Exper.

1

**Sinais – Representação e Manipulação**

# Objetivo

Este laboratório descreve como usar ferramentas computacionais (Matlab e Python[[1]](#footnote-1)) para representar e manipular alguns sinais elementares:

* Criação e armazenamento
* Importação
* Amostragem e reamostragem
* Visualização
* Modelagem de ruído
* Modulação

# Fundamentação Teórica

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

Trabalharemos apenas com sinais unidimensionais (funções de uma variável) nos domínios dos números REAIS e COMPLEXOS.

# Procedimento Prático

1. **Criação e Armazenamento de Sinais**

Base de tempo:

Amplitude do sinal:

**Matlab**

t = [0.0 0.1 0.2 0.3]

x = [1.0 3.2 2.0 8.5]

**Python**

t = array([0.0,0.1,0.2,0.3])

x = array([1.0,3.2,2.0,8.5])

A construção principal de dados em *Matlab/Python* é o arranjo numérico, uma coleção ordenada de dados numéricos reais ou complexos, com uma ou mais dimensões. Os objetos de dados do processamento de sinais (sinais ou sequências unidimensionais, bidimensionais e multicanais) são todos naturalmente representados em arranjos (vetores e matrizes).

O *Matlab* e a *Ling. Python* representam sinais unidimensionais como vetores. Vetores são arranjos de dimensões 1 x *N* (vetor linha) ou *N* x1(vetor coluna), onde *N* é o número de amostras existentes na sequência. Para se armazenar uma sequência no *Matlab/Python* deve-se entrar uma lista de elementos na linha de comando. Para criar uma sequência de cinco valores numéricos reais em um **vetor linha** use os comandos:

**Matlab**

x = [1 2 3 4 5]

**Python**

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

A sequência pode ser convertida em um **vetor coluna** fazendo-se a transposição do vetor:

**Matlab**

y = x'

**Python**

y = array([x]).transpose()

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

Outra forma de se criar vetores de dados é usar o **operador dois pontos** do *Matlab* ou a função **range( )** do *Python*. Considere o tempo de um segundo de um sinal amostrado a 1000 Hz. O vetor de tempo apropriado para esse sinal seria dado por:

**Matlab**

t = 0:1e-3:1;

**Python**

t = arange(0,1+1e-3,1e-3)

Essas operações criam vetores linha de 1001 elementos, de 0 a 1 segundo, em passos de milésimos de segundos. Você também pode usar a função **linspace( )** para criar o mesmo vetor com os dados de tempo:

**Matlab**

t = linspace(0,1,1001);

**Python**

t = linspace(0,1,1001)

Digite os seguintes comandos:

**Matlab**

t1 = [0 .1 .2 .3];

t2 = 0:0.1:0.3;

t3 = linspace(0, 0.3, 4);

t = [t1; t2; t3;]';

x = sin(t)

**Python**

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

t2 = arange(0,**0.4**,0.1)

t3 = linspace(0, 0.3, 4)

t = array([t1,t2,t3]).transpose()

x = sin(t)

print x

O que faz o script mostrado acima?

1. **Sinais Amostrados**

Algumas fontes de sinais analógicas normalmente encontradas: geradores eletromagnéticos, geradores de áudio, sonar, corpo humano (sinais biomédicos) e outras. Sinais analógicos precisam ser amostrados e quantizados antes de serem processados digitalmente.

**Amostragem**: x[n] = xa(nTs)

onde: *x*[*n*] é um sinal discreto obtido a partir de um sinal analógico *xa*(*t*) amostrado com período *TS* (frequência de amostragem *fS* = 1/*TS*).

Execute o seguinte script:

**Matlab**

fs = 100; N = 1000;

tfinal = 9.99;

t1 = (0:N-1)/fs;

t2 = 0:1/fs:tfinal;

x1 = sin(2\*pi\*2\*t1);

x2 = sin(2\*pi\*3\*t2);

plot(x1), figure, plot(x2)

**Python**

fs = 100; N = 1000;

tfinal = 9.99;

t1 = arange(0,N,dtype=float)/fs

t2 = arange(0,10,1./fs,dtype=float)

x1 = sin(2\*pi\*2\*t1)

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

plot(x1)

plot(x2)

Quantos ciclos são mostrados em cada gráfico plotado pelo script acima?

Uma alternativa à criação de sinais é usar uma função das diversas caixas de ferramentas do *Matlab/Python* (toolbox/package). Uma variedade de funções geram formas de onda usuais. Cada uma delas requer um vetor que represente a base temporal. Algumas dessas funções serão descritas mais adiante.

**Aliasing**:

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

*fs > 2.fmáx*

A frequência máxima *fmáx* é conhecida por ***frequência de Nyquist***. Se a freqüência de amostragem não for maior que duas vezes a *frequência de Nyquist*, o sinal contínuo não poderá ser recuperado ao seu estado anterior. O sinal recuperado é denominado ‘*aliasing*’ por ser tratar de um sinal 'parecido' ao original.

Experimente o script:

**Matlab**

t = 0:0.001:2;

xa = sin(2\*pi\*5\*t);  **% sinal analóg.**

plot(t,xa), hold on

fs = 15;

ts = 0:1/fs:2;

xs1 = sin(2\*pi\*5\*ts);**% s. amostr.**

plot(ts,xs1,’bo-’)

fs = 7.5;

ts = 0:1/fs:2;

xs2= sin(2\*pi\*5\*ts); **% s. amostrado**

plot(ts,xs2,’ro-’), hold off

**Python**

t = arange(0,2,1e-3,dtype=float)

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

plot(t,xa)

fs = 15

ts = arange(0,2,1./fs,dtype=float)

xs1 = sin(2\*pi\*5\*ts) **# s. amostr.**

plot(ts,xs1,'bo-')

fs = 7.5

ts = arange(0,2,1./fs,dtype=float)

xs2 = sin(2\*pi\*5\*ts) **# s. amostrado**

plot(ts,xs2,'ro-')

Existe *aliasing* em algum dos sinais amostrados? Em qual? Por quê?

1. **Visualização de Sinais**

* Visualizar a amplitude do sinal em função do tempo
* Funções: **plot, stem, stairs**
* 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 for este o caso, veja os sinais graficamente ao invés de ouvi-los.

Experimente:

**Matlab**

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);

% .\* => mult. elemento a elemento

strips(x,0.25,fs)

sound(x,fs)

plot(ts,x)

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

**Python**

t = array([0.1,0.2,0.3,0.4])

x = array([1.0,8.0,4.5,9.7])

plot(t,x,'r')

stem(t,x)

bar(t,x)

fs = 1000;

ts = arange(0,2,1./fs,dtype=float)

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

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

figure()

plot(ts,x)

plot(ts[0:200],x[0:200])

**Matlab**: Para que serve o comando strips? (dica: digite >> ’*help strips*’ ou >>> *help*(*bar*).)

Para que serve o operador .\* ?

**Python**: O que faz o comando plot(ts[0:200],x[0:200]) ?

Execute o script mostrado a seguir e indique o resultado gerado pelo mesmo.

from matplotlib.pylab import plot, setp, xlabel, ylabel, title, \

grid, figure, draw, gca, legend

from numpy import arange

plot(range(10))

xlabel('medido')

ylabel('calculado')

title('Medido vs. Calculado')

grid(True)

meugraf = gca() # pega uma referência do gráfico (plot)

linha = meugraf.lines[0] # pega ref. da primeira linha

linha.set\_marker('o') # ajusta propriedades usando mét.s 'set\_algumacoisa'

setp(linha, color='g') # ou usando a função 'setp'

draw() # aplica novas propriedades e redesenha a figura

x = arange(100)

linear = arange(100)

quadrado = [v \* v for v in arange(0, 10, 0.1)]

plot(x, linear, x, quadrado) # adiciona linhas em um gráfico

legend(('linear', 'quadrado'))

figure() # limpa o gráfico antigo (clf)

plot(x, linear, 'g:+', x, quadrado, 'r--o')

leg = legend(('linear', 'quadrado'), loc='upper left')

fonte: <http://www.tp.umu.se/~nylen/pylect/intro/matplotlib/matplotlib.html>

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

A aplicação da caixa de ferramentas de processamento de sinais do Matlab, 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

• Freqüência, 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 freqüência de amostragem *FS* é associada com *2π* radianos, e a freqüência de Nyquist *FS/2* é associada com *π* radianos. Valores de Ω no semi-plano superior, em unidades de Hz, então corresponde às freqüências dentro da faixa de freqüências 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’)

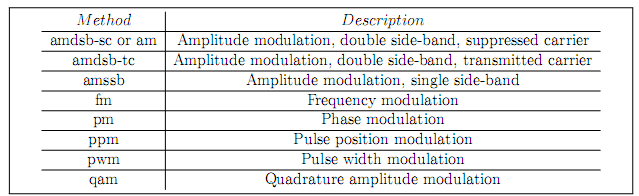
onde:

• *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:
   1. Duração de 3 segundos
   2. Freqüência de amostragem de 2 kHz
   3. Senóide 1: freqüência de 50 Hz (baixa), amplitude 10, fase nula.
   4. Senóide 2: freqüência de 950 Hz (alta), amplitude 1, fase nula.
2. Veja e ouça ao sinal usando, inicialmente, um arquivo *.m*, e posteriormente usando o *SPTool*.
3. As linhas de código mostradas abaixo esboçam dois sinais elementares contínuos no tempo, quais são?

**Matlab**

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('SINAL CONTÍNUO NO TEMPO'); grid

# -\*- coding: utf-8 -\*-

**Python**

from numpy import arange

from matplotlib.pylab import plot, xlabel, ylabel, title, grid, setp

t = arange(-10,10.01,.01) # tempo (abscissas)

y1 = (t >= 0) # sinal 1

y2 = 0.2\*t # sinal 2

l1,l2 = plot(t,y1,'r',t,y2,'b',linewidth=2)

setp(l1,linewidth=3)

# >>> setp(h, linewidth=2, color='r')

# >>> setp(h, 'linestyle')

# linestyle: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' ]

# >>> getp(line, 'linewidth')

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

size(x,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');

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 no Matlab/Python?
2. Trace o gráfico para o sinal contínuo no tempo .
3. Trace o gráfico para o sinal discreto no tempo .
4. Trace o gráfico para o sinal discreto no tempo .
5. Trace o gráfico do produto dos sinais gerados no exercício 3.

1. Ambiente IPython (www.ipython.org) [↑](#footnote-ref-1)