Processamento Digital de Sinais
Tutorial 09 - Convolução

Fabio Casagrande Hirono

Objetivos do Tutorial

Ao final desta sessão, você será capaz de:

- Calcular a convolução entre duas sequências,
- Utilizar a convolução para adicionar reverberação artificial a um sinal de áudio.

Sumário

1 Convolução

1.1 Revisão

Dado um sinal de tempo discreto $x[n]$ de comprimento L amostras, e um segundo sinal de tempo discreto $h[n]$ de comprimento M, a convolução $y[n]$ de $x[n]$ e $h[n]$ é definida como

$$y[n] = x[n] * h[n] = \sum_{m=0}^{M-1} h[m] \cdot x[n-m]. \tag{1}$$

1.2 Tarefas

1. Questão teórica: sejam `x` e `h` vetores de comprimento `L=100` e `M=4`, respectivamente. Qual será o comprimento do vetor `y` resultante da convolução de `x` e `h`? Forneça uma solução geral para o comprimento `Q` de `y`.

2. Escreva uma função Python `y = convtempo(x, h)` que implemente a equação (1) (não use a DFT!).

3. Escreva uma segunda função `y = convdft(x, h)` para realizar a convolução com DFT usando a função DFT que você criou no Tutorial 3. Lembre-se de que o número de pontos da DFT deve ser igual ao comprimento da saída. Se houver problemas com sua função DFT, use a funçã `numpy.fft.fft()`.

4. Teste as funções criadas nas Tarefas 2 e 3 usando `x = [3, 4, 5]` e `h = [1, 2, 3, 4]` como entradas. O pacote Numpy fornece a função `numpy.convolve()` para realizar uma convolução. Compare as saídas das três funções. Elas são idênticas?

2 Aplicações de Convolução

A convolução pode ser usada para implementar reverberação artificial de um sinal gravado em condições *anecóicas* usando uma resposta ao impulso medida em um espaço acústico.

2.1 Tarefas

1. Leia o arquivo de áudio `C4DM_GreatHall_Omni_x00y05.wav` [1], que contém uma resposta ao impulso medida em um salão, e escolha um sinal de entrada: o arquivo `voz.wav` ou o arquivo `cuica.wav`. A frequência de amostragem é de 48 kHz nos três casos. Plote a resposta ao impulso e pense sobre o que pode ter causado as principais características no gráfico entre 0s e 0,25 s.

2. Execute a convolução dos arquivos de áudio escolhidos usando sua função `convdft()` e `numpy.convolve()`. **Aviso**: Dependendo do desempenho do computador no qual o script é executado, este processo pode levar até vários minutos.

3. Auralize o sinal de saída e ouça o efeito resultante.

3 OPCIONAL: Convolução Overlap-Add

A convolução tipo sobreposição-e-soma (*Overlap-Add*) é uma implementação da convolução adequada para aplicações em tempo real porque divide o sinal de entrada em quadros relativamente pequenos e, em seguida, executa uma convolução para cada quadro para adicioná-los posteriormente.

3.1 Revisão

Para aplicar o método Overlap-Add, a sequência $x[n]$ definida na Seção 1.1 precisa ser subdividida em quadros com o comprimento escolhido N:

$$x_k[n] = \begin{cases} x[n + k \cdot N], & n = 0 \dots N-1 \\ 0, & \text{caso contrário,} \end{cases}$$

onde k denota o índice do quadro. O número total K de quadros é $K = \lceil \frac{L}{N} \rceil$, onde $\lceil \cdot \rceil$ denota o primeiro inteiro maior que o argumento.

Isso então leva a

$$x[n] = \sum_{k=0}^{K-1} x_k[n - k \cdot N].$$

[1] Obtido em http://isophonics.net/content/room-impulse-response-data-set

Esta subdivisão de $x[n]$ permite realizar a convolução para cada quadro individualmente para depois recombiná-los, em vez de executar uma única convolução longa. O resultado da convolução é então dado por

$$y[n] = \left(\sum_{k=0}^{K-1} x_k[n - k \cdot N] \right) * h[n] = \sum_{k=0}^{K-1} \sum_{m=0}^{M-1} h[m] \cdot x_k[n - k \cdot N - m] = \sum_{k=0}^{K-1} y_k[n - k \cdot N].$$

3.2 Tarefas

1. Gere um vetor `x` de comprimento `L = 100`, contendo apenas o valor 1 em todos os elementos. Em seguida, gere um vetor `h=[1, 1, 1, 1]`, representando a resposta ao impulso de um sistema.

2. Divida `x` em quadros de comprimento `N=10` e salve-os em uma matriz `X`.

3. Faça a convolução de cada vetor linha de `X` com a resposta ao impulso `h`, usando sua função `convdft`, e salve os resultados em uma segunda matriz `Y`. Certifique-se de que `Y` tem as dimensões apropriadas.

4. Finalmente, componha o sinal `y` a partir da matriz `Y`, usando o método Overlap-Add. Você pode verificar seus resultados de sua convolução Overlap-Add comparando-a com uma convolução de bloco único usando `numpy.convolve()`.

5. Refaça as tarefas 2-4 usando `N = 52`. Observe as implicações disso, já que o último quadro do sinal será mais curto que `N`.

6. Escreva uma função `y = OverlapAdd(x, h, N_DFT)` executando a convolução de `x` e `h` com o método Overlap-Add. Certifique-se de que ela pode lidar com sinais que não podem ser divididos exatamente em quadros de comprimento `N`. Note que `N` é definido de forma única como $N = N_{\mathrm{DFT}} - M + 1$.