CS1112-206/211/212–Fall 2009

Lab 12 Solution

Friday, November 20

(i) If `rate` is multiplied by 1.5, then the sound will be played in time $T/1.5$, where T is the duration of the original sound. It will also have a higher pitch when listening (like a girl talking). Similarly, if `rate` is divided by 1.5, then the sound will be played in time $T * 1.5$, and it will also have a lower pitch when listening (like an old man talking).

(ii) Since the user indicates the locations of the end of the chimes and the beginning of the gong, we can simply zero this part of the vector to obtain silence.

(iii) To reverse the sound, simply play it from the end to the beginning. That is, reverse the sound vector so that the last component becomes first, and the first becomes last. This can be done using a `for` loop or a vectorized code.

(iv) Since the user indicates the location of the beginning of the gong, and we can assume that the end of the sound file is the end of the gong, we can determine the length of the gong. The halfway point of the gong is exactly half the length. Let BG be the beginning of the gong and M be the halfway point of the gong. Then the first half of the gong is `OneOClock(BG:M)`. We just have to play this three times by calling `sound` three times.

(v) Once the user indicates the end of each four notes (there are three of them (the last one is the end of the gong the user indicated earlier); let these locations be e_1, e_2, e_3), we just have to play the sound from the beginning to each of these locations. For example, at the half hour, we play `OneOClock(1:e2)`. We pause using the `pause` function.

(vi) To make a vector of continuous and repetitive sound, we first need to determine the length of the final output. Since the duration is two minutes, i.e., 120 seconds, and for each second, `rate` values are played, we need to set up a vector of length

$$rate * 120 + 1$$

(we add one to obtain the precise duration). Then, we fill in the vector, one gong at a time. If there is enough space to fill in, then simply use vectorized code to assign the values. For the ith iteration of filling in, the starting location is $(i-1) * lenGong + 1$, where *lenGong* is the length of the gong vector. We repeat this

$$numReps = \left\lfloor \frac{\text{round}(rate * 120) + 1}{lenGong} \right\rfloor$$

times, which is the number of iterations we can fill in the whole gong completely.

Now, we have to fill in the last part of the sound vector, which will be a partial gong. Since we have filled in *numReps* iterations, each with *lenGong* locations, we have filled in $numReps \cdot lenGong$ locations. Now we still have

$$(rate * 120 + 1) - numReps \cdot lenGong$$

locations left. We simply extract the first $(rate * 120 + 1) - numReps \cdot lenGong$ locations of the gong and pad it to the resuling vector, starting at location $numReps \cdot lenGong + 1$. We can verify that the lengths of these two vectors are the same.

The code from the above procedure is given below:

```
lenTwo=rate*120+1;
TwoMinuteGong=zeros(lenTwo,1);
numReps=floor(lenTwo/lenGong);
for i=1:numReps
    TwoMinuteGong((i-1)*lenGong+1:i*lenGong)=Gong;
end

TwoMinuteGong(numReps*lenGong+1:lenTwo)=Gong(1:lenTwo-(numReps*lenGong));
```