Sinais e Sistemas Digitais

Osmar Tormena Junior, Prof. Me.*

1 Formato digital

Sistemas discretos são sistemas teóricos que operam sobre sinais discretos, que por sua vez, são funções matemáticas $f : \mathbb{Z} \mapsto \mathbb{R}$ idealizadas. Embora sinais e sistemas discretos sejam uma ferramenta útil no desenvolvimento teórico do processamento de sinais, a implementação prática de processamento digital de sinais requer atenção em alguns pontos.

Sistemas digitais possuem tamanho de palavra finito, em seu número de bits. Isto faz com que o mapeamento sobre os reais de sinais discretos fique restrito à um subconjunto finito dos números racionais ($\mathbb{Q}$).

Um tratamento formal dos erros inseridos pela quantização, e seu impacto nas técnicas discretas, implementadas digitalmente, requerem uma abordagem de variáveis aleatórias e processo estocástico que fogem do escopo a nível de graduação.

Dessa maneira, muito embora as particularidades de sinais e sistemas digitais não sejam abordadas especificamente, serão encontradas situações onde erros numéricos, devido à precisão finita, serão relevantes, exigindo um discernimento por parte de quem implementa o sistema.

De uma forma geral, sistemas digitais trabalham com um número de bits como: 8, 16, 32, 64. Essas representações podem ser de inteiros sem sinal (`uint8`, `uint16`, `uint32` e `uint64`), inteiros com sinal (complemento de 2, como: `int8`, `int16`, `int32` e `int64`), ponto fixo (*single precision* de 32 bits ou *double precision* de 64 bits) e ponto flutuante (IEEE 754—também *single precision* de 32 bits ou *double precision* de 64 bits).

Em simulações computacionais, é comum o uso do ponto flutuante *double precision*, onde seus 64 bits estão dispostos como: 1 bit de sinal, 11 bits de exponente (base 2) e 52 bits para representar a parte fracionária do binário "$1, \ldots$". A função `eps` é útil para averiguar a incerteza intrínseca à representação digital de um número.

Exemplo 1: O polinômio de Wilkinson, definido por

$$p(x) = \prod_{i=1}^{20} (x - i) = (x-1)(x-2)\cdots(x-20),$$

é um exemplo clássico das limitações da aritmética de ponto flutuante. Nele, um polinômio, com as raízes de 1 a 20, é construído (i.e. seus coeficientes são determinados), para logo em seguida, suas raízes serem obtidas, a partir dos coeficientes do polinômio. Verifique, utilizando os comandos `poly` e `roots`, os erros em sua plataforma de simulação.

```
%% Polinômio de Wilkinson

roots(poly(1:20))
```

□

2 Sinais digitais

Em um computador, devido à valores finitos de capacidade de memória e poder de processamento, todos os sinais digitais são de *suporte finito* (i.e. existem M e $N \in \mathbb{Z}$, com $M < N$, tal que $x(n) = 0$ para $n < M$ ou

*tormena@utfpr.edu.br

$n > N$). Desta maneira, um sinal pode ser visto como um vetor multidimensional. Isto permite que as poderosas ferramentas da álgebra de espaços vetoriais sejam aplicadas à operações com sinais.

Exemplo 2: Crie e plote o sinal senoidal $x(n) = \text{sen}(\omega_0 n)$, com $\omega_0 = 0.1\pi$, para $n \in [-10, 10]$.

```
%% Visualização de sinais digitais

w0 = 0.1*pi;
n = -10:10;
x = sin(w0*n);

stem(n,x), grid on
xlabel('n (Amostras)')
ylabel('x(n)')
```

□

Exercício 1: Modifique o código do Exemplo 2, para visualizar a sequência $x(n) = 0{,}5^n$, para $n \in [0, 8]$.

Muito embora a plataforma GNU Octave ofereça suporte direto a um grande número de funções matemáticas, estranhamente, ela não define dois sinais de grande importância: o impulso unitário $\delta(n)$ e o degrau unitário $u(n)$.

Exemplo 3: O script `impulso.m` define a função $\delta(n - m)$, no intervalo $n \in [M, N]$.

```
function [n,x] = impulso(M,N,m)
% Produz impulso deslocado de m, no
% intervalo [M,N].
% [n,x] = impulso(M,N,m)
% Argumentos de entrada
%   M: limite inferior do sinal
%   N: limite superior do sinal
%   m: posicao do impulso
% Argumentos de saida
%   n: indices do sinal
%   x: sinal impulso deslocado

if ~isnumeric([M,N,m])
    error('Argumentos de entrada
    devem ser numeros.')
end
if M~=round(M) || N~=round(N) ||
m~=round(m)
    error('Argumentos de entrada
    devem ser inteiros.')
end
if M>=N
    error('N deve ser maior que M.')
end

n = M:N;
x = double((n-m)==0);
```

□

Exercício 2: Modifique o código do Exemplo 3, produzindo uma script `degrau.m`, implementando a função $u(n - m)$, no intervalo $n \in [M, N]$.

É interessante que os discentes mantenha uma biblioteca de funções, para diminuir o retrabalho, no decorrer do curso. A validação das entradas, uma das boas práticas de programação, evita que *bugs* escondidos em sub-funções compliquem a depuração de problemas.

3 Sistemas digitais

Um sistema digital pode ser um *operador*, que produz um sinal de saída $y(n)$ aplicando uma operação (ou transformação) sobre um sinal de entrada $x(n)$. Também é possível que o sistema seja um *funcional*, onde um sinal de entrada $x(n)$ produz como saída um único valor escalar.

Conforme estabelecido, é possível representar um sinal digital de suporte finito como um vetor multidimensional. Como o GNU Octave opera primariamente sobre matrizes, há uma simplicidade da notação. Porém alguns cuidados devem ser tomados, como ilustra o Exemplo 4.

Exemplo 4: A multiplicação de dois vetores coluna, de mesma dimensão, pode ser definida de diferentes maneiras.

```
x1 = randi(5,4,1);
x2 = randi(5,4,1);
```

```
x1*x2 % Resulta em erro
x1*x2.'% Resulta numa matriz 4x4
x1.'*x2 % Resulta num escalar
x1.*x2 % Resulta num vetor coluna
```

Observe que cada resultado possui uma significância particular, em especial, os dois últimos, que equivalem ao produto interno e à multiplicação elemento-a-elemento, respectivamente. □

Exercício 3: Implemente os seguintes sistemas, plotando os sinais de cada problema em uma única figura (Dica: comandos `figure` e `hold`). Considere o sinal de entrada senoidal.

$$y(n) = 3x(n) + 1 \tag{1}$$

$$y(n) = x(n)^2 \tag{2}$$

$$y(n) = \sum_{m=-\infty}^{n} x(m) \tag{3}$$

$$y(n) = x(n) - x(n-1) \tag{4}$$