On my honor, I have neither given nor received any aid on this work, nor am I aware of any breach of the Honor Code that I shall not immediately report.

Pledged: _____

Print Name: _____

Work together with your group to complete this project. This project is due Thursday, September 27, at 5 PM.

In our last class we discussed a method of creating the auditory equivalent of the *Penrose staircase* illusion.

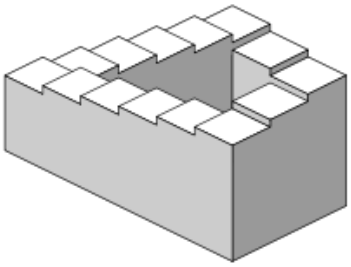

Figure 1. Penrose stairs.

An auditory equivalent of this illusion would be a sound that seemed to fall (or rise) in pitch constantly, yet never "go anywhere," whether to very low pitches or very high ones. The idea we hit upon in class was to use a sequence of overlapping chirps – either exponential or linear – to achieve the illusion. The following graph illustrates the concept.

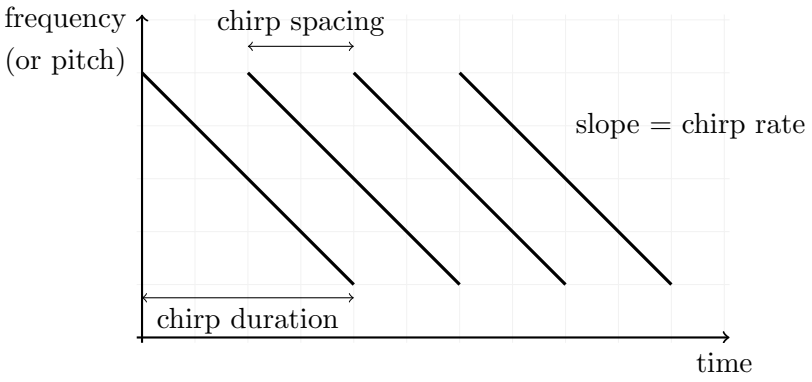

Figure 2. Conceptual time-frequency plot for the Risset glissando.

The individual chirps will be enveloped, in order to "mask" their entry and exit in the overall sound. Jean-Claude Risset, a French composer who did pioneering work in computer-generated music, was the first to use this idea. The resulting sound, known as a *Risset glissando*, is named after him. In this lab, we will use MATLAB to create a Risset glissando.

1. We begin with an almost-complete MATLAB script to compute a Risset glissando. This code printed below is contained in the file `risset1.m`. (The printout below only shows the main computations; the input and output sections are not shown.) The main input parameters for this script are as follows:

   - `f0`: start frequency for the chirp.
   - `rho`: chirp rate, in octaves per second.
   - `chirpDuration`: duration of an individual chirp in the glissando, in seconds.

- `chirpSpacing`: the time between successive chirps in the glissando, in seconds.
- `numChirps`: the total number of chirps in the glissando.
- `riseFrac`: the fraction of the chirp duration occupied by the rising edge of the envelope. This should be between 0 and 1.
- `fallFrac`: the fraction of the chirp duration occupied by the falling edge of the envelope. This should be between 0 and 1.

The basic idea of the script is straightforward: based on the input parameters, compute the basic chirp for the glissando. Then create an array to hold the entire glissando, and add the chirp into the array at the correct locations. Our code will use MATLAB's *array slicing* capabilities to do this.

```
36 %% Computations
37
38 % create a single exponential chirp
39 sChirp = expChirp(...
40
41 % envelope it
42 tauRise = riseFrac * chirpDuration;
43 tauFall = fallFrac * chirpDuration;
44 sChirp = cubicEnvelope(sChirp, tauRise, 0,  0, tauFall,  0,  0, fs);
45
46 % get the number of samples in the chirp
47 nChirp = length(sChirp);
48 % compute the number of samples between successive copies of the chirp
49 nSkip = ...
50 % compute the total number of samples required
51 nTot = ...
52 % time spacing of the samples
53 dt = 1/fs;
54 % compute the total duration of the glissando
55 T = ...
56
57 % initialize the sample array
58 s = zeros(1, nTot);
59
60 % loop to create the samples
61 for k = 1 : numChirps
62     nStart = ...
63     nEnd = nStart + nChirp - 1;
64     s(nStart : nEnd) = s(nStart : nEnd) + sChirp;
65 end;
66
67 % normalize to unit amplitude
68 s = s / max(abs(s));
69
70 % (more code to plot and play the sound...)
```

Finish writing this MATLAB script by completing lines 39, 49, 51, 55, and 62 of the code.

2. Once your script file is running correctly, you can be confident your code is *correct*. To make your code really *useful*, you need to make a MATLAB function the performs the essential computations. Here is the signature of such a function (the code is contained in the file `expRisset.m`):

```
1  function s = expRisset(f0, rho, chirpDuration, chirpSpacing, numChirps, riseFrac, fallFrac, fs)
2  %    function s = expRisset(f0, rho, chirpDuration, chirpSpacing, numChirps, riseFrac, fallFrac, fs)
3  %
4  % This function creates a basic Risset glissando.  The glissando is built
5  % out of identical overlapping exponential chirps.  The user may specify
6  % the initial frequency of the chirp, its chirp rate, and the duration of
7  % the chirp.  The user also specifies the time gap between adjacent chirps
8  % in the glissando, the total number of chirps in the glissando, and the
9  % rise time and fall time of the envelopes on the individual chirps.
10 %
11 % Inputs:
12 %   f0: the initial frequency of each chirp, in Hertz.
13 %   rho: the chirp rate, in octaves per second.
14 %   chirpDuration: the duration of each chirp, in seconds.
15 %   chirpSpacing: the time gap between successive chirps, in seconds.
16 %   numChirps: the total number of chirps in the glissando.
17 %   riseFrac: the fraction of the chirp duration taken up by the rising
```

```
%     edge.  Should be between 0 and 1.
%   fallFrac: the fraction of the chirp duration taken up by the falling
%     edge.  Should be between 0 and 1.
%   fs: the sampling rate, in Hertz.
%
% Output:
%   s: The array of samples from the Risset glissando.
```

Notice that this function returns as output an array containing the samples from the Risset glissando; i.e. exactly the array you calculated in `risset1.m`. Use the code from `risset1.m` to complete the `expRisset` function above. Test your `expRisset` function by running the script `expRissetDemo.m`.

3. Write a function called `linRisset` that creates a Risset glissando using a linear chirp rather than a exponential one. Model your `linRisset` function on the `expRisset` function you just wrote. Make sure to comment the new function correctly. Also, write a MATLAB script called `linRissetDemo.m` which tests the `linRisset` function.

4. Find input parameters that make the exponential Risset glissando as convincing as possible as an auditory illusion. Do the same for the linear Risset glissando. Which of these do you prefer?

5. Write a brief report on this lab, using the LaTeX template provided.