Processamento de Sinal (DF-FCUL)

Outubro de 2014

Convolução linear, convolução circular, FFT, *"overlap-add"* (V2.0)

Este trabalho aborda o processamento de sinais efectuado por Sistemas Lineares e Invariantes no Tempo (SLITs) discretos, quer no domínio do tempo, quer no domínio da frequência.

No domínio do tempo a operação por eles realizada é a **Convolução Linear**. O adjectivo "linear" é aqui utilizado para a diferenciar da **Convolução Circular** que, como iremos ver, é uma operação diferente mas essencial à implementação da convolução linear no domínio da frequência, via DFT/FFT.

De facto, no domínio da frequência a convolução linear pode ser implementada eficientemente através da "Fast Fourier Transform" (FFT que, recorde, é um algoritmo que calcula eficientemente a Discrete Fourier Transform, a DFT) como consequência da associação da DFT à convolução circular.

Há um aspecto importante a ter em conta: **no Scilab, os índices dos vectores começam em $n = 1$ e não em $n = 0$.** Assim, todos os sinais e os limites dos somatórios nas convoluções deverão ter em consideração este desvio de uma unidade de tempo relativamente à convenção usada nas aulas (e na generalidade dos textos de PDS), onde se iniciam em $n = 0$.

Convolução Linear

No Scilab convolve-se $x[n]$ e $h[n]$ com `convol(x,h)`.

Este operador realiza a **convolução linear** de $x[n]$ com $h[n]$ (onde N é a dimensão de $h[n]$ e M é a dimensão de $x[n]$). Recorde que essa convolução linear serve para calcular a saída do sistema, $y[n]$:

$$y[n] = \sum_{k=0}^{N-1} h[k]\, x[n-k] \qquad n = 0, \cdots, N+M-2 \quad (1)$$

$$y[n] = \sum_{k=0}^{M-1} x[k]\, h[n-k] \qquad n = 0, \cdots, N+M-2$$

A convolução de duas sequências de dimensões finitas, respectivamente M e N, tem a dimensão $M+N-1$.

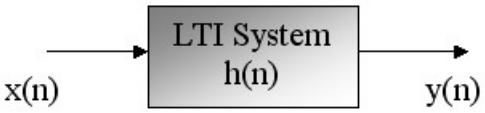


Fig. 1: Sistema LTI como unidade de processamento.

Exemplifique a convolução com sinais de reduzida dimensão e verifique manualmente os resultados.
Utilize duas vias: (i) a operação convol() do Scilab; (ii) concretize a convolução pela implementação explícita do somatório (1), usando 'ciclos for' e tendo atenção aos limites dos índices que irá usar.
Concretamente, use os sinais "curtos", de dimensão $M = 8$, respectivamente definidos por $f[n] = U[n] - U[n-3] - \delta[n-7]$ e $g[n] = -n\,U[n]$, faça a sua convolução $f * g$ pelos dois processos referidos e observe o gráfico. Verifique que a dimensão de $f * g$ é $2M - 1$.

A Fast Fourier Transform (FFT)

Os operadores no Scilab relevantes são `fft(x)` e `ifft(X)`. O operador `fft()` calcula eficientemente a DFT de um sinal, utilizando um algoritmo da famíla das "Fast Fourier Transforms" . O operador `ifft()` calcula a DFT inversa (IDFT).

Em concreto, calcula-se a DFT com:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{nk} \qquad k = 0, 1, \cdots, N-1 \qquad (2)$$

onde $W_N = e^{-j2\pi/N}$ é o inverso (ou o conjugado) da N-ésima raiz primitiva da unidade. Diz-se que $X[k]$ é a DFT de $x[n]$.

A síntese do sinal $x[n]$ efectuada a partir de $X[k]$ é denominada IDFT (i.e., DFT Inversa) e consiste em

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k]\, W_N^{-nk} \qquad n = 0, 1, \cdots, N-1 \quad (3)$$

É visível a simetria entre as expressões de $X[k]$ e $x[n]$, o que explica o facto de a FFT, o algoritmo eficiente de cálculo da DFT, ser usada também para calcular a IDFT. $X[k]$ e $x[n]$ podem ser sequências complexas, embora na maioria das aplicações $x[n]$ seja real.

Das expressões (2) e (3), retira-se que o cálculo de cada ponto de $X[k]$ ou de $x[n]$ exige aproximadamente

N somas e N multiplicações (complexas), ou seja, é da ordem de $2N$ *flops* ("floating-point operations"). Como há N pontos para calcular em cada caso, conclui-se que, aparentemente, no cálculo das sequências $\{X[k]\}$, na DFT, ou $\{x[n]\}$, na DFTI, estão envolvidas $\sim N^2$ flops (i.e., a complexidade é da ordem de N^2).

Felizmente, há um "atalho" que permite diminuir imenso aquele valor. A FFT é um algoritmo que explora a periodicidade de W_N^{nk} para calcular a sequência $\{X[k]\}$ usando apenas $\sim N \log_2 N$ flops, o que representa uma economia sensacional face aos anteriores N^2. Por exemplo, se $N = 1024$, então $N^2 \approx 10^6$ e $N \log_2 N \approx 10^4$, o que corresponde a uma economia de 99% na computação da DFT. O algoritmo subjacente à FFT será estudado nas aulas.

A convolução circular

Considere que $X[k]$ e $H[k]$ são DFTs dos sinais $\hat{x}[n]$ e $\hat{h}[n]$, respectivamente[1]. Todos eles têm dimensão N. O vector $Y[k]$ dado pela multiplicação ponto a ponto de $X[k]$ e $H[k]$

$$Y[k] = X[k] \times H[k] \tag{4}$$

é a DFT do sinal $\hat{y}[n]$, de dimensão N, correspondente à convolução circular de $\hat{x}[n]$ e $\hat{h}[n]$

$$\hat{y}[n] = \hat{x}[n] \circledast \hat{h}[n]$$

A convolução circular é igual à convolução linear de $\hat{x}[n]$ com a **extensão periódica**, de período N, de $\hat{h}[n]$, que denominamos de $\hat{h}_N[n]$

$$\hat{y}[n] = \sum_{k=0}^{N-1} \hat{x}[k]\, \hat{h}_N[n-k] \qquad n = 0, \cdots, N-1 \tag{5}$$

Considera-se que $\hat{y}[n]$ **existe no intervalo** $0 \leq n < N$ **e tem dimensão** N. Compare-a com a dimensão da convolução linear de $\hat{x}[n]$ e $\hat{h}[n]$ que seria $2N-1$.

Convolução linear pela FFT

Considere agora que quer realizar a convolução linear de dois sinais $x[n]$ e $h[n]$ de dimensões N_X e N_H respectivamente. Essa convolução, $y[n] = x[n] * h[n]$, terá, como vimos, a dimensão $N = N_Y = N_X + N_H - 1$. Eis como poderá realizá-la calculando apenas FFTs.

Primeiro, recorde que existe uma relação entre o produto de duas FFTs/DFTs (4) e a convolução circular (5). De seguida deve notar-se que se se **estender com zeros** ("padding") os sinais s $x[n]$ e $h[n]$ de dimensões N_X e N_H, até à dimensão $N = N_X + N_H - 1$, chamando a estas extensões $\hat{x}[n]$ e $\hat{h}[n]$, então ter-se-á

$$\hat{y}[n] = \hat{x}[n] \circledast \hat{h}[n] = y[n] = x[n] * h[n]$$

e atendendo ao facto de a convolução circular poder ser realizada indirectamente pela FFT, pois

$$\hat{y}[n] = \mathcal{IFFT}\{Y[k]\}$$

temos a seguinte "receita" para realizar a convolução linear $y[n] = x[n] * h[n]$ através da FFT:

1. Efectuar a extensão com zeros dos sinais $x[n]$ e $h[n]$ de dimensões N_X e N_H, até à dimensão $N = N_X + N_H - 1$, chamando-lhe $\hat{x}[n]$ e $\hat{h}[n]$;
2. Calcular as respectivas FFTs, $X[k] = \mathcal{FFT}\{\hat{x}[n]\}$ e $H[k] = \mathcal{FFT}\{\hat{h}[n]\}$, de dimensões N;
3. Calcular o seu produto ponto-a-ponto $Y[k] = X[k] \times H[k]$ conforme (4);
4. Finalmente, obter o sinal desejado, $y[n] = \hat{y}[n] = \mathcal{IFFT}\{Y[k]\}$.

Devido à comutatividade da convolução circular, os papéis de $x[n]$ e $h[n]$ em (5) podiam ser trocados, ou seja, poder-se-ia convolver $h[n]$ com $\hat{x}_N[n]$, a extensão periódica de $x[n]$.

Aplique estes operadores a alguns sinais de dimensão reduzida. Calcule o produto das transformadas de dois sinais (e.g. para os sinais x e h calcule $Y = X\,H$) e verifique que **a IDFT de Y corresponde à convolução circular** dos referidos sinais, i.e. $y = x \circledast h$. Note que ambos os sinais e a sua convolução circular têm a mesma dimensão, N (se não tiverem inicialmente, terão de ser estendidos com zeros).
A convolução linear de dois sinais de duração N_1 e N_2 é um sinal de duração $N = N_1 + N_2 - 1$. Para poder realizar convoluções lineares (as operações associadas ao processamento de sinais com SLITs discretos) indirectamente através da DFT/FFT, *há que estender os sinais originais até à dimensão* N, *usando-se um número suficiente de zeros* ("padding"), por forma que a convolução circular destes sinais "estendidos" seja igual à sua convolução linear. Faça a extensão de vectores com as funções `ones()` e `zeros()`.
Exemplifique esta operação com alguns sinais (mormente aqueles usados no exemplo anterior) e verifique a equivalência entre os resultados obtidos directamente com a convolução linear e indirectamente usando o produto de DFTs no domínio da frequência.

[1] A razão do uso do '^' nos sinais será clara mais adiante.

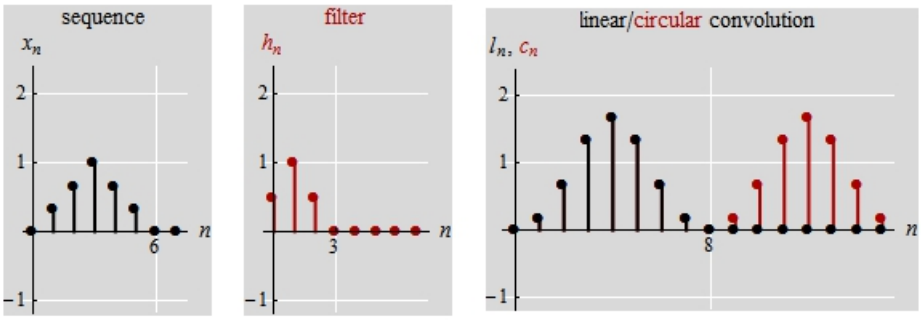


Fig. 2: Igualdade das convoluções circular e linear quando os sinais são previamente 'enchidos' com zeros até à dimensão final $N_H + N_X - 1$. Neste exemplo $N_X = 6$ e $N_H = 3$. Logo, $N = N_X + N_H - 1 = 8$. Logo, note que as convoluções circular e linear são iguais para $0 \leq n < 8$ (como se disse, apenas nos interessa aqui o resultado da convolução circular para $0 \leq n < N$).

Sinais longos e a técnica *overlap-add*

Em muitas situações práticas de processamento digital de sinais, $h[n]$ é o *kernel* de um filtro FIR e tem uma dimensão limitada (corresponde a algumas dezenas ou poucas centenas de pontos). Por seu lado, $x[n]$ é um sinal amostrado continuamente em tempo real (um sinal audio ou vídeo, um sinal de radar, um sinal de origem biológica, etc...), tem uma dimensão virtualmente "infinita" e, aparentemente, quando se substitui a convolução (linear) pelo produto das FFTs só se poderia obter o sinal de saída do filtro, $y[n]$, após $x[n]$ já ser conhecido na sua totalidade.

Para ser possível efectuar processamento em tempo real no domínio da frequência, aquela limitação tem de ser ultrapassada. Uma das técnicas que permite fazê-lo consiste em "fatiar" o sinal de entrada em blocos de dimensão L, denominados aqui $x_p[n]$ e, assentando no princípio da linearidade, fazer à vez o produto das transformadas $X_p[k]$ destes blocos com $H[k]$, calcular a IFFT desse produto para obter a saída parcelar $y_p[n]$ e adicionar correctamente (i.e., com a adequada translacção no tempo) os resultados parcelares, resultantes do processamento de cada bloco, para obter a solução total. Esta técnica denomina-se "overlap-add" e permite debitar para o exterior, periodicamente, blocos de $y[n]$ com dimensão L.

Exemplifique o "overlap-add" para $h[n]$ com dimensão $N_H = 32$ ou 64, e um sinal $x[n]$ de dimensão $N_X = 1024$. Faça um programa para efectuar esta operação usando as capacidades do *Scilab* no que respeita a instruções de programação convencional (ciclos 'for' e iteração, 'if-then-else', etc...) e selecção de sub-vectores (e.g. `vec(ninit:nfinal)`). Use fatias do sinal de entrada de dimensão $L = 64$, por exemplo.
O filtro $h[n]$ poderá ser o filtro passa-baixo correspondente à amostragem $h[n] = T\,h(nT)$, com $h(t) = e^{-t/\tau}U(t)$ a corresponder à resposta impulsiva de um filtro passa baixo de primeira ordem com frequência de corte $\omega_C = 1/\tau$. Ou, então, poderá usar um outro sinal $h[n]$ à sua vontade.

Note que o processamento em tempo real pela realização explícita da convolução não necessitaria do fatiamento do sinal de entrada em blocos. Porém, verifica-se que na prática a abordagem da filtragem baseada em FFTs é mais rápida quando a dimensão N_H da resposta impulsiva $h[n]$ ultrapassa aproximadamente trinta pontos...

Sobre a utilização do Scilab

Os sinais discretos no tempo (sequências) são representados no *Scilab* por vectores, que podem ser considerados matrizes de dimensão $n \times 1$ (ou $1 \times n$). As funções `ones(m,n)` e `zeros(m,n)` criam matrizes com m linhas e n colunas preenchidas com uns ou zeros, respectivamente. A função `rand()` gera um número aleatório colhido numa distribuição uniforme entre 0 e 1. Por seu lado, `rand(m,n)` gera uma matriz de

dimensão $m \times n$ preenchida com números aleatórios seguindo aquela distribuição estatística. Para mudar a distribuição de números aleatórios para uma Gaussiana de média nula e variância unitária , $N(0,1)$, executa-se a instrução `rand('normal')`. Para voltar a utilizar a distribuição uniforme entre 0 e 1, $U(0,1)$, executa-se `rand('uniform')`.

Para aceder a um elemento de um vector indica-se o seu índice com parêntesis curvos. Por exemplo, `vec(5)` refere-se ao $5^{\underline{o}}$ elemento do vector `vec`. Os índices dos vectores e matrizes começam sempre em 1 no Scilab. Estes objectos podem ser criados e preenchidos explicitamente, escrevendo os elementos linha a linha e separando cada linha por ';': por exemplo, `I2=[1 0 ; 0 1]` cria uma matriz identidade com dimensão 2 (a mesma matriz pode ser criada com `eye(2,2)`). A dimensão de um objecto `obj` (que pode ser um vector, uma matriz, etc...) é dada pelo operador `length(obj)`. No caso de ser uma matriz ou um vector, `size(obj)` devolve simultaneamente as duas dimensões. A plica (') permite transpor um vector ou uma matriz, e.g. `vec'` é o transposto do vestor `vec` (um deles é vector coluna e o outro é vector linha).

Para visualizar os sinais usa-se a função `plot2d()` ou uma das suas variantes: `plot2d2()`, `plot2d3()`, etc... Pode também ser usada a função `plot()` que tenta mimetizar a sua homónima do *Matlab*. A função `clf()` limpa o painel gráfico anterior e deve ser escrita num script antes dos 'plots' serem executados para evitar a sobreposição de gráficos. É útil quando se usa o Scilab interactivamente, para visualizar gráficos consecutivamente. A função `sleep(m)` congela o processamento por m milissegundos (é útil para mostrar vários gráficos ou tocar vários sons com `sound()` em sequência).

No *Scilab*, a constante $j = \sqrt{-1}$ é representada por `%i`. A base do logaritmo neperiano é `%e`= 2.7182818.... Finalmente, π é representado por `%pi`.

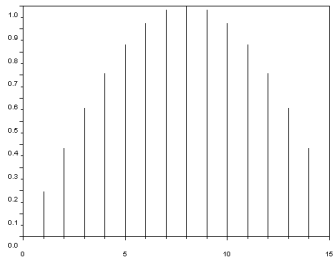
Sinais

Para testar as funções de processamento de sinal podem ser usados sinais de várias dimensões. Podemos criar degraus e impulsos com `ones()` e `zeros()`, criar troços de sinusóides ou exponenciais, etc...

Por exemplo, o trecho de código seguinte cria e mostra um sinal básico:

```
N=16
k=0:N-1
clf
plot2d3(k,sin(%pi*k/N))
```

produz o gráfico seguinte:



O limite na criação de sinais depende, obviamente, da sua criatividade :-).