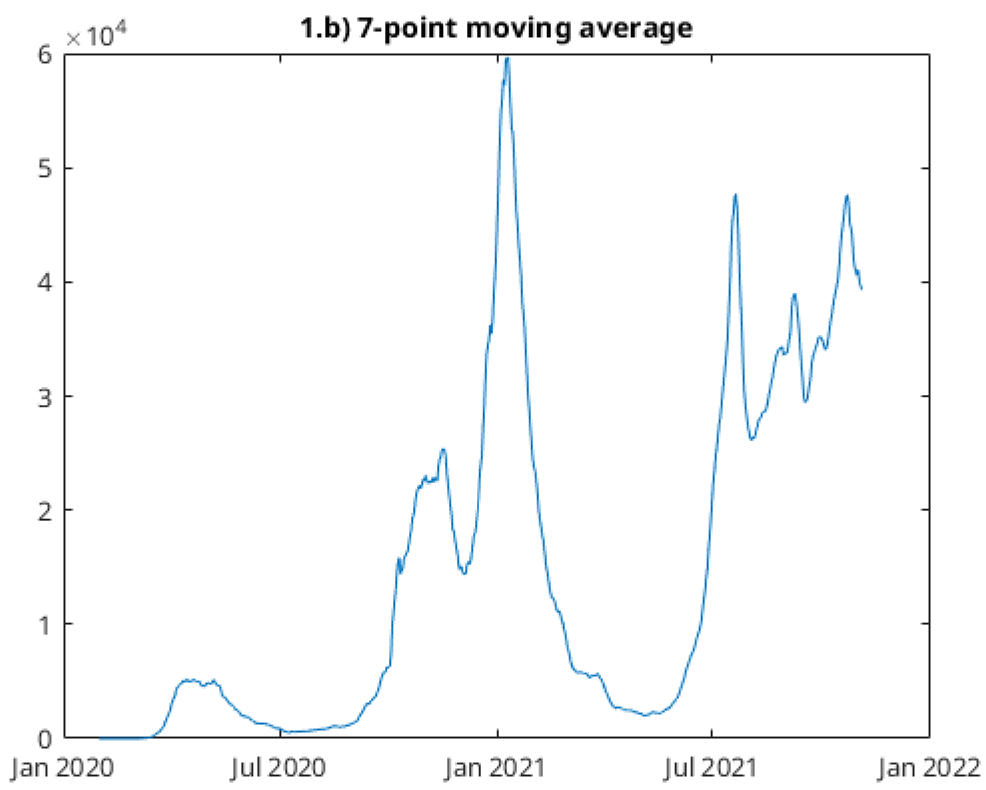
1.b

Get table date, taking only the 'date' and 'newCasesByPublishDate' columns

```
newcases = readtable("covid_new_cases_2021_11_04.csv", 'Range', 'D:E');
% Sort table by 'date'
newcases = sortrows(newcases, 1);
% Take x = newcases.newCasesByPublishDate
x = newcases.newCasesByPublishDate;
```

1.b) 7 Point Moving Average

```
a=[1];
b=[1 1 1 1 1 1 1]/7;
y = filter(b, a, x);
figure;
plot(newcases.date, y)
title("1.b) 7-point moving average");
```



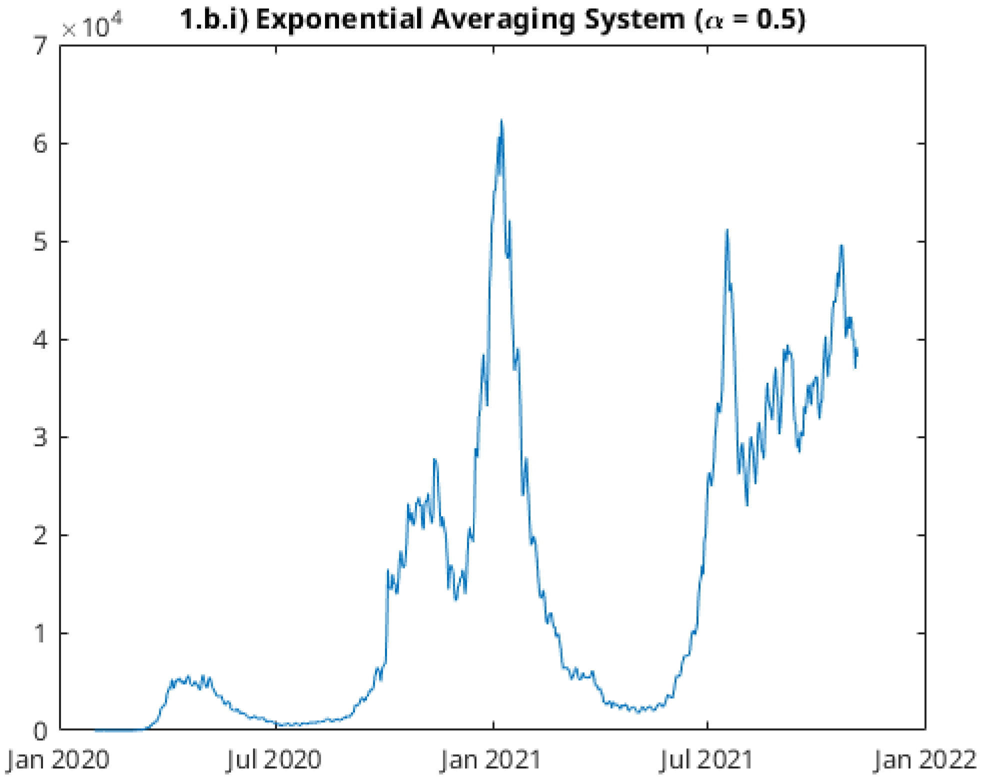
1.b.i) Exponential Averaging System

$y_n = alpha * x_n + (1 - alpha) * y_{n-1}$

$=> y_n - (1 - alpha) * y_{n-1} = alpha * x_n$

$=> [1, -(1 - alpha)].[y_n, y_{n-1}] = [alpha].[x_n]$

```
alpha = 0.5;
a=[1, -(1 - alpha)];
b=[alpha];
y = filter(b, a, x);
figure;
plot(newcases.date, y)
title("1.b.i) Exponential Averaging System ({\alpha = 0.5})");
```



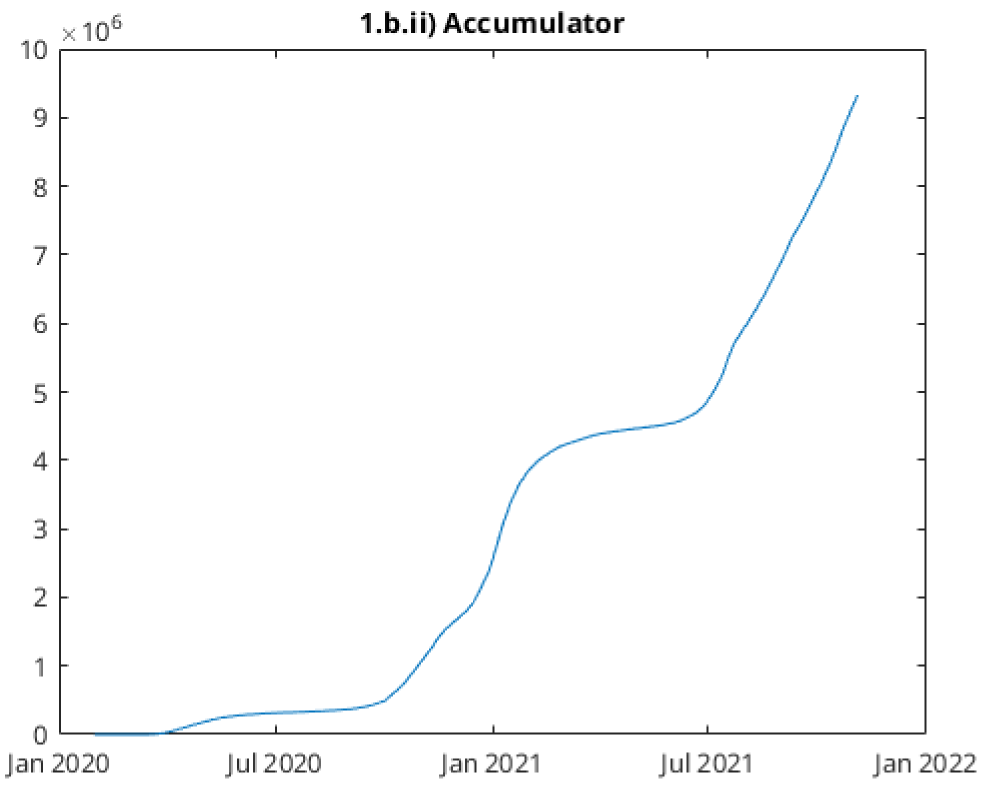
1.b.ii) Accumulator

$y_n = x_n + y_{n-1}$

$=> y_n - y_{n-1} = x_n$

$=> [1, -1].[y_n, y_{n-1}] = [1].[x_n]$

```
a=[1, -1];
b=[1];
y = filter(b, a, x);
figure;
plot(newcases.date, y)
title("1.b.ii) Accumulator");
```

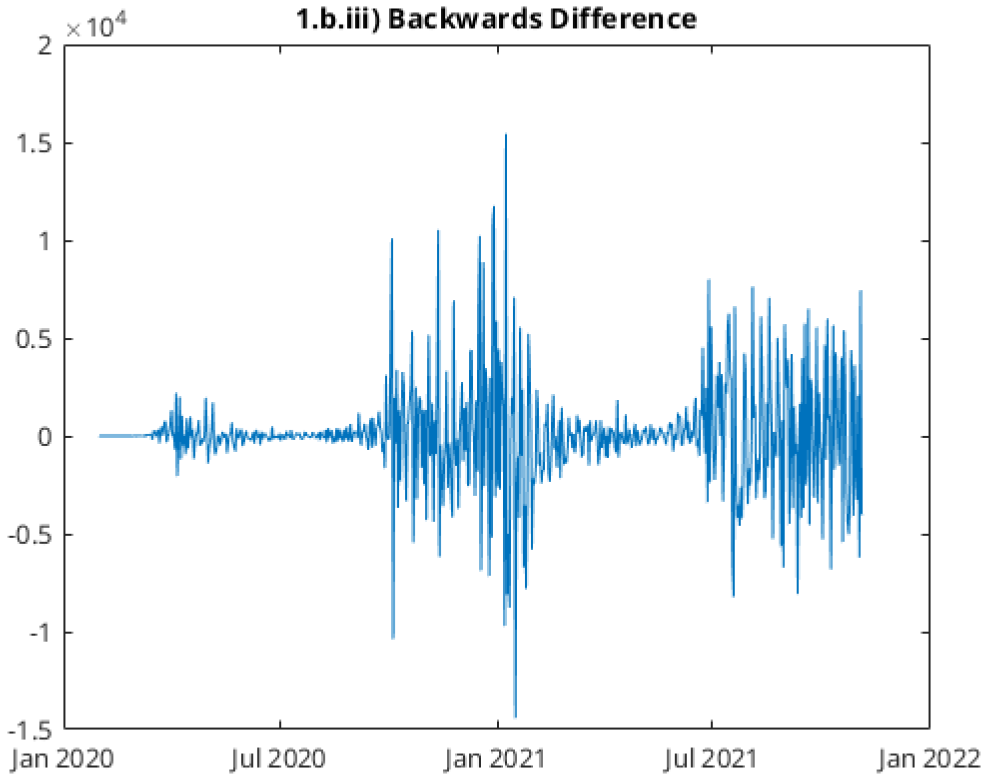


1.b.iii) Backwards Difference System

$y_n = x_n - x_{n-1}$

$[1].[y_n] = [1, -1].[x_n, x_{n-1}]$

```
a=[1];
b=[1, -1];
y = filter(b, a, x);
figure;
plot(newcases.date, y)
title("1.b.iii) Backwards Difference");
```



1.c

Generate a one second long Gaussian noise sequence r with a sampling rate of 300Hz

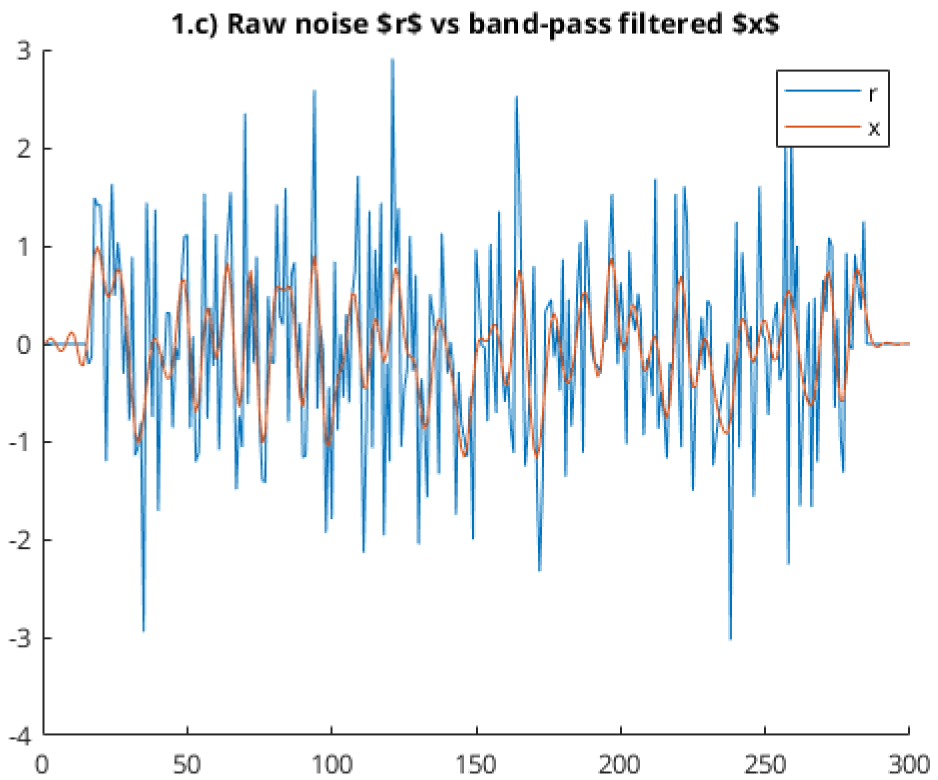
```
f_s = 300;
t = 1; % 1 second long
N = f_s * t;
ts = linspace(1/f_s, t, N);
r = randn(N,1);
% Taper $r$ by setting its first and last 15 samples to zero.
r(1:15) = 0;
r(end-15:end) = 0;
```

Make a Finite-Impulse Response low-pass filter with cut-off frequency

```
f_c = 45; % cutoff frequency
a = [1]; % FIR has no y-terms
b = fir1(50, f_c/(f_s/2));
% Use the filtfilt function in order to apply that filter to the
% generated noise signal, resulting in the filtered noise signal x
x = filtfilt(b, a, r);
```

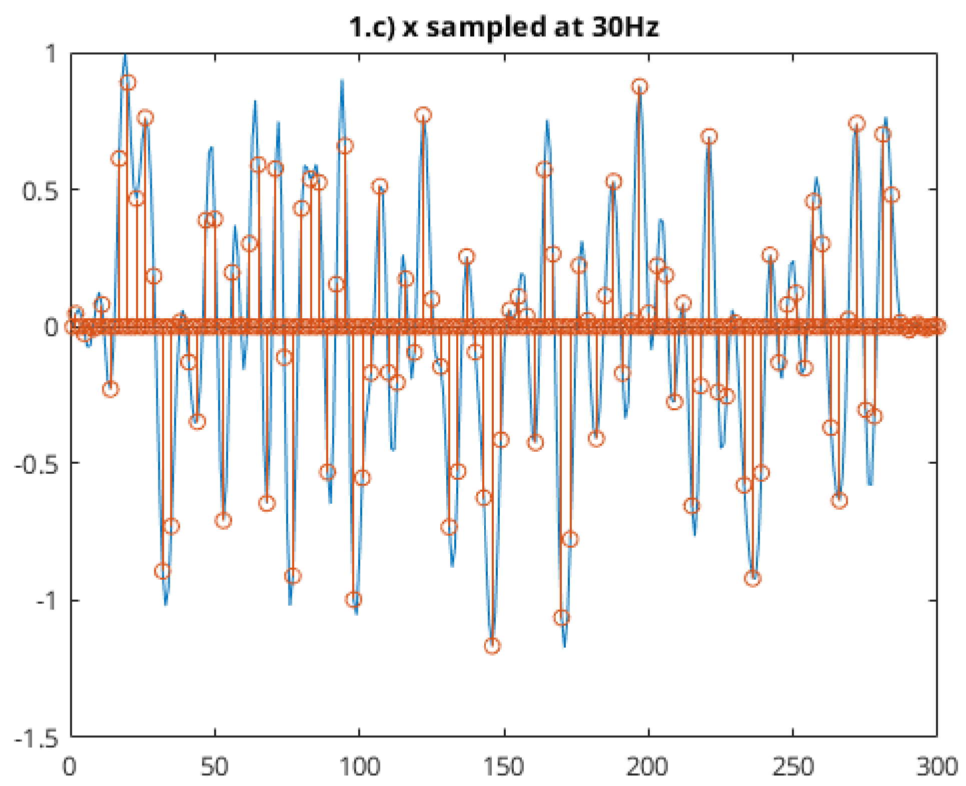
Plot r, x together.

```
figure;
hold on;
plot(r);
plot(x);
legend("r","x");
title("1.c) Raw noise $r$ vs band-pass filtered $x$");
hold off;
```



Sample x at 100Hz by setting all but every third sample value to zero

```
y = x;
% element 1, 4, 7... = 0
y(1:3:end) = 0;
% element 2, 5, 8... = 0
y(3:3:end) = 0;
% element 3, 6, 9... = unchanged

figure;
plot(x);
hold on;
stem(y);
title("1.c) x sampled at 30Hz");
hold off;
```



Implement sinc interpolation to reconstruct the zeroed samples of y

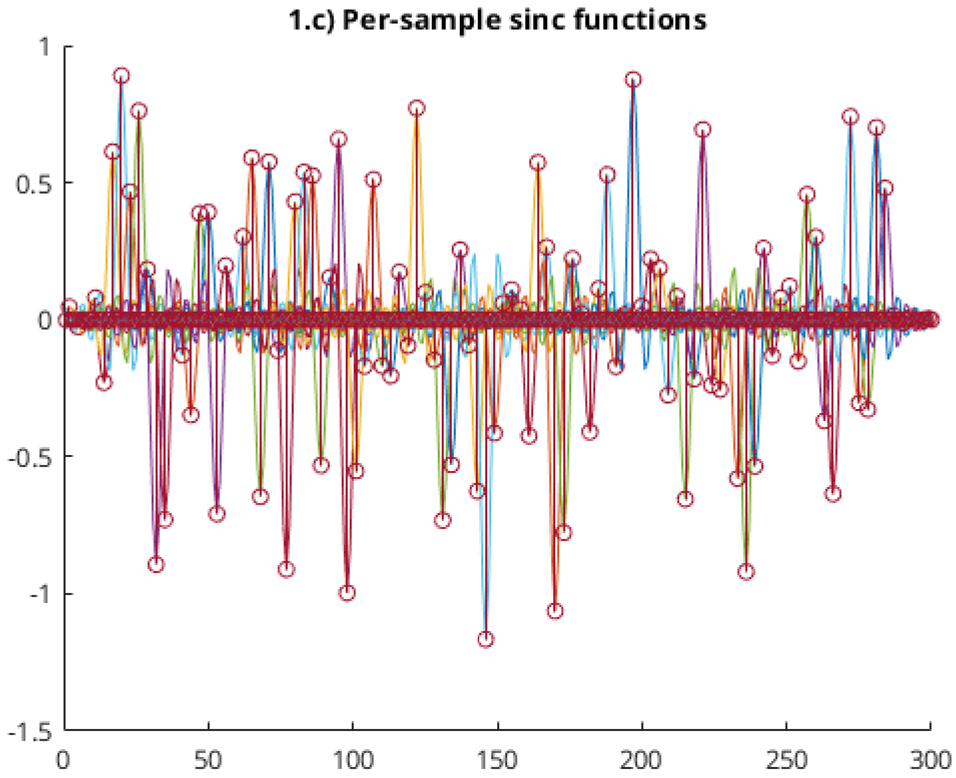
$$x(t) = \sum_{n=-\infty}^{\infty} x_n * sinc(t/t_s - n)$$

MATLAB version:

$$z = \sum_{i_y=1}^{N} y(i_y) * sinc((ts/t - i_y/N) * f_s/3)$$

Translate the sinc by i_y/N to align it with the sample, and scale it by f_s over 3 to align the zero crossings with the other sample points.

```
z = zeros(N,1);
figure;
hold on;
for i_y = 1:N
    data = y(i_y) * sinc((ts/t - (i_y)/N)* f_s/3);
    plot(data);
    z = z + data';
end
stem(y)
hold off;
title("1.c) Per-sample sinc functions")
```



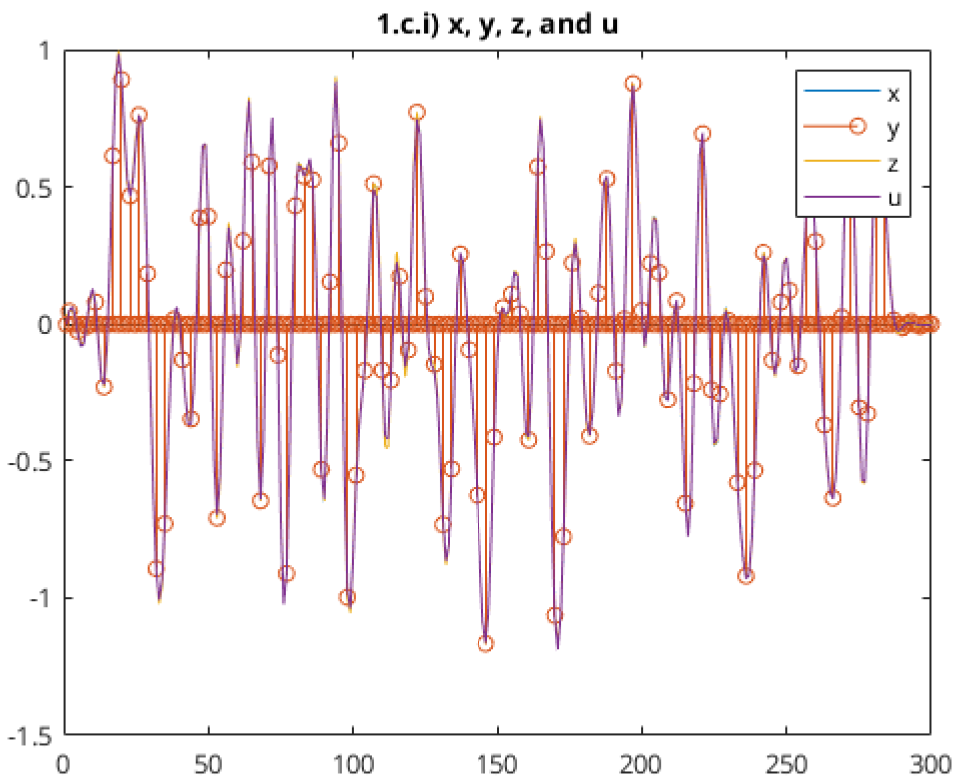
Generate another low-pass filter with fir1, cut-off frequency 50Hz

```
f_c = 50; % cutoff frequency
a = [1]; % FIR has no y-terms
b = fir1(50, f_c/(f_s/2));
% Apply it to y, resulting in interpolated sequence u.
% multiply by 3 to compensate for energy lost during sampling.
u = 3 * filtfilt(b, a, y);
```

1.c.i) Comparisons

Plot x, y, z, and u on top of each other in one figure

```
figure;
plot(x);
hold on;
stem(y);
plot(z);
plot(u);
legend("x", "y", "z", "u");
title("1.c.i) x, y, z, and u")
hold off;
```

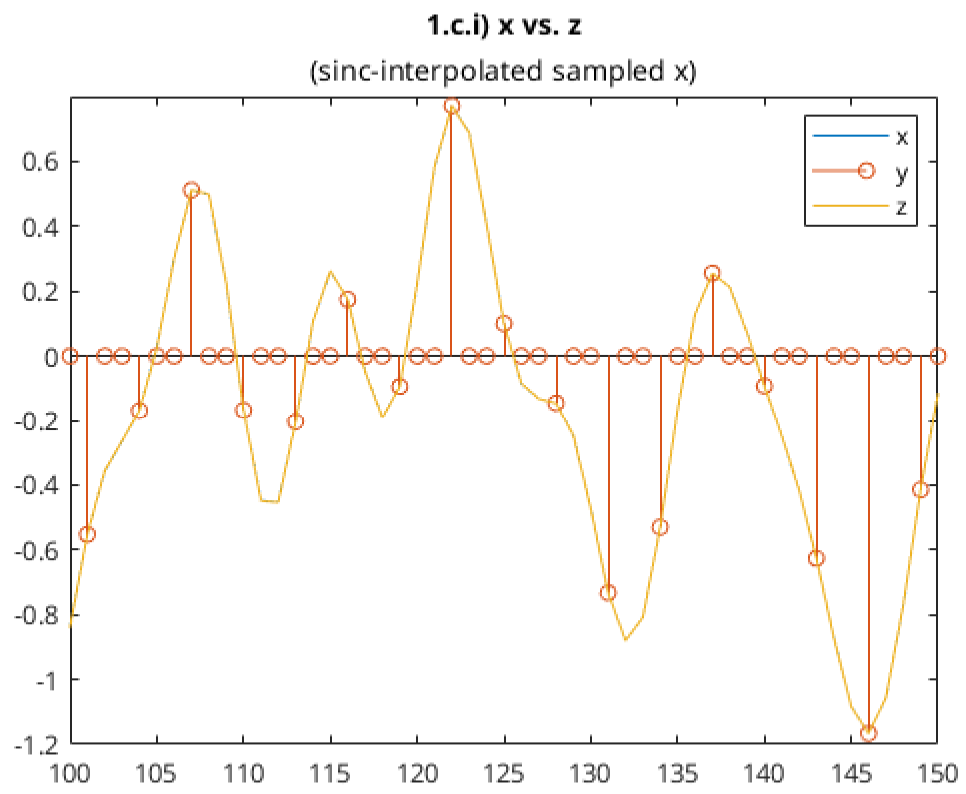


1.c.i) x vs z

x and z are equal

```
figure;
plot(x);
hold on;
stem(y);
plot(z);

legend("x", "y", "z");
xlim([100 150]);
```

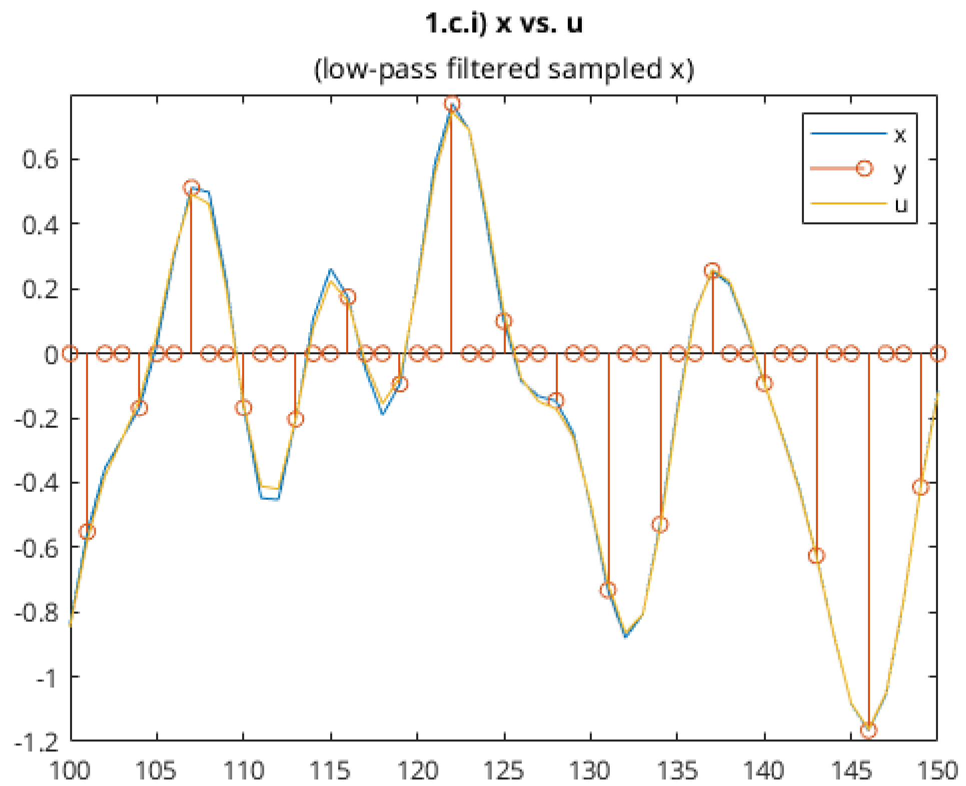
```
title("1.c.i) x vs. z", "(sinc-interpolated sampled x)");
hold off;
```



1.c.i) x vs u

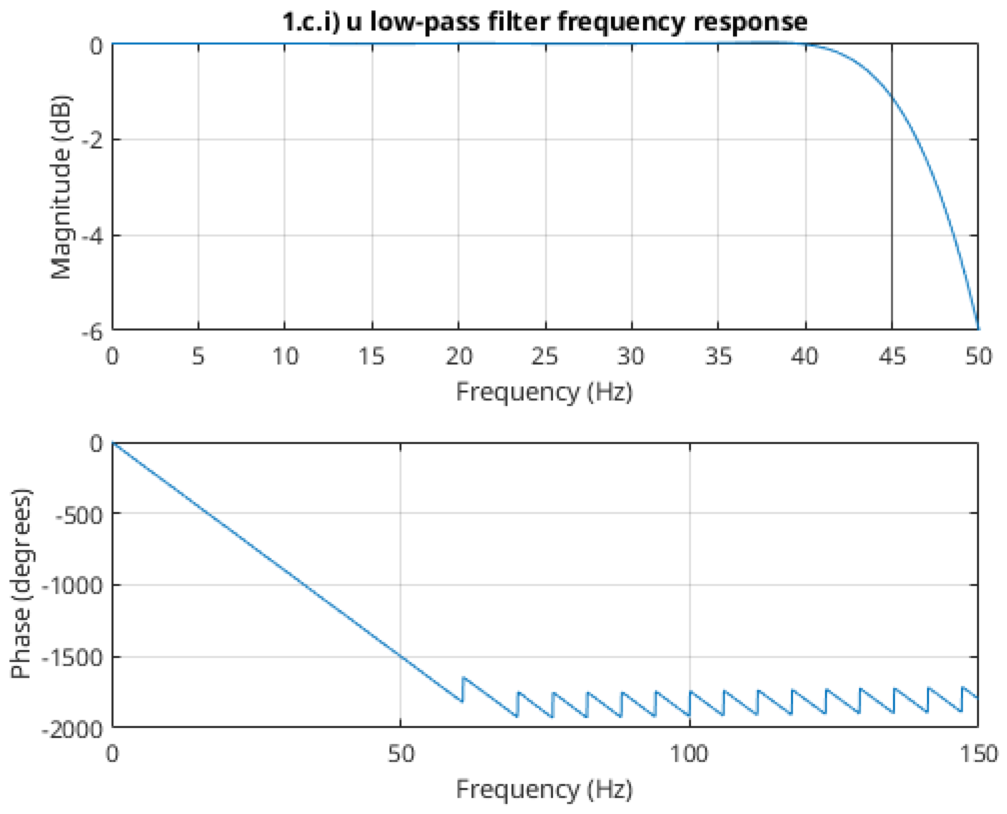
x and u are similar, but not equal. I think this is because the low-pass filter isn't perfect.

```
figure;
plot(x);
hold on;
stem(y);
plot(u);

legend("x", "y", "u");
xlim([100 150]);
title("1.c.i) x vs. u", "(low-pass filtered sampled x)");
hold off;
```



Frequency response of the second low-pass filter. The magnitude at the original cutoff (45Hz) is -3dB, so higher frequencies in x were be unfairly reduced when generating u.

```
f_c = 50;
a = [1];
b = fir1(50, f_c/(f_s/2));
figure;
freqz(b,a,2048,f_s);
xlim([0, 50]);
xline(45);
title("1.c.i) u low-pass filter frequency response");
```



1.c.ii)

Q: Why should the first filter have a lower cut-off frequency than the second?

If the second low-pass filter had a lower cut-off frequency than the first filter, it would discard higher-frequency information and not reconstruct it.

1.d

Simulate the reconstruction of a sampled band-pass signal

Generate a 1 s noise sequence r, as in part (c)(i), but this time use a sampling frequency of 3 kHz. Set the first and last 500 samples to zero.

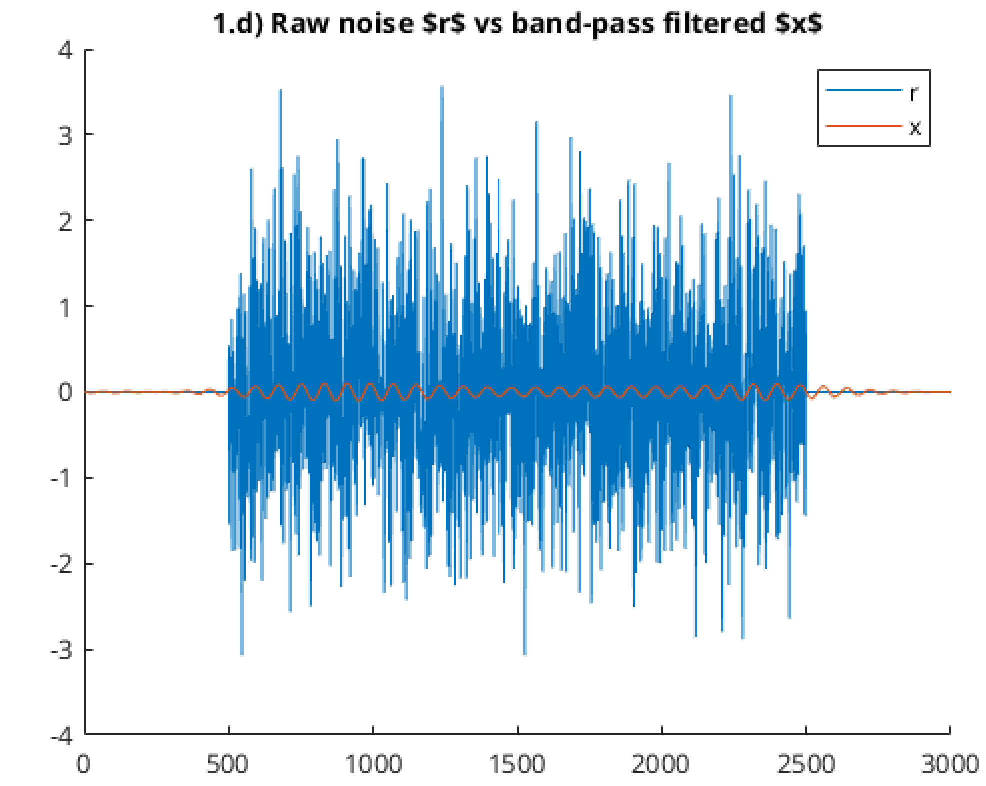
```
f_s = 3000;
t = 1; % 1 second long
N = f_s * t;
ts = linspace(1/f_s, t, N);
r = randn(N,1);
r(1:500) = 0;
r(end-500:end) = 0;
```

Apply a band-pass filter that attenuates frequencies outside 31-44Hz

```
f1 = 31;
f2 = 44;
% 3 = 3rd order filter
% 30 = -30dB for frequencies outside the range
[b, a] = cheby2(3, 30, [f1 f2]/(f_s/2));
% Apply the filter
x = filtfilt(b, a, r);
```

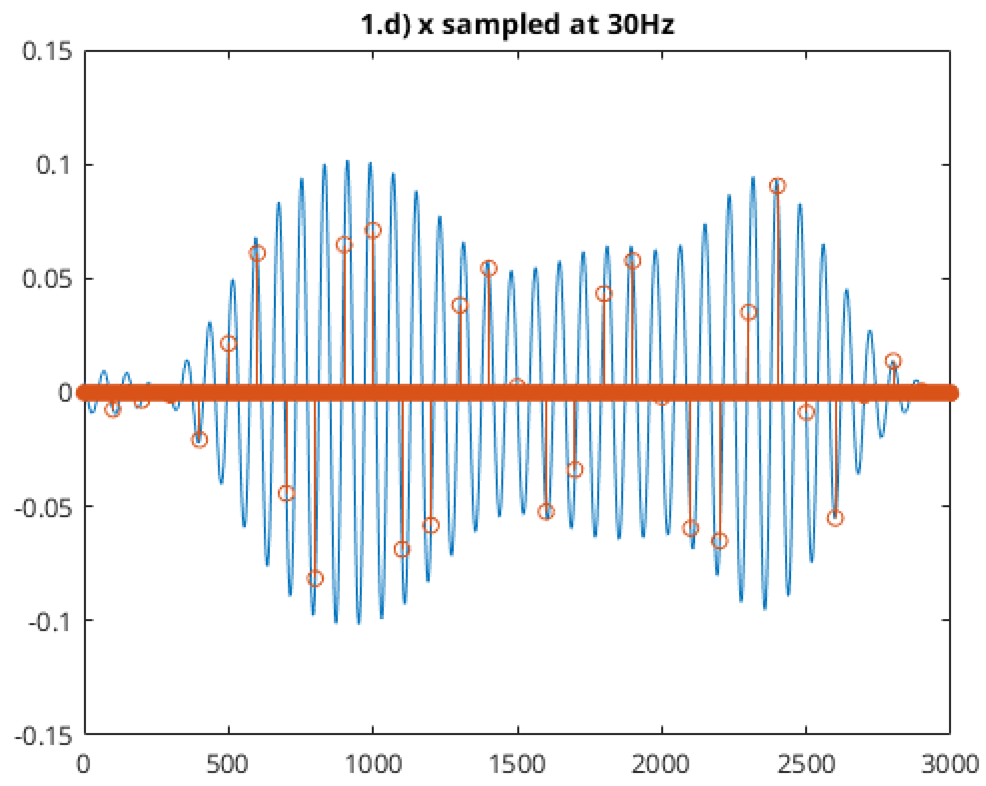
Plot r, x together. r is very high frequency, so the band-pass will remove a lot and x will have a much lower amplitiude

```
figure;
hold on;
plot(r);
plot(x);
legend("r","x");
title("1.d) Raw noise $r$ vs band-pass filtered $x$");
hold off;
```



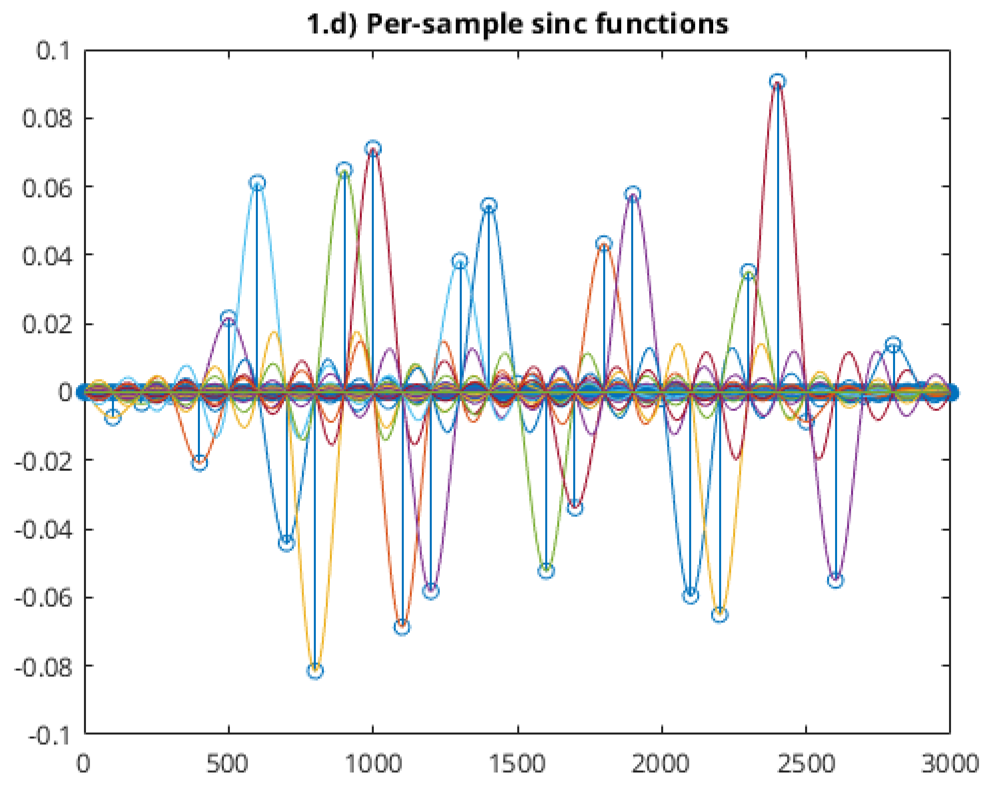
Sample x at 30Hz, set all but every 100th value to 0

```
y = x;
for i = 1:99
    y(i:100:end) = 0;
end

figure;
plot(x);
hold on;
stem(y);
title("1.d) x sampled at 30Hz");
hold off;
```



Reconstruct y with sinc interpolation (see 1.c.i for working) Change the scaling factor to 1/100 instead of 1/3, because we sampled every 100th value

```
z = zeros(N,1);
figure;
stem(y)
hold on;
for i_y = 1:N
    data = y(i_y) * sinc((ts/t - i_y/N) * f_s/100);
    plot(data);
    z = z + data';
end
hold off;
title("1.d) Per-sample sinc functions")
```



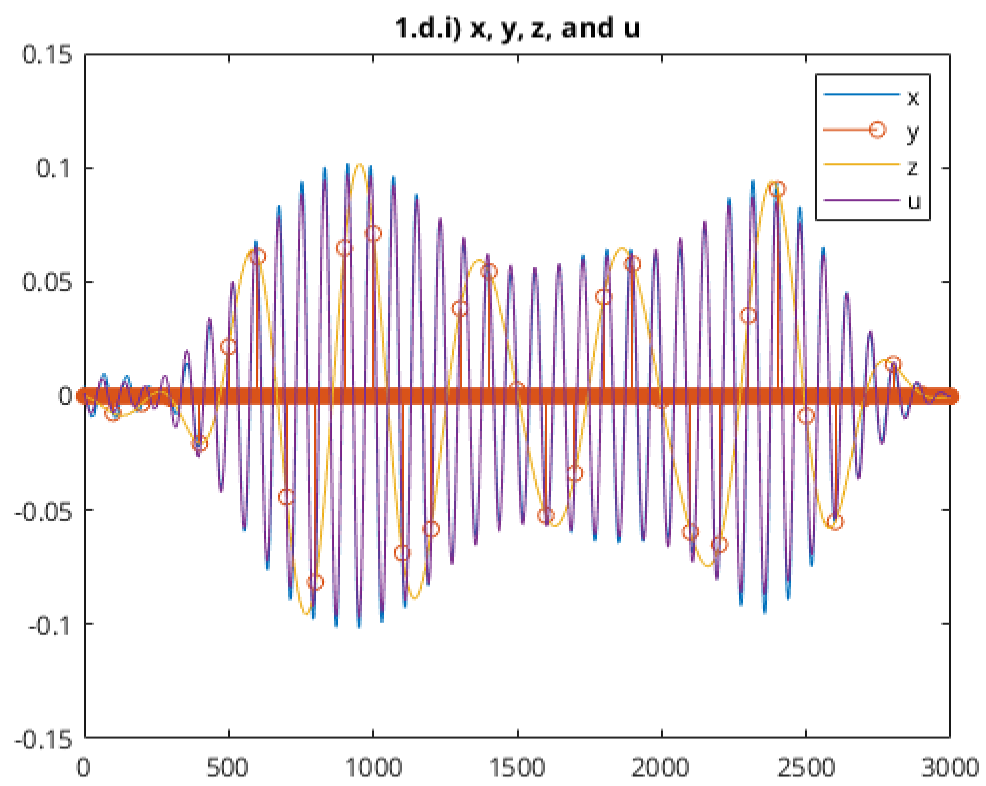
Generate another band-pass filter for 30-45Hz, apply to y to reconstruct as u. Multiply by 100 to compensate for energy lost during sampling.

```
f1 = 30;
f2 = 45;
% 3 = 3rd order filter
% 30 = -30dB for frequencies outside the range
[b, a] = cheby2(3, 30, [f1 f2]/(f_s/2));
u = 100 * filtfilt(b, a, y);
```

1.d.i) Comparisons

Plot x, y, z, and u on top of each other in one figure

```
figure;
plot(x);
hold on;
stem(y);
plot(z);
plot(u);
legend("x", "y", "z", "u");
title("1.d.i) x, y, z, and u")
hold off;
```

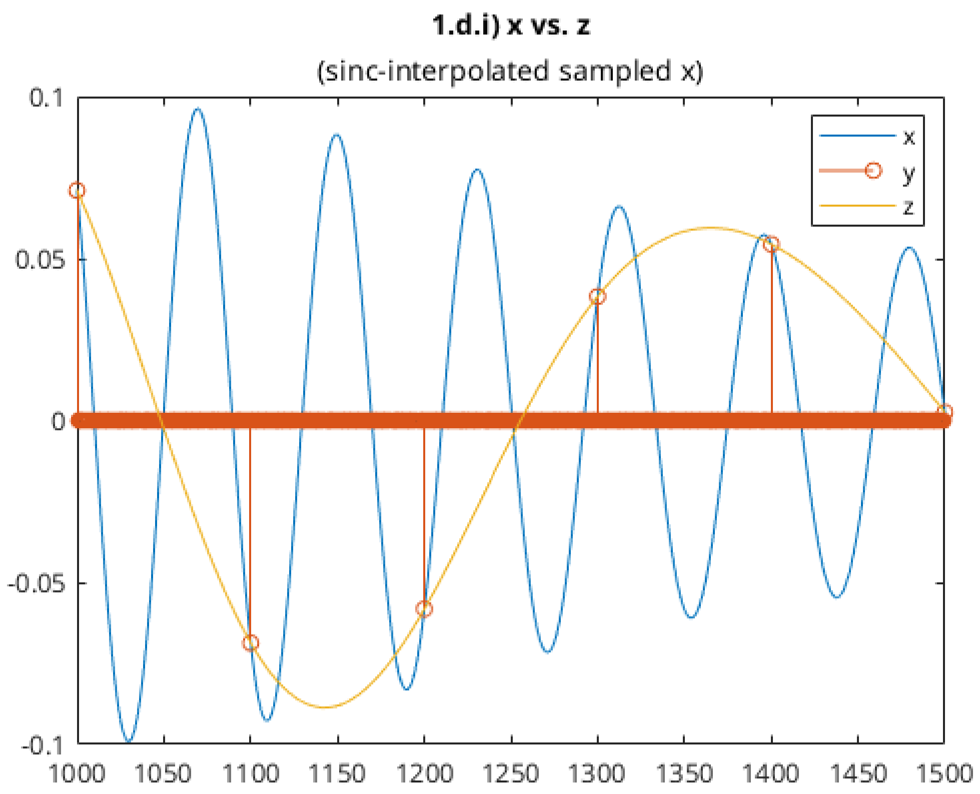


1.d.i) x vs z

x and z are not equal - sinc interpolation can only reconstruct frequencies up to $f_{sample}/2 = 30/2 = 15Hz$, but x contains higher-frequency data.

```
figure;
plot(x);
hold on;
stem(y);
plot(z);

legend("x", "y", "z");
xlim([1000 1500]);
```

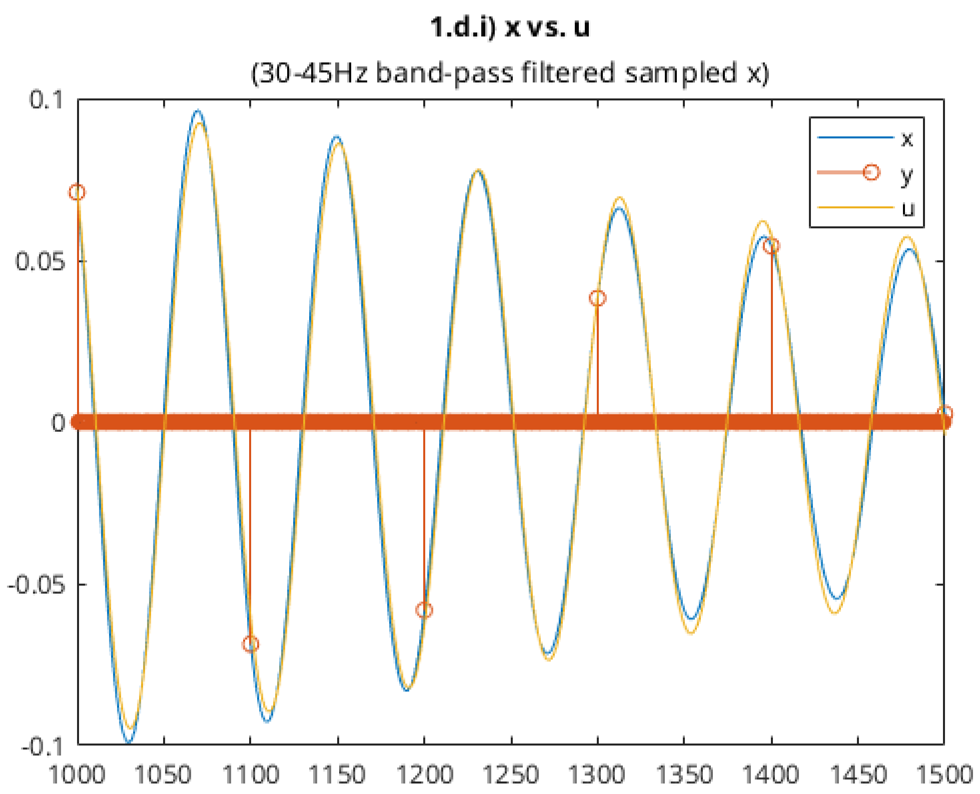
```
title("1.d.i) x vs. z", "(sinc-interpolated sampled x)");
hold off;
```



1.d.i) x vs u

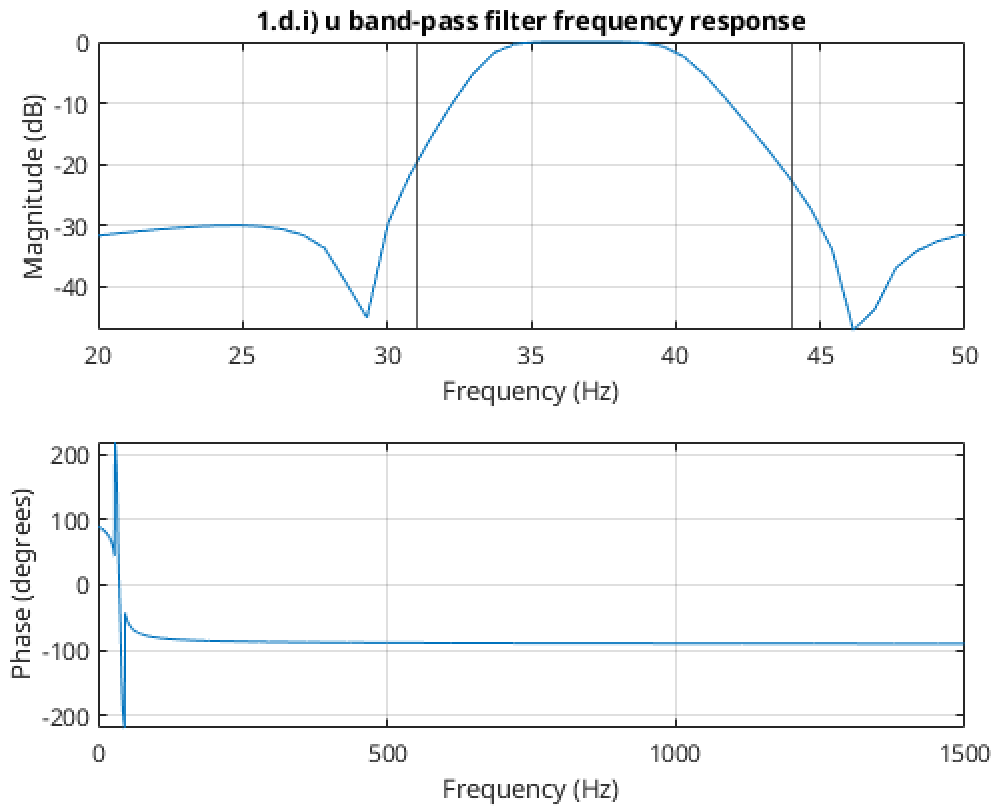
x and u are similar, but not equal. I think this is because the band-pass filter is not perfect.

```
figure;
plot(x);
hold on;
stem(y);
plot(u);

legend("x", "y", "u");
xlim([1000 1500]);
title("1.d.i) x vs. u", "(30-45Hz band-pass filtered sampled x)");
hold off;
```



Frequency response of the second band-pass filter. Note that the values for the 31-44Hz range (used to generate x) drop to around -20dB. These frequencies are unfairly reduced when generating u, so it doesn't match x exactly.

```
f1 = 30;
f2 = 45;
[b, a] = cheby2(3, 30, [f1 f2]/(f_s/2));
figure;
freqz(b,a,2048,f_s);
xlim([20, 50]);
xline(31);
xline(44);
title("1.d.i) u band-pass filter frequency response");
```



1.d.ii)

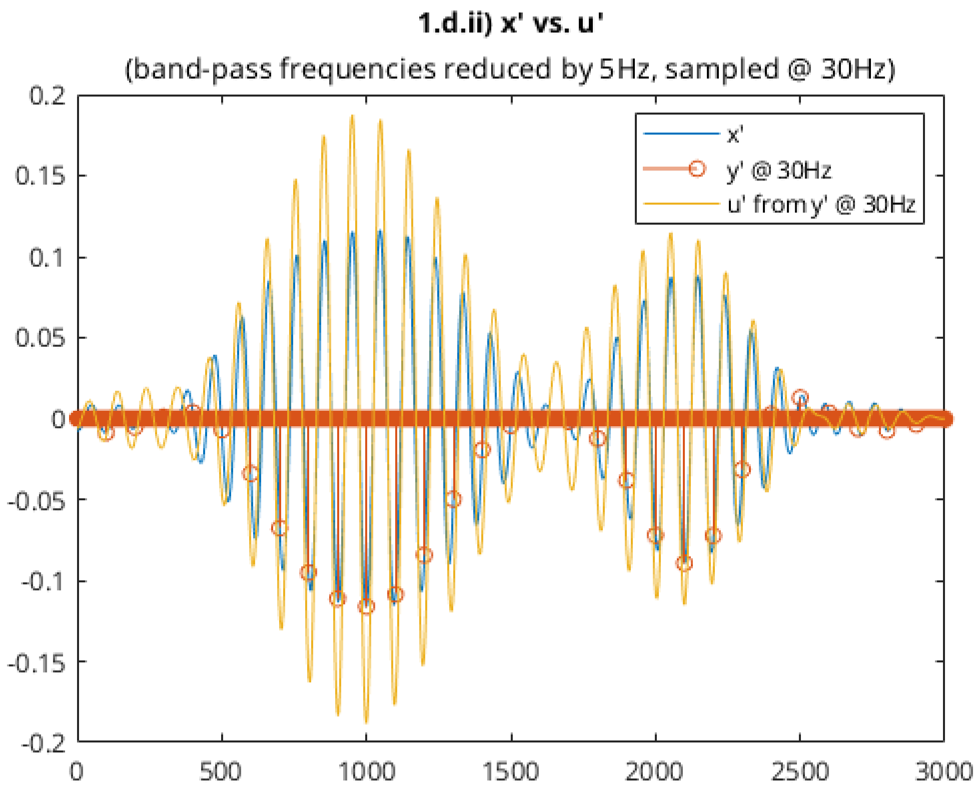
Q: Why does the reconstructed waveform differ much more from the original if you reduce the cut-off frequencies of all band-pass filters by 5 Hz?

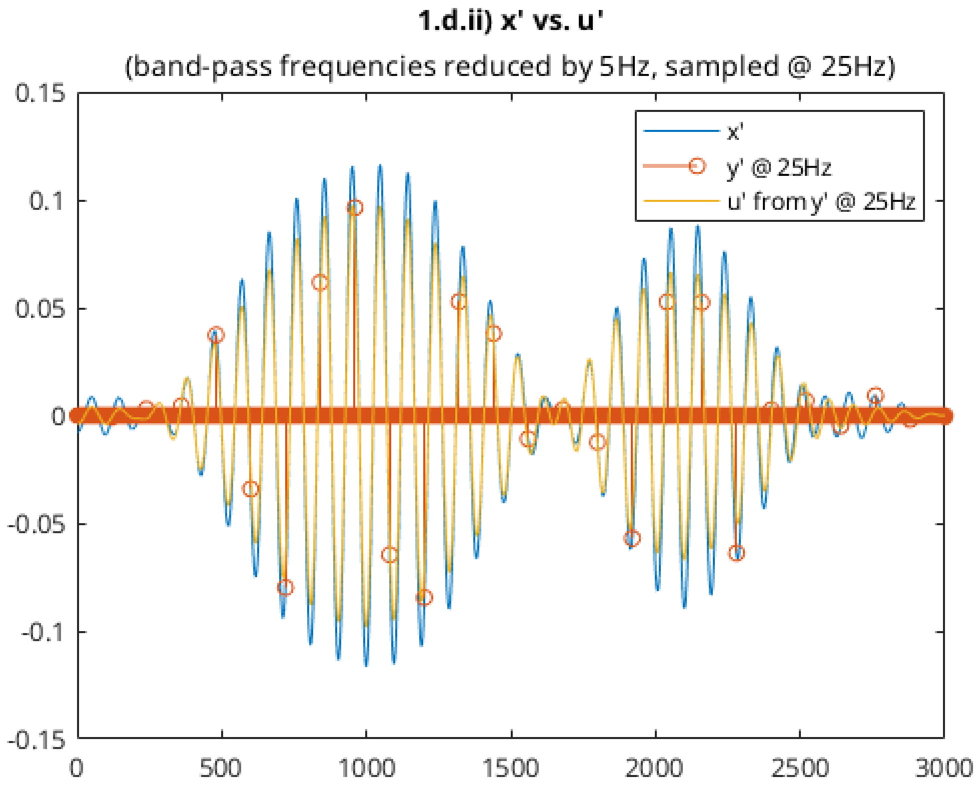
The question specifically states that the band-pass filters change, but does not state that the sampling frequency for y changes from 30Hz.

When the band-pass frequencies change, x's frequency range is 26-39Hz. The sampling frequency for y, 30Hz, is within this range. This causes aliasing, which distorts the frequency domain representation (like in slide 58) and thus distorts the output of the low-pass filter.

Before, x's range was 31-44Hz, and f_y = 30Hz was outside this range, so there wasn't a problem. With the decreased frequencies we can also resolve the problem by decreasing f_y to 25Hz, and the final figure shows this fixes the issue.

```
% Find x' = x but frequencies reduced by 5
f1 = 31-5;
f2 = 44-5;
[b, a] = cheby2(3, 30, [f1 f2]/(f_s/2));
x_prime = filtfilt(b, a, r);

% Find y'_30 = sampling of x' @ 30Hz
y_prime_30 = x_prime;
for i = 1:99
    y_prime_30(i:100:end) = 0;
end
% Find y'_25 = sampling of x' @ 25Hz
y_prime_25 = x_prime;
for i = 1:119
    y_prime_25(i:120:end) = 0;
end

% Find u' = reconstruction of x' from filter of y' with freqs reduced by 5
f1 = 30-5;
f2 = 45-5;
[b, a] = cheby2(3, 30, [f1 f2]/(f_s/2));
u_prime_30 = 100 * filtfilt(b, a, y_prime_30);
u_prime_25 = 100 * filtfilt(b, a, y_prime_25);

% Plot all
figure;
plot(x_prime);
hold on;
stem(y_prime_30);
plot(u_prime_30);
legend("x'", "y' @ 30Hz", "u' from y' @ 30Hz");
title("1.d.ii) x' vs. u'","(band-pass frequencies reduced by 5Hz, sampled @ 30Hz)");
hold off;

figure;
plot(x_prime);
hold on;
stem(y_prime_25);
plot(u_prime_25);
legend("x'", "y' @ 25Hz", "u' from y' @ 25Hz");
title("1.d.ii) x' vs. u'","(band-pass frequencies reduced by 5Hz, sampled @ 25Hz)");
hold off;
```





Published with MATLAB® R2021b