Exper.

1

**Sinais – Representação e Manipulação**

# Objetivo

Este laboratório descreve como representar e manipular alguns sinais elementares computacionalmente:

* Criação e armazenamento de sinais
* Importação de sinais
* Visualização de sinais graficamente

# Fundamentação Teórica

SINAL: É uma função matemática, de uma ou mais variáveis independentes, que representa informações.

Nesta disciplina trabalharemos apenas com sinais unidimensionais (funções de uma única variável, geralmente o tempo, *t*) nos domínios dos números REAIS e COMPLEXOS.

# Procedimento Prático

1. **Criação e Armazenamento de Sinais em Python**

O armazenamento de sinais unidimensionais faz uso de arranjos numéricos ou vetores (estruturas de dados homogêneos, no caso, valores numéricos reais relacionados às amplitudes das amostras), e na ling. Python, será preciso carregar o pacote **Numpy** para manipular vetores:

import numpy as np

Base de tempo: t = np.array([0.0, 0.1, 0.2, 0.3])

Amplitude do sinal: x = np.array([1.0, 3.2, 2.0, 8.5])

Os objetos de dados básicos do processamento de sinais (sinais unidimensionais, ou seqüências, sinais multicanais, e sinais bidimensionais) são todos naturalmente acomodados/representados em arranjos numéricos (vetores e matrizes).

Vetores são arranjos de dimensões 1xN (vetor linha) ou Nx1 (vetor coluna), onde *N* é o número de amostras no sinal. Para se armazenar uma sequência de valores numéricos em Python deve-se entrar uma lista de elementos na linha de comando e convertê-la para arranjo. O comando:

x = np.array([[1., 2., 3., 4., 5.]])

cria uma sequência de valores reais contendo cinco elementos em um vetor linha: 1x5. O vetor linha x pode ser convertido em vetor coluna fazendo-se a transposição vetorial:

y = x.transpose()

Vetores podem ser estendidos naturalmente para o caso de sinais multicanais (multidimensionais), no qual cada canal é representado por uma coluna (ou linha) de um arranjo matricial.

Uma forma eficiente de se criar uma sequência de dados é usar a função arange(). Por exemplo, para armazenar um segundo de um sinal amostrado a 1 kHz (1000 amostras), usamos

Ts = 1.0e-3 # ou Ts = 0.001

t = np.arange(0.,1.,Ts) # arange(lim\_inf, lim\_sup, passo)

O vetor *t* é criado com 1000 elementos indexados de 0 a 999 e que representam o intervalo de tempo de zero a um segundo, em passos de milésimos de segundo. Você também pode usar a função *linspace* para criar o mesmo vetor de dados:

t = np.linspace(0,0.999,1000)

Digite os seguintes comandos:

t1 = np.array([0.,.1,.2,.3])

t2 = np.arange(0.,0.4,0.1)

t3 = np.linspace(0.,0.3,4)

T = np.array([t1,t2,t3])

X = np.sin(T)

print X.transpose()

O que faz o código mostrado acima?

1. **Sinais Amostrados**

Fontes de sinais analógicos incluem geradores eletromagnéticos, de áudio, sonar, biomédico e outras. Sinais analógicos precisam ser amostrados e quantizados antes de serem processados digitalmente.

**Amostragem**: x[n] = x*a*(nT*s*)

onde:   
*x* é um sinal discreto amostrado a partir de um sinal analógico *xa* com frequência de amostragem *fS* = 1/*TS*.

Experimente o seguinte trecho de programa Python (*script*):

import matplotlib.pylab as mpl

import numpy as np

Fs = 100.; N = 1000; tempofinal = 9.99

t1 = np.arange(0,N)/Fs

t2 = np.arange(0.,tempofinal,1./Fs)

x1 = np.sin(2\*np.pi\*0.5\*t1)

x2 = np.sin(2\*np.pi\*3\*t2)

mpl.plot(t1,x1,lw=2), mpl.figure(), mpl.plot(t2,x2,lw=2)

Quantos ciclos são mostrados em cada um dos sinais plotados pelo código acima?

Uma alternativa à criação de sinais é usar as funções do módulo **signal** (pacote [**scipy**](http://docs.scipy.org/doc/scipy/reference/signal.html)): chirp, gausspulse, sawtooth, square, ... Uma variedade de funções geram formas de onda. Cada uma delas requer que você inicie com um vetor representando a base temporal (var. independente, abscissas). Algumas dessas funções serão descritas mais adiante.

**Aliasing**:

Sinais digitais são normalmente derivados da amostragem de sinais de tempo contínuo via *Conversor Analógico-Digital* (A/D). Se o sinal contínuo, *xa(t)*, for limitado em banda (não contém frequências acima de uma frequência máxima *fM*), então o teorema da amostragem (Shannon-Nyquist) diz que esse sinal pode ser completamente recuperado a partir de um conjunto de amostras, se a frequência de amostragem *fs* for maior do que duas vezes a máxima frequência do sinal contínuo a ser amostrado:

*fs > 2.fM*

Esta freqüência máxima *fM* é conhecida como a frequência de *Nyquist*. Se a frequência de amostragem não for maior que duas vezes a frequência máxima do sinal, então o sinal contínuo não poderá ser unicamente recuperado a partir das amostras, por acontecer o fenômeno ‘*aliasing*’ (sobreposição de réplicas do espectro de frequências do sinal contínuo).

Experimente:

t = np.arange(0.,2.,0.001)

xa = np.sin(2\*np.pi\*5\*t) # sinal analógico

mpl.plot(t,xa); mpl.hold('on')

fs = 15

ts =np.arange(0.,2.,1./fs); xs1=np.sin(2\*np.pi\*5\*ts) # sinal amostrado 15Hz

mpl.plot(ts,xs1,'bo-')

fs = 7.5

ts = np.arange(0.,2.,1./fs); xs2= np.sin(2\*np.pi\*5\*ts) # sinal amostr. 7,5Hz

mpl.plot(ts,xs2,'ro-')

Existe *aliasing* em algum dos sinais amostrados? Qual? Por quê?

1. **Visualização de Sinais**

* Visualizar a amplitude do sinal em função do tempo
* Funções: *plot*, *stem*
* Ouvir o sinal de dados: *sound*

Obs.: os comandos *sound* e *soundsc* não funcionarão se o hardware do seu computador não estiver instalado corretamente. Se este for o caso, veja os sinais ao invés de ouvi-los.

Experimente o seguinte script (Matlab/Python):

t = [0.1 0.2 0.3 0.4];

x = [1.0 8.0 4.5 9.7];

plot(t,x)

figure, stem(t,x)

figure, stairs(t,x)

fs = 1000;

ts = 0:1/fs:2;

f = 250 + 240\*sin(2\*pi\*ts);

x = sin(2\*pi\*f.\*ts); % .\* => multiplic. elemento a elemento

strips(x,0.25,fs)

sound(x,fs)

plot(ts,x)

plot(ts(1:200),x(1:200))

from numpy import array, arange, pi, sin, float32

from pylab import plot, figure, stem, step, xlim; import pyaudio

def som(x,fs):

p = pyaudio.PyAudio()

stream = p.open(format=pyaudio.paFloat32, channels=1, rate=fs, output=True)

stream.write(x.astype(float32))

stream.stop\_stream()

stream.close()

p.terminate()

t = array([0.1, 0.2, 0.3, 0.4]); x = array([1.0, 8.0, 4.5, 9.7]); plot(t,x)

figure(); stem(t,x)

figure(); step(t,x,where='post'); xlim(0.1,0.5) # Em Matlab: stairs(t,x)

fs = 10000; ts = arange(0,2,1./fs)

f = 250 + 240\*sin(2\*pi\*ts); y = sin(2\*pi\*f\*ts)

som(y,fs)

figure(); plot(ts,y)

figure(); plot(ts[:199],y[:199])

O que faz o comando *strips*? (dica: digite na linha de comandos do Matlab: ’>> *help strips*’)

Para que serve o operador **.\*** usado no *script* Matlab?

**Ferramenta de Processamento de Sinais (sptool):**

A aplicação da caixa de ferramentas de processamento de sinais, denominada SPTool (Signal Processing Tool), disponibiliza um ambiente gráfico para visualização de sinais, projeto de filtros, e análise espectral. Essa ferramenta utiliza a interface gráfica com o usuário (GUI) para facilitar a execução das tarefas citadas. O SPTool é constituído por:

* *Paginador de sinais* (Signal Browser) – usado para se analisar sinais. Você pode também partes de sinais usando o hardware de áudio do seu computador.
* *Projetista de filtros* (Filter Designer) – usado para se projetar ou editar filtros digitais do tipo FIR ou IIR. A ferramenta FDATool é outra aplicação GUI mais específica para se projetar filtros.
* *Visualizador de filtros* (Filter Viewer) – usado para se analisar as características dos filtros.
* *Visualizador de espectro* (Spectrum Viewer) – usado para se fazer análise espectral dos sinais.

Para carregar e abrir a aplicação SPTool, comande:

sptool

Veja o sinal *train*, o filtro *FIRbp*, e o espectro *trainse*. (você verá 3 figuras – Sinais, Filtros, Espectros. O Filter Designer está disponível via opção de menu *File → Preferences*. Você pode tocar os sons usando o ícone do alto-falante disponível na barra de ferramentas do *Signal Browser*.)

*Importando um Sinal no SPTool*

Você pode usar o SPTool para analisar sinais, filtros ou espectros que você criar na linha de comandos do Matlab. Você pode importar sinais, filtros ou espectros do espaço de trabalho (workspace) do Matlab para dentro do SPTool usando a opção *Import* do menu *File*.

Experimente:

fs = 1000;

ts = 0:1/fs:0.5;

f = 250 + 240\*sin(2\*pi\*ts);

x = sin(2\*pi\*f.\*ts);

Importe os sinais *f* e *x* para dentro do SPTool e use a ferramenta para examiná-los.

Quais são os ícones disponíveis para ampliação de visão horizontal (horizontal zoom)?

*Paginador de Sinais* (Signal Browser)

Essa ferramenta é um ambiente interativo de exploração do sinal. Ela fornece uma visualização gráfica dos objetos sinais correntemente selecionados na lista de sinais do SPTool. Usando o Signal Browser você pode:

* Visualizar e comparar vetores e arranjos de sinais
* Visualizar mais de perto uma faixa de dados do sinal (zoom in) para examiná-lo com mais precisão.
* Medir uma variedade de características do sinal
* Tocar um sinal no hardware de áudio

Para abrir/ativar o Signal Browser do SPTool, proceda assim:

* Clique em um ou mais sinais (use a tecla Shift para seleções múltiplas) na lista de sinais do SPTool.
* Clique o botão [View] na lista de sinais do SPTool.

1. **Geração de Sinais**

**Sinais**

Crie um vetor com a base de tempo para o sinal: t = [0:0.1:2];

Crie um sinal como função do tempo: x = sin(pi\*t/2); plot(t,x)

Funções úteis do Matlab para gerar sinais:

• funções não-periódicas: *ones, zeros, step, impulse, gensig*

• funções periódicas: *sin, cos, square, sawtooth*

**Sinais não-periódicos**

t = linspace(0,1,11)

• Degrau Unitário (step):

y = ones(1,11);

stem(y)

• Impulso Unitário:

y = [1 zeros(1,10)];

stem(y)

• Rampa:

y = 2\*t;

plot(y)

Experimente: *Degrau unitário*

fs = 10;

ts = [0:1/fs:5 5:1/fs:10];

x = [zeros(1,51) ones(1,51)];

stairs(ts,x)

*Impulso com largura* *w*:

fs = 10;

w = 0.1;

ts = [-1:1/fs:-w 0 w:1/fs:1];

x = [zeros(1,10) 1 zeros(1,10)];

plot(ts,x)

*impulso unitário*:

ts = 0:0.5:5;

x = [1 zeros(1,length(ts)-1)];

stem(ts,x)

axis([-1 6 0 2])

• Senóides: parâmetros de senóides:

• Amplitude, A

• Frequência, f

• Fase, φ

• Deslocamento Vertical (polarização, oﬀset), B

A forma geral de uma senóide é: y = A.sen(2πft + φ) + B

*Exemplo*: gerar uma senóide com as seguintes especificações: A = 5; f = 2 Hz; φ = π/8 rad.

t = linspace(0,1,1001);

Ampl = 5;

freq = 2;

fase = pi/8;

senoide = Ampl \* sin(2\*pi\*freq\*t + fase);

plot(t, senoide)

• Sinais Quadrados: formas de onda quadradas devem ter especificado o ciclo de trabalho (o percentual de tempo em que a amplitude do sinal fica em nível alto – duty cycle).

*Exemplo*: Duty cycle de 50% (padrão do Matlab) e freqüência de 4 Hz.

t = linspace(0,1,1001);

squad1 = square(2\*pi\*4\*t);

plot(t,squad1)

axis([-0.1 1.1 -1.1 1.1])

*Exemplo*: Duty cycle de 75% e freqüência de 4 Hz.

t = linspace(0,1,1001);

squad2 = square(2\*pi\*4\*t,75);

plot(t,squad2)

axis([-0.1 1.1 -1.1 1.1])

• Dente-de-serra: assim como as ondas quadradas, esses sinais precisam de um parâmetro que indica a localização do pico da onda.

*Exemplo*: Pico no final do período (padrão do Matlab) e freqüência de 3 Hz.

t = linspace(0,1,1001);

ds1 = sawtooth(2\*pi\*3\*t);

plot(t,ds1)

*Exemplo*: Pico na metade do período e freqüência de 3 Hz.

t = linspace(0,1,1001);

ds2 = sawtooth(2\*pi\*3\*t,1/2);

plot(t,ds2)

• Sinais Complexos: sinais periódicos podem ser representados por exponenciais complexas:

*x*(*t*)  *= ej2πft = cos*(*2πft*) *+ j sen*(*2πft*) *= cos*(*Ωt*) *+ j sen*(*Ωt*)

se *t* for medido em segundos, então *f* terá unidades de seg-1, e Ω terá unidades de radianos/seg.

Em processamento de sinais, nós associamos o círculo unitário com um ciclo de amostragem, de modo que a frequência de amostragem *FS* é associada com *2π* radianos, e a frequência de Nyquist *FS/2* é associada com *π* radianos. Valores de Ω no semiplano superior, em unidades de Hz, então corresponde às frequências dentro da faixa de frequências do sinal contínuo.

Experimente:

f = 0.1;

t = 0:0.05:10;

x = exp(2\*pi\*j\*f\*t);

plot(t,real(x),t,imag(x),'r'); grid

title('\bfExponencial Complexa'); xlabel('\itt \rm(s)'); ylabel('Amplitude'); legend('Parte Real','Parte Imaginária');

axis([0 10 -1.1 1.1])

O Matlab reconhece as letras *j*  ou *i* como sendo a raiz quadrada de -1, desde que você não defina essas variáveis, *j*  ou *i,* com valores diferentes, por exemplo: j = 0; i = 15;

Funções do Matlab para manipulação de dados complexos: *real, imag, abs, angle*

1. **Salvando e Carregando Sinais**

Duas funções de entrada/saída de dados são especialmente úteis quando trabalhamos com variáveis Matlab:

* O comando *save* grava variáveis do espaço de trabalho (*workspace*) em um arquivo binário do Matlab (arquivo .*mat*). O arquivo é colocado no diretório corrente.
* O comando *load* lê variáveis de um arquivo de dados binários (arquivo .*mat*) para o espaço de trabalho (*workspace*).

Embora pouco especializados, esses comandos podem ser usados no gerenciamento cotidiano de seus cálculos no Matlab.

Experimente:

doc save

doc load

t = 0:0.1:10;

x1 = sin(t);

x2 = sin(2\*t);

x3 = sin(3\*t);

save variaveis

clear

load variaveis t x3

1. **Modulação e Demodulação**

A Modulação varia a amplitude, a fase, ou a freqüência de um sinal portador (portadora) da informação, o sinal mensagem. A função Matlab denominada *modulate* faz a modulação de um sinal mensagem com um determinado método de modulação. A sintaxe dessa função é:

y = modulate(x,fc,fs,’metodo’)

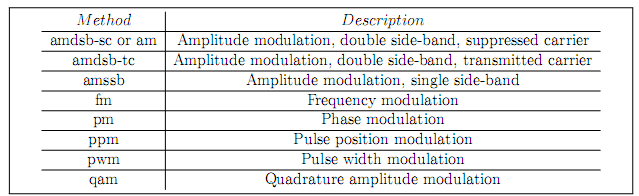
onde:

• *x* é o sinal mensagem.

• *fc* é a freqüência da portadora.

• *fs* é a freqüência de amostragem.

• *metodo* é um sinalizador para o método de modulação desejado (ver tabela).



A função Matlab *demod* faz a demodulação, isto é, ela obtêm a mensagem original a partir do sinal modulado. A sintaxe da função é:

x = demod(y,fs,fs,’method’)

O sinal x é atenuado em relação ao sinal y, porque a demodulação usa filtragem passa-baixas.

**Exercícios:**

1. Crie um sinal igual à soma de duas senóides com as seguintes características:
   * Duração de 3 segundos
   * Frequência de amostragem de 2 kHz
   * Senóide 1: frequência de 50 Hz (baixa), amplitude 10, fase nula.
   * Senóide 2: frequência de 950 Hz (alta), amplitude 1, fase nula.
2. Trace e ouça ao sinal produzido no exercício anterior.
3. O *script* (Matlab/Python) seguinte esboça dois sinais elementares contínuos no tempo, quais são?

t = -10:0.01:10; % tempo (abscissas)

y1 = (t >= 0); % sinal 1

y2 = 0.2\*t; % sinal 2

h = plot(t,y1,'r',t,y2,'b');

set(h,'linewidth',3);

xlabel('Tempo'); ylabel('Amplitude');

title('SINAIS CONTÍNUOS NO TEMPO'); grid

from numpy import arange

from pylab import plot, xlabel, ylabel, title, grid

t = arange(-10,10.01,0.01) # tempo (abscissas)

y1 = (t >= 0) # sinal 1

y2 = 0.2\*t # sinal 2

h = plot(t,y1,'r',t,y2,'b',linewidth=3)

xlabel('Tempo'); ylabel('Amplitude');

title(u'SINAIS CONTÍNUOS NO TEMPO'); grid('on')

1. Dado o código abaixo, estude as formas de onda dos sinais discretos no tempo.

y1 = ones(1,10); % sinal degrau unitário

y2 = 1:10; % sinal rampa

x = -4:5;

size(y1,1) % qtde de linhas do vetor x

size(y1,2) % qtde de colunas do vetor x

stem(x,y1,'r'); xlabel('Amostras'); ylabel('Amplitude');

title('SINAL DISCRETO NO TEMPO');

figure; stem(x,y2,'r'); xlabel('Amostras'); ylabel('Amplitude');

title('SINAL DISCRETO NO TEMPO');

from numpy import arange

from pylab import plot, xlabel, ylabel, title, grid

y1 = ones((1,10)) # sinal degrau unitário

y2 = arange(1,11) # sinal rampa

x = arange(-4,6)

y1.shape[0] # qtde de linhas do vetor x

y1.shape[1] # qtde de colunas do vetor x

stem(x,y1[0],'r'); xlabel('Amostras'); ylabel('Amplitude');

title('SINAL DISCRETO NO TEMPO');

figure(); stem(x,y2,'r'); xlabel('Amostras'); ylabel('Amplitude');

title('SINAL DISCRETO NO TEMPO');

1. Tomando por base os programas apresentados nos itens anteriores, diga o que muda na representação e exibição de sinais de tempo contínuo e discreto?
2. Trace o gráfico para o sinal contínuo no tempo .
3. Trace o gráfico para os sinais discretos no tempo  e .
4. Trace o gráfico do produto dos sinais gerados no exercício 3.