EE369C: Assignment 1

Due Thursday Sept. 24th

Assignments This quarter the assignments will be partly matlab, and partly calculations you will need to work out by hand. I encourage you to typeset your solutions in LaTeX. Convert the output to a pdf file, turn it is as described below. Your solution is due sometime the night of the due date.

I will put the latex source code on the web site, so you can splice your answers directly in. This is a nice way to keep the problems and the solutions together. A good introduction and reference for LaTeX is the original book, available in the bookstore, or online here:

`http://en.wikibooks.org/wiki/LaTeX`

Links for popular Mac and PC distributions are available on the course web site, on the assignments page.

One approach for your matlab plots is to generate the figure, and then use the command line

```
>> print -dpdf myplot.pdf
```

This makes sure you don't have a rasterized image. This will be centered on a full page, so use a pdf viewer to trim it to fit. On a mac, Preview does this. First select the plot with the "select" tool, the use the "crop" command, found under the "Tools" menu. You can add annotation in matlab if you'd like, or another program.

The default matlab plot linewidth is 0.5 pt, which almost disappears. You can change this in the plot window by choosing "Axes Properties" under the "Edit" menu. Click on the line you want to change, and you'll get a set of controls to change the width, color, and other properties. A width of two points shows up well.

For the matlab problems, if you are asked to write an mfile, include the code in your solution. This will probably be very short. Include the requested plots, along with the listing of how the plot was generated. Use subplots to save paper.

Turning in Your Assignment Email me your solutions. Include your name, the assignment number, and the course number in the subject line. For example, I would include

Subject: John Pauly, Assignment 1, EE369C

That way I can easily find your assignments, and keep track of them.

Introduction This assignment concerns sampling and reconstruction of band limited signals. We'll start with a simple signal that will be important in about two weeks. Define the sampled sequence by

```
>> d = [zeros(1,10) [10:-1:1] 0 [1:10] zeros(1,10)]
>> x = [-20:20];
```

and the plot it

```
>> subplot(211)  % makes it fit better in your assignment
>> stem(x,d);
>> xlabel('x');
>> print -dpdf plot1.pdf
```

The result should look like Fig. 1.

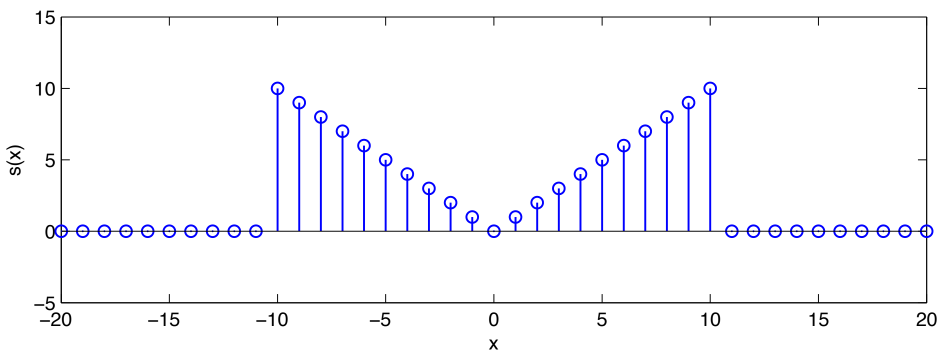

Figure 1: Stem plot of the test signal

1. Sinc Interpolation The first task it to write an m-file to perform sinc interpolation on a sampled signal. We will take the test signal, and upsample by a factor of 10. We will consider this to be the gold standard, the signal we will try to recover in the subsequent problems. Write an m-file

```
function di = sinc_interp(d,x,xi)
%
% inputs
%     d -- uniformly sampled data points, spaced by 1
%     x -- uniform sample locations
%    xi -- locations to evaluation for the sinc interpolation
% outputs
%    di -- since interpolated values at locations xi
```

The matlab sinc(x) function is useful here, it takes a vector x and returns a vector sinc(x). We will assume that the uniform sample spacing is 1 for convenience.

Then, define a fine grid to use for evaluation

```
 >> xi = [-20:0.1:20];
 >> di = sinc_interp(d,x,xi);
 >> subplot(211);
 >> plot(xi,di); hold;
 >> stem(x,d)
 >> print -dpdf plot2.pdf
```

The result should look something like Fig. 2. Include your m-file, and your plot in your assignment.

Solution

With this sinc_interp function, we are able to interpolate the discrete ramp signal (Fig. 3).

```
function di = sinc_interp(d,x,xi)
```

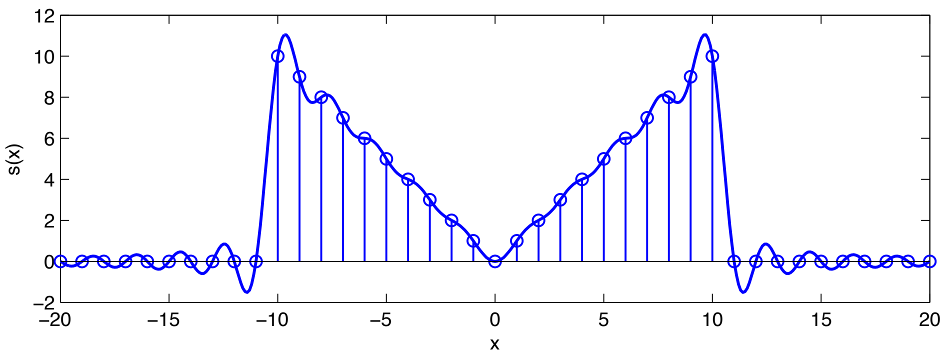

Figure 2: Original data, and the sinc interpolation.

```
X = x(2)-x(1);
di = zeros(size(xi));
for i = 1:length(d)
    di = di + d(i)*sinc((xi-x(i))/X);
end
```

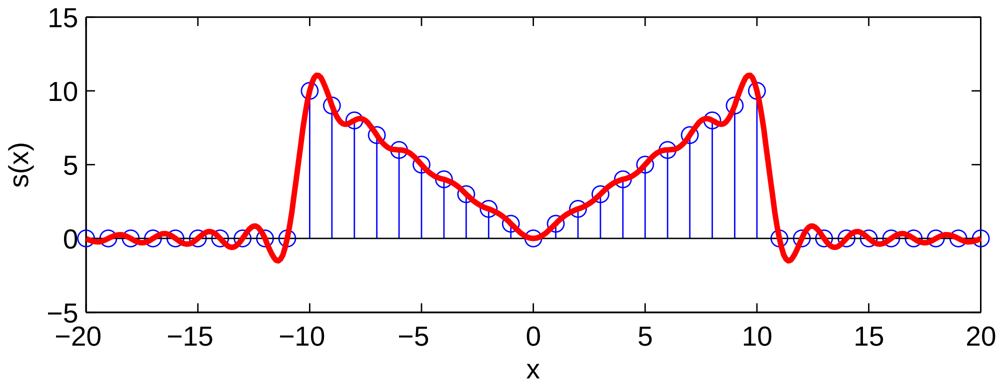

Figure 3: We have the same sinc interpolation.

Non-Uniform Sampling Next, we will look at the case where the sampling is not uniform. In class we set this up as a matrix equation. If we knew the uniform samples, we could sinc interpolate to solve for the nonuniform samples

$$s(x_n) = \sum_{m=1}^{M} s(x_m)\,\mathrm{sinc}\left(\frac{x_n - x_m}{X}\right)$$

where x_n are the nonuniform samples, and x_m are the uniform samples. The matrix equation was then

$$\underline{s}_n = E\underline{s}_u$$

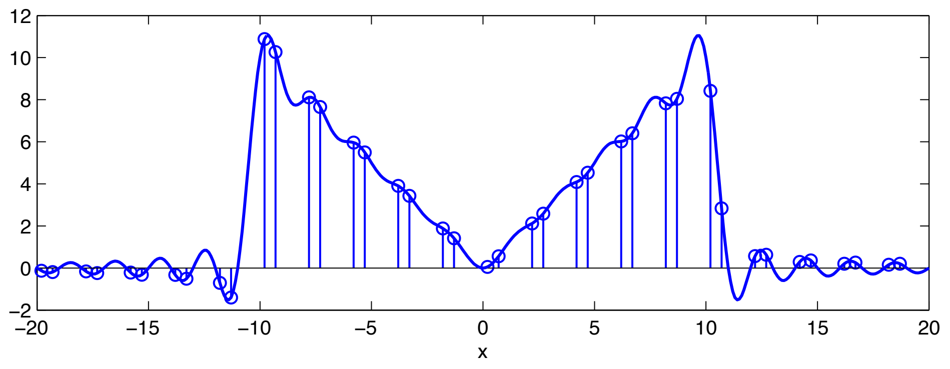

Figure 4: Bunched samples, and the original sinc interpolation. Note that we've completely missed the peak ripple on the right.

were $\underline{s}_n$ is an N length vector of the non-uniform samples, and $\underline{s}_u$ is an M length vector of the uniform samples, and E is a matrix of the sinc coefficients. We can solve for the uniform samples with the pseudo inverse

$$\underline{s}_u = (E^* E)^{-1} E^* \underline{s}_n$$

Write an m-file that does this. Note that in matlab, you can perform this operation more quickly and accurately using the "\" operator.

```
function du= sinc_resample(dn,xn,xu)
%
% inputs
%     dn -- non-uniformly sampled data points
%     xn -- non-uniform sample locations
%     xu -- uniform sample points, spaced by 1
% outputs
%    du-- uniformly sampled data
```

Again, for convenience, assume the uniform sample spacing is 1, and that this corresponds to the Nyquist rate for the underlying signal. Include a listing of your m-file.

To test this, we'll look at two test cases to see how this works.

2. Bunched Samples the first example is of bunched samples. We will generate the bunched sample data by subsampling our sinc interpolated data. Generate the data and plot it,

```
>> db = [di(3:20:end) di(8:20:end)]
>> xb = [xi(3:20:end) xi(8:20:end)]
>> stem(xb,db);
>> hold
>> plot(xi,di)
```

The samples aren't in order, but you can sort them if you'd like. This should look like Fig. 3. Use your m-file to see if you can recover the original samples (Fig. 1) from the bunched samples. There are only 40 samples in `db` and 41 in `d`, so this problem is underdetermined. You can use only 40 samples of `d`, or use the matlab \ operator which does the right thing here.

Include a plot of the uniformly sampled data. With any luck, it should look like Fig. 1.

Solution

Using `sinc_resample`, we are able to recover the original samples from the bunched samples (Fig. 5).

```
function du = sinc_resample(dn,xn,xu)

X = xu(2)-xu(1);
E = zeros(length(xn),length(xu));
for i = 1:length(xu)
    E(:,i) = sinc((xn-xu(i))/X);
end
du = E\dn(:);
```

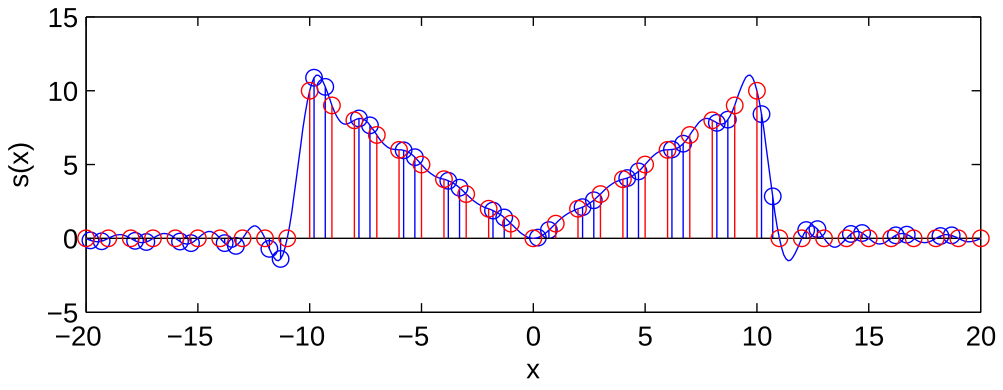

Figure 5: Bunched sampling – the recovered uniform samples are in red.

3. Random Samples In the case of bunched sampling, there are actually explicit solutions for the interpolators (See ”The Fourier Transform and its Applications” by Ron Bracewell for one derivation). It gets more interesting with random samples, in which case each sample has its own unique interpolator. This is what we will look at next.

First, we need to generate random samples. We'll choose random samples from the sinc interpolated signal

```
>> ndx = unique(randi(length(di),1,50));
>> dr = di(ndx);
>> xr = xi(ndx);
>> stem(xr,dr)
>> hold
>> plot(xi,di)
```

The unique() function sorts ndx, and removes duplicates. Make sure that you have at least 41 samples left! If not, generate another set of points.

Include your plot. See if you can recover the uniform samples from this data.

Solution
We are able to perfectly recover the uniform samples from the noiseless random samples (Fig. 6).

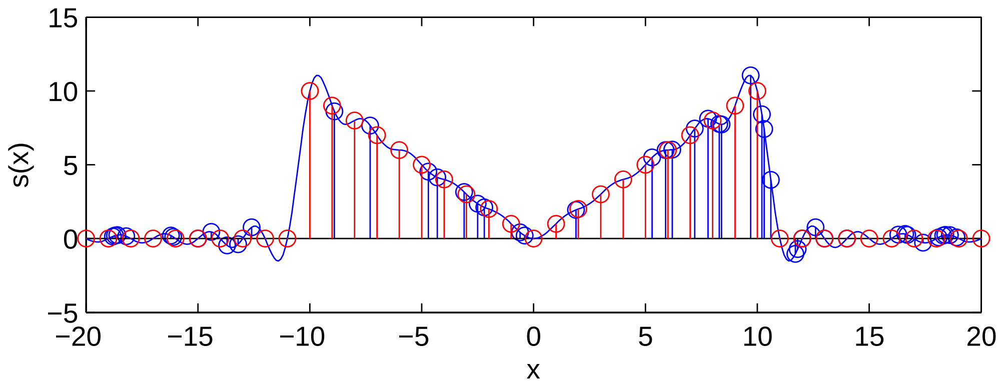

Figure 6: Random sampling – the recovered uniform samples are in red.

4. Interpolators We can generate what the interpolators look like for your specific random pattern by using an impulse input. For example, for the 15th sample, we generate an input

```
>> dp15 = zeros(1,length(dr));
>> dp15(15) = 1;
>> dp15u = sinc_resample(dp15,xr,x)
```

This gives us the uniformly sampled values for this interpolator, which we can then sinc interpolate and plot.

```
>> stem(xr,ones(1,length(xr))
>> hold
>> plot(xi,sinc_interp(dp15u,x,xi))
```

This compares the interpolator for the 15th sample to impulses at all of the random samples. Pick a couple of interesting samples, and plot the results. Do they do what you expect?

There are only two interpolators for the bunched samples case. Make a plot showing any two of these, along with the stem plot of the sample locations, as we did above for the random samples.

Solution
The interpolators for the 15th and 20th samples are shown in red and green, respectively, in Fig. 7. Of note, the interpolators can be quite large in regions where there is a large spacing between samples. Also, the interpolators are not necessarily 1 at the impulse and 0 at the other random samples. This is because the interpolators cannot go through 0 at all sample locations (the problem is overdetermined with 50 random samples locations to satisfy by only 41 uniform locations).

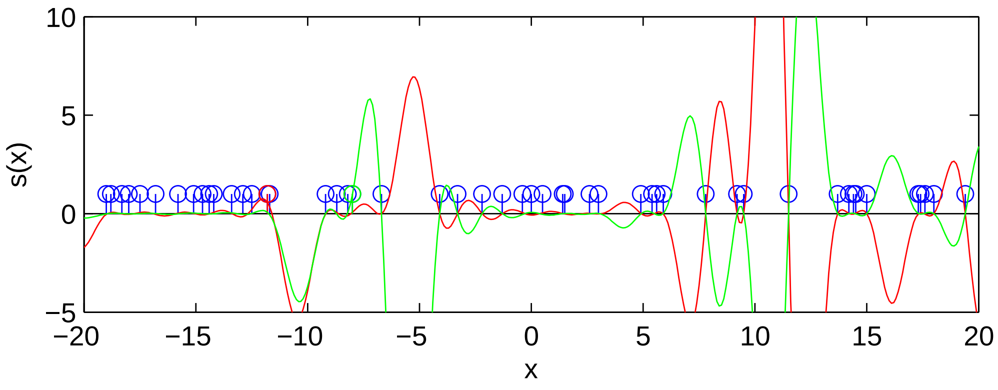

Figure 7: Random sampling interpolators.

There are only two interpolators for the bunched sampling – a left and right interpolator for each bunched pair (Fig. 8). Note that they are mirror symmetric and that these do go through 1 at the impulse and 0 at the other bunched samples. Because the bunched pairs are periodic, we can expect the interpolators to be the same at each bunched pair.

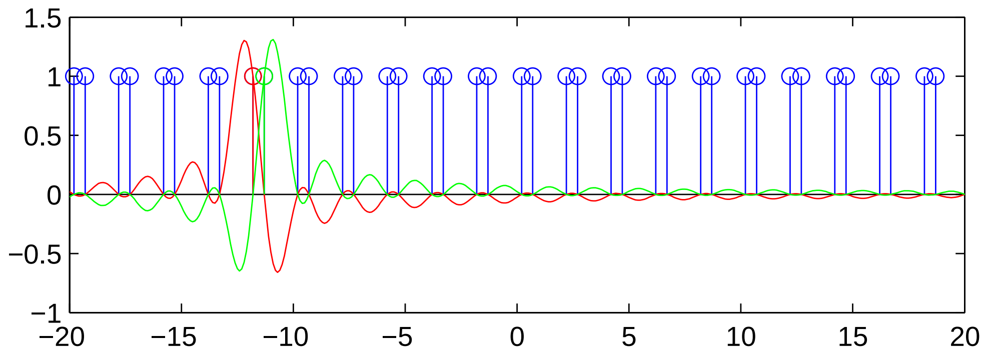

Figure 8: Bunched sampling interpolators.

5. Signal Noise and Sample Timing Jitter Physical systems cannot sample a signal perfectly. We will consider the effect of error in the sampled values (noise) and in the sampling time (jitter). First we will consider sample noise. Add noise (stdev = 0.25) to the randomly spaced samples. How good is the reconstruction if there are 50 samples, as in the previous section?

```
drn = dr + 0.25*randn(size(dr));
du = sinc_resample(drn,xr,x);
figure, plot(xi,di);  % ground truth
hold; stem(x, du);    % uniform samples
```

```
plot(xr,drn,'r.');     % random samples, with noise
```

Does doubling the number of samples to 100 samples help? Do this by adding another 50 random samples. Include plots in your report to support your conclusion.

Now consider the effect of uncertainty in the timing of the sampling. Add an error of stdev = 0.05. How good is the reconstruction if there are 50 samples?

```
xrn = xr + 0.05*randn(size(xr));
du = sinc_resample(dr,xrn,x);
figure, plot(xi,di);  % ground truth
hold; stem(x, du);    % uniform samples
plot(xrn,dr,'r.');    % random samples, with jitter
```

Does doubling the number of samples to 100 samples help here? Again, include plots to support your conclusion.

Solution

For both the noise and jitter cases, the uniform sample recovery with 50 samples is generally very poor, especially in regions between samples that are spaced far apart. This can be expected since the interpolators have very large magnitude and any noise is magnified in these regions so that the large interpolators don't combine perfectly to recover the signal. Jitter causes a random shift of the interpolators, so again the large magnitude of the interpolators cannot combine perfectly and result in large error. Both noise and jitter improve significantly at 100 samples since the errors tend to average out to the correct value. The following figures show 'typical' cases, although sometimes the recovery may be far worse or far better depending on the location of the random samples.

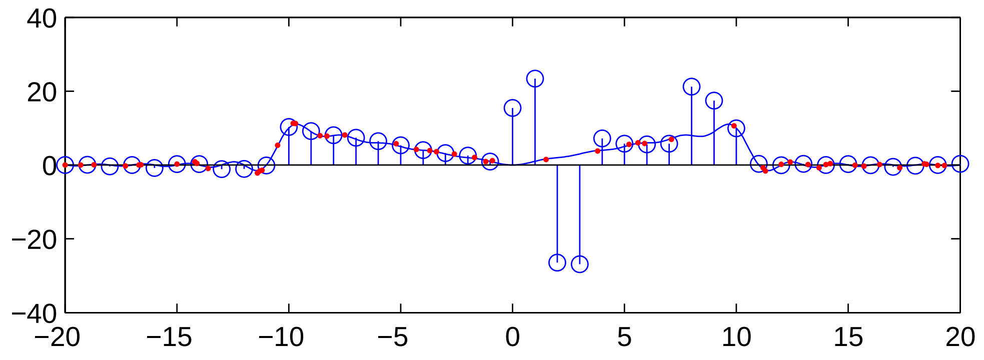

Figure 9: With noise - 50 samples

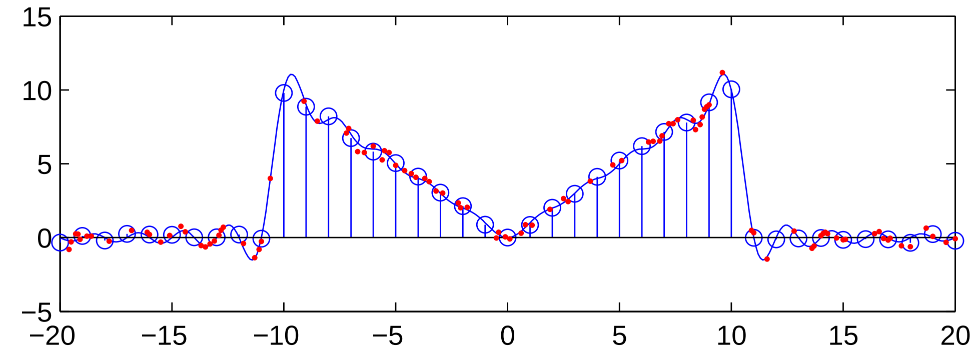

Figure 10: With noise - 100 samples

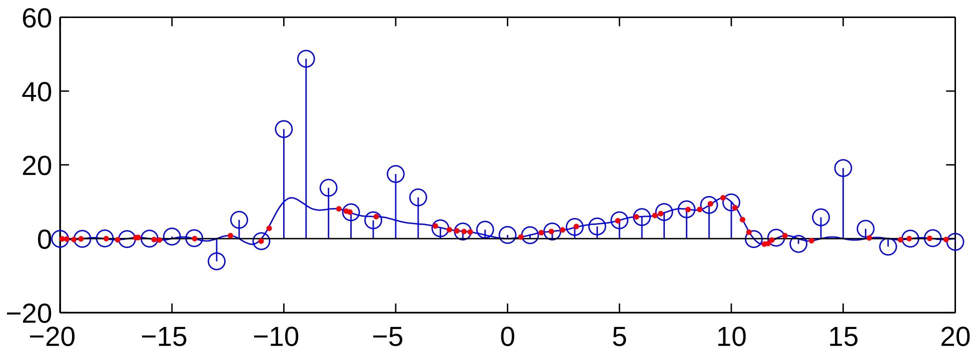

Figure 11: With jitter - 50 samples

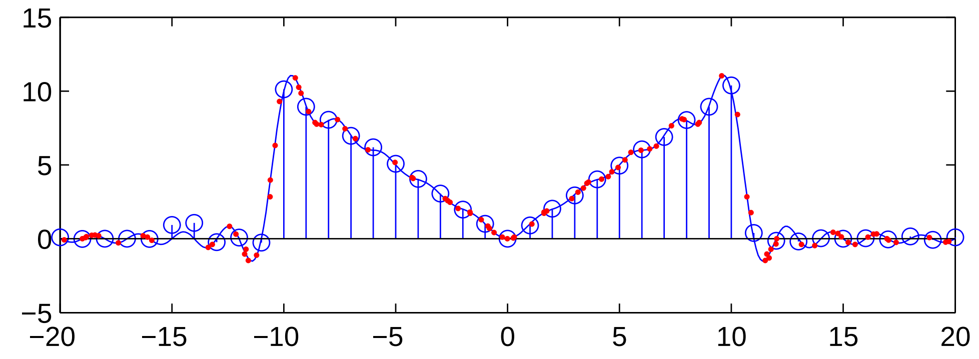

Figure 12: With jitter - 100 samples