EP1 de Computação Musical – Entrega: 23/4/2017 até 23:55 pelo PACA

Entendendo o processamento de fluxos de áudio

Entrega: 23/4/2017 até 23:55 pelo PACA

1 Introdução: Processamento de fluxos de áudio em tempo real

O principal objetivo desse primeiro exercício é nos familiarizarmos com a ideia de processamento de fluxos de áudio em tempo real. Tal processamento é estruturado em torno da ideia de "blocos", que são segmentos de N amostras consecutivas dos fluxos de áudio. A ideia principal é que a cada bloco, todos os objetos que consomem ou produzem fluxos de áudio em um fluxograma (ou patch na nomenclatura do Pd) são ordenados de acordo com suas posições relativas e seus processamentos são escalonados e processados de acordo com essa ordem (não há paralelismo, numa conexão como `[osc~]`→`[dac~]` primeiro o oscilador produz todo o seu bloco, e só depois disso o dac envia esse bloco para a placa de áudio). Esse é o chamado *ciclo DSP*, e ocorre a cada N/T segundos, onde T é a taxa de amostragem.

Muitos processamentos são sensíveis ao parâmetro N que define o tamanho de bloco. Em um dos casos extremos, N=1, cada nova amostra força a ordenação dos objetos de áudio, o que aumenta a exigência computacional do patch como um todo; no outro extremo, valores muito grandes de N dificultam um acoplamento estreito entre os objetos que processam os blocos, e introduzem latência nos resultados da computação (afinal o primeiro bloco começará a ser processado apenas após N amostras ou N/T segundos). Além disso, muitas informações extraídas a partir da análise de blocos produzem resultados muito discrepantes entre blocos pequenos e blocos grandes, ou entre sinais com frequências muito baixas ou muito altas, como veremos nesse EP.

Para entendermos a influência do N no processamento de fluxos, usaremos um esqueleto de patch inicial para nossas implementações, que já possui os objetos necessários para abrir arquivos de áudio, selecionar o tamanho de bloco no subpatch de análise, bem como gerar alguns gráficos dos cálculos que faremos. Procure entender essas construções, abrindo todos os patches de ajuda desses objetos fornecidos (use o botão direito do mouse para isso).

2 Armazenando blocos e sincronizando cálculos ao ciclo DSP

A primeira coisa que iremos fazer é guardar os blocos de áudio que chegam da entrada em um objeto `[array]`, usando o objeto `[tabwrite~]`. Crie esses objetos e abra o patch de ajuda deles; isso deverá elucidar o seu uso. Aproveite para ler também os exemplos que vêm dentro da ajuda do array, sobre como calcular somas e máximos, e sobre como redimensionar o tamanho do array (para permitir as mudanças de tamanho de bloco). Para sincronizar a transferência do bloco para o array a cada ciclo DSP, utilize o objeto `[bang~]`.

3 Medindo a amplitude do sinal

Implementaremos duas medidas diferentes de amplitude para o sinal da entrada, a *amplitude de pico* e a *amplitude rms*, definidas para o bloco $x = (x_0, x_1, \ldots, x_{N-1})$ como

$$\begin{aligned} \text{amp\_pico}(x) &= \max\{|x_n| \mid n = 0, 1, \ldots, N-1\}, \text{ e} \\ \text{amp\_rms}(x) &= \sqrt{\frac{\sum_{n=0}^{N-1} |x_n|^2}{N}} \end{aligned}$$

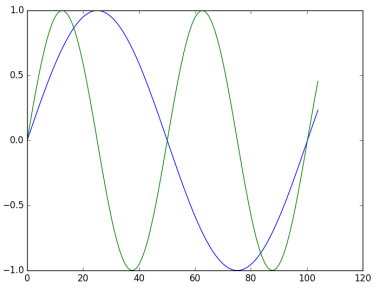
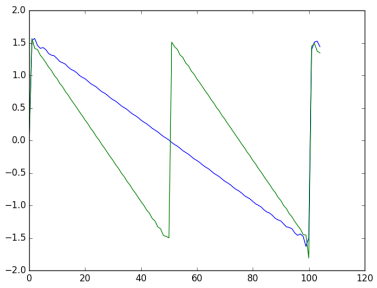
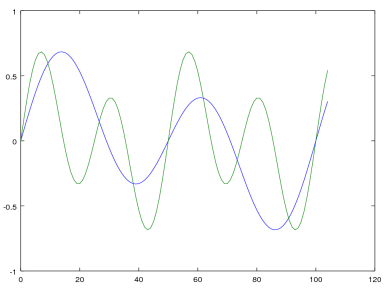
Note que essas medidas são geralmente diferentes: como exemplo, uma senoide da forma $\text{sen}(2\pi ft)$ possui amplitude de pico igual a 1, e amplitude rms igual a $\sqrt{1/2} \approx 0.707$.
Para verificar a corretude dos valores obtidos pelas nossas estimativas, iremos comparar suas saídas com aquela dos objetos `[cyclone/peakamp~]` (para a amplitude de pico) e `[zexy/envrms~]` (para o rms). Esses objetos já estão inseridos no esqueleto fornecido para a implementação de vocês.

4 Medindo a "velocidade" do sinal: Zero-Crossing Rate (ZCR)

ZCR é um descritor de áudio que calcula a taxa de cruzamentos do sinal de áudio pelo valor de amplitude nula (o eixo horizontal do gráfico do sinal). Para um bloco $x = (x_0, x_1, \ldots, x_{N-1})$ essa medida poderia ser expressa por

$$ZCR(x) = \Big|\big\{n \mid \text{as amostras } x_n \text{ e } x_{n-1} \text{ têm sinais opostos}\big\}\Big|.$$

A motivação para esse estimador é o fato de que se um sinal é periódico, então ele cruza o eixo horizontal um número fixo de vezes por período, o que pode nos dar alguma pista de sua frequência fundamental, ao menos para formas de onda simples (por exemplo, uma senoide de frequência f_0 troca de sinal $2f_0$ vezes por segundo). Observe as três figuras a seguir, com dois sinais simples que possuem relação de 2:1 entre suas respectivas frequências fundamentais, e também entre suas taxas de cruzamento por zero.

Você verá que nos dois primeiros casos as formas de onda cruzam duas vezes o eixo a cada período ($f_0 = ZCR/2$), porém no terceiro caso a forma de onda cruza o eixo cinco vezes a cada período ($f_0 = ZCR/5$). Assim ZCR fornece uma medida útil que se correlaciona com a frequência fundamental, porém não deve ser confundida com essa última.

Para calcular a taxa de cruzamentos por zero em PureData iremos implementar a fórmula acima diretamente. Alguns objetos do Pd que podem te ajudar nessa parte são `[sgn~]` (devolve valores de sinal, +1, 0 ou -1), `[lrshift~]` (desloca um sinal um certo número de amostras para a esquerda ou para a direita), os objetos aritméticos (como `[*~]` e `[+~]` e os comparadores da biblioteca zexy (como `[zexy/==~]` ou `[zexy/<~]`). O esqueleto de patch que você deverá usar para sua implementação já traz uma solução direta usando `[cyclone/zerox~]`, que fornecerá um padrão de comparação.

5 Entrega

Você deverá entregar no Paca:

- 1 arquivo .pd com suas implementações dos valores estimados da amplitude de pico, amplitude rms e ZCR. Esas implementações devem ser feitas a partir do patch `esqueleto_ep1.pd`, que já possui uma infra-estrutura para selecionar e abrir um arquivo de áudio, e produzir os gráficos a partir de suas estimativas e também das soluções usando objetos prontos (o objetivo é zerar os gráficos das diferenças entre as implementações, mas talvez isso não seja possível em alguns casos).
- 1 arquivo .pdf com um pequeno relatório respondendo as seguintes questões:
    1. Dos arquivos de entrada que fornecemos, quais funcionaram bem com seus estimadores e com o tamanho de bloco default (N=64)? Quais não funcionaram tão bem?
    2. Explique caso a caso os motivos das estimativas serem melhores ou piores.
    3. Realize todas as experiências de novo com os tamanhos de bloco 128, 256, 512 e 1024. Alguma estimação que antes não estava funcionando passou a funcionar? Comente.
    4. Explique a relação entre a qualidade dos estimadores, os tamanhos de bloco e as frequências fundamentais. Qual é a menor frequência detectável dado o tamanho do bloco? Justifique.

Lembre-se de documentar bem o seu patch. Caso você implemente algo levemente errado, mas explique corretamente o que pretendia fazer, você ainda poderá receber alguma nota por isso.

6 Links importantes

https://en.wikipedia.org/wiki/Pitch_detection_algorithm
https://en.wikipedia.org/wiki/Zero_crossing
https://en.wikipedia.org/wiki/Autocorrelation
http://blog.joehahn.ws/list-puredata-objects-and-extended-objects
http://www.pd-tutorial.com/english/index.html

Boas implementações! =)