Las propiedades en Matlab se definen según ciertos atributos entre ellos están los siguientes:

\****Hidden***: Determina si una propiedad se va a mostrar o no en una lista de propiedades que sale en el prompt. Si es *true* no se muestra de lo contrario si es *false* se muestra por pantalla. Por defecto es false.

\****Access***: este atributo refiere al acceso de las propiedades, y tiene prácticamente tres tipos:*public* su acceso no está restringido, *protected* solo puede ser accedida por clases o subclases y *private* solo puede ser accedida por los miembros de la clase únicamente (no por subclases). Por defecto es public.

\****Dependent***: este tipo de propiedades no almacenan datos, ya que una propiedad *dependent* depende del estado actual de otra, como por ejemplo el valor concreto de una propiedad. Si se define a una propiedad como dependent(solo con escribir *dependent*), Matlab no muestra por el prompt el nombre y valor de la variable, si Hidden es falso, hasta que no tenga un método *get*.

Si está definido el constructor al inicializar primero pasa por el mismo y luegosigue con el resto... al definir las propiedades también se pasa por el método get de las mismas.

CLASE **samdirAudio**

La clase *samdirAudio* va a contener los datos en el dominio del tiempo y en el dominio de la frecuencia. Para ellova a necesitar contener en variables determinados datos, por ejemplo *mDomain* nos va a especificar si los datos los tenemos en el tiempo o en la frecuencia.

La variable *mEvenSamples* nos indica si la señal es par o impar.

Contener los datos en el dominio de la frecuencia nos facilita realizar las operaciones, como la convolucion y correlacion.

ATRIBUTOS

**Properties(Access = private, Hidden = true)** %variables internas controladas por variables dependientes

**mDimensions = 0;**

**mData = [];** % alternativesyntax, equals .timeData.'

**mDomain = 'time';** % 'time' / 'freq'

**mSamplingRate = 44100;**

**mSignalType = 'power';** % 'power' / 'energy'

El tipo de señal es importante a la hora de calcular la fft. Obtendremos una señal de potencia (potencia finita, energia infinita, señales periodicas) por ej. para el habla, el ruido y una señal de energia (potencia = 0) por ej. para la respuesta al impulso.

**mEvenSamples = true;** %Si la señal es par o impar. Es necesaria para calcular la fft.

**mDataType = 'double'**; % Typeforinternal data

**mDataTypeOutput = 'double';** % Typeusedfor output

**mDataTypeEqual = true;** % On/Off Switchfor data typecasts (speedreason)

**mDataFactor = 1;** % Factor forscalingifusingint

**properties(Constant, Hidden = true)**

%El constant refiere a que en todas las instancias de la clase

%samdirAudio esta variable va a tener un solo valor.

%Para hacer referencia a la misma se usa el nombre de la clase + el

%operador de acceso: samdirAudio.VALID\_SIGNAL\_TYPES

VALID\_SIGNAL\_TYPES = {'power', 'energy'};

**properties(Dependent = true, Hidden = false)**

Estas variables son de tipo *dependent* por lo tanto para ser vista necesitan tener definido por lo menos el método *get*. Para estas están definidos los métodos *get/set*. Si se usan estas propiedades al nombrarla se llama al método *get*. Es decir si en un método cualquiera uso *this.samplingRate* cuando se pasa por esa línea se llama al método *get.samplingRate*.

samplingRate % samplingRate in Hertz

signalType % energy (filter) orpower (signal) normalizationfor FFT/IFFTsignalType

fftDegree % 2^fftDegree samples in time domain

trackLength % secondslong

wavenumber % k = 2\*pi\*f/c

nBins %number of frequencysamplescalledbins

nSamples % number of time samples está definida en itaSuper

time %time domain data (get/set) - full dimensions

timeData %time domain data in 2D

freq %frequencydomain data (get/set) - full dimensions

freqData %frequencydomain data in 2D

domain % Agrego properties of itaSuper %domain of data ('time'/'freq')

freqAmp

dimensions % dimensions of the data field

dataFactor

dataType %raw data type, data isstored in... (single/double)

dataTypeOutput %data typewhenyouaccess data (single/double), conversion

**properties(Dependent = true, Hidden = true)**

En esta variable van a estar los datos en crudo, y se puede acceder por medio del *set* y *get* de las variables *frecData* y*timeData*. Para obtenerla en la frecuencia usa la FFT y para obtenerla en el tiempo la IFFT. Ver bien cual es la funcionfftque se utiliza en este caso.

data %raw data, could be time orfrequencydomain (CAREFUL!)

METODOS

Los métodos usan por dentro cuatro identificadores respecto a lo que son variables de entrada y variables de salida.

*varargin* significa que es una estructura que contiene parámetros que se le pasa a la función, si son varios para acceder a los mismos se usa varargin{index},

*nargin* indica el número de argumentos que la funcion recibió.

*varargout* significa las variables de salida que va a sacar la función,

*nargout* indica el número de objetos de salida.

Por ejemplo si llamamos a una función *>>ita\_portaudio\_run()* sin recibir en una variable la salida de la función va a realizar la función de *play* pero si escribimos *>>result = ita\_portaudio\_run()* realizará la función record and play.

% esto se puede observar en las lineas de codigo de la función ita\_portaudio\_run():

% 93 ifnargout> 0

% 94 record = true;

% 95 else

% 96 record = false;

% 97 end

**this = samdirAudio(varargin)**

**result=get.spk(this)**

**this=set.spk(this,value)**

result = get.time(this)

result = reshape(this.timeData, [this.nSamplesthis.dimensions]);

this = set.time(this,value)

result = get.freq(this)

this = set.freq(this,value)

**result = get.timeData(this)**

**this = set.timeData(this,value)**

**result = get.freqData(this)**

**this = set.freqData(this,value)**

**result = get.freqAmp(this)**

**this = set.freqAmp(this,value)**

**result = get.data(this)**

**this = set.data(this,value)**

**result = get.domain(this)**

**this=set.domain(this,value)**

**result = get.samplingRate(this)**

**this = set.samplingRate(this,value)**

**result = get.signalType(this)**

**this = set.signalType(this,value)**

**result=get.fftDegree(this)**

**this=set.fftDegree(this,value)**

**result = get\_data(this)**

**this = set\_data(this,value)**

result = get.dimensions(this)

this = set.dimensions(this,value)

**result = get.dataType(this)**

**set = set.dataType(this)**

**result = get.dataTypeOutput(this)**

**this = set.dataTypeOutput(this,value)**

**result = get.dataFactor(this)**

**this = set.dataFactor(this,value)**

**result = isFreq(this)**

**result = isEvenSamples(this)**

**result = isPower(this)**

**result = isEnergy(this)**

**result = isempty(this)**

**result = timeVector(this,idx)**

**result = freqVector(this, index)**

**result = get.trackLength(this)**

**this = set.trackLength(this, value)**

**result = get.nBins(this)**

**result = get\_nBins(this)**

**nBins = nSamples2nBins(this)**

**result = get.nSamples(this)**

**this = set.nSamples(this,value)**

**result = get\_nSamples(this)**

**nSamples = nBins2nSamples(this)**

**result = isTime(this)**

varargout = plot(this)

play(this)

varargout = plot\_freq(this)

plot\_spk(this,varargin)

**result=get.allowDbplot**

**this=set.allowDbplot**

Los que no están resaltados en negrita no están anotados en tu hoja :O

**cal\_fft**

Llama a *samdir\_fft()* asegurándose que las dimensiones de los argumentos de entrada y salida coincidan.

**cal\_ifft**

Llama a *samdir\_ifft()* asegurándose que las dimensiones de los argumentos de entrada y salida coincidan.

**FUNCIONES**

**SEÑALES DE EXCITACIÓN**

**samdir\_generate**

Se pueden generar tonos, ruidos, impulsos o barridos. Para las primeras, la sintaxis es la siguiente:

samdirAudio = samdir\_generate('tipo', Amplitud,(Frec),sr, fft\_degree)

Por ejemplo,

a = samdir\_generate('impulse',1,44100,15)

a = samdir\_generate('impulsetrain',1,44100,15,cantidad de impulsos)

a = samdir\_generate('sine',1,1000,44100,15)

a = samdir\_generate('cosine',1,1000,44100,15)

a = samdir\_generate('whitenoise',1,44100,15)

a = samdir\_generate('pinknoise',1,44100,15)

Para barridos la sintaxis varía:

b = samdir\_generate(sweeptype,freq\_range,[stopmargin],sr, fft\_degree)

*sweeptype*: puede ser *linsweep* (lineal) o *expsweep/logsweep*(exponencial)

*freq\_range*: vector con la frecuencia inicial y la final de la forma [f1 f2]

*stopmargin*: segundos de silencio al final del barrido

*sr*: frecuencia de muestreo en Hz

*fft\_degree*: grado de la FFT, lo que indica la duración de la señal.

Ejemplo:

b = samdir\_generate('expsweep',[2 22000],44100,18);

Genera el barrido senoidal exponencial de 22Hz a 22kHz, con una frecuencia de muestreo de 44100Hz y una longitud de 262.144 muestras, lo que sería una duración de 5.94seg. Se realiza con estas longitudes múltiplo de 2^N para que el algoritmo de la FFT sea más eficiente (ejemplo software Dirac de B&J).

Para generar la señal utiliza la función chirp() de Matlab, la cual tiene la siguiente sintaxis:

x = chirp(T,F0,T1,F1,method,PHI)

F0 y F1 son los límites en frecuencia del barrido

T1 es la duración del mismo

T es el vector de tiempo, con N samples

Method es la forma de variación, sólo usaremos las dos comentadas arriba

PHI es el desfasaje inicial, samdir\_generate le pone 90 y empieza negativo

El margen de silencio al final de señal puede utilizarse en ciertas mediciones repetitivas para dar un tiempo de decaimiento a los sistemas. Para implementar este silencio, se calcula el barrido para una duración igual a (duración total – silencio) y luego se agrega un vector de ceros. En el caso de utilizar deconvolución lineal, es necesario generar el filtro inverso de la señal de excitación.

**samdir\_inverso**

Realiza la inversión de las muestras de la señal en el dominio del tiempo, y luego le aplica una envolvente de -10db/década para compensar la distribución de energía en el espectro. La sintaxis es:

samdirAudio = samdir\_inverso(samdirAudio)

Ejemplo:

b = samdir\_inverso(a)

Donde ‘a’ es un barrido senoidal exponencial y ‘b’ su filtro inverso.

**samdir\_ventaneo**

Luego de generada la señal, se realiza un ventaneo en el tiempo utilizando las ventanas que trae incluido Matlab, mediante la función samdir\_ventaneo.m. Las ventanas disponibles en la función realizada son Hann, Rectangular, Hamming, Flat Top, Blackman, Bartlett, Bartlett-Hann, Blackman-Harris, Bohman, Chebyshev, Gaussian, Kaiser, Nuttall, Parzen, Taylor, Tukey, Triangular. Si se utilizó margen de silencio en la señal original es necesario pasarlo como argumento así la ventana se aplica a la señal y no al silencio (lo que no tendría sentido). La sintaxis es la siguiente:

samdirAudio= samdir\_ventaneo('tipo’, inic, fin, samdirAudio,[stopmargin])

Ejemplo:

a = samdir\_ventaneo('hann',0.1,0.1,y,0.2)

tipo: ventana

inic: cantidad de segundos al inicio

fin: cantidad de segundos al final

samdirAudio: señal del tipo samdirAudio

stopmargin: silencio al final de la señal, si lo tiene.

Una vez generada la señal, es necesario reproducirla y grabar la respuesta. Para ello se utiliza la función *samdir\_reproduce\_graba()*.

**ENTRADAS Y SALIDAS DE AUDIO**

Utilizan las funciones *audiodevinfo()*, *audiorecorder()* y *audioplayer()* (incluidas en Matlab) para manejar los dispositivos de audio del sistema.

**samdir\_disp\_audio**

Reconoce los dispositivos de audio disponibles y los muestra en pantalla (crea un struct donde se puede acceder a su nombre e ID para la GUI).

samdir\_disp\_audio()

**samdir\_reproduce**

Reproduce el objeto de tipo samdirAudio. También es llamada por el método *play* de la clase samdirAudio.

Ejemplo:

samdir\_reproduce(a)

a.play

**samdir\_graba**

Graba durante un cierto tiempo que es recibido como argumento de entrada. Se utiliza para calcular el ruido del sistema o detectar si está bien configurado el dispositivo de entrada. Devuelve un objeto de tipo samdirAudio.

Ejemplo:

a = samdir\_graba(5)

**samdir\_reproduce\_graba**

Reproduce una señal ingresada como argumento y devuelve la grabación. Debido a que la latencia del sistema es variable, se debe realizar un desfasamiento de la señal de excitación para no perder información en la grabación. Luego se realiza una correlación entre ambas señales y se ponen en fase nuevamente.

samdirAudio = samdir\_reproduce\_graba(samdirAudio)

Ejemplo:

b =samdir\_reproduce\_graba(a)

**PROCESAMIENTO DE LA SEÑAL**

**samdir\_fft**

Calcula el espectro de una señal dada en el dominio del tiempo. Se descartan las frecuencias negativas ya que no tienen sentido físico. Para obtener la magnitud correcta de las componentes en frecuencia es necesario normalizar dividiendo la salida de la FFT entre el valor N del total de samples.

Además, como sólo utilizamos la mitad del espectro (frecuencias positivas), las amplitudes deben ser multiplicadas por 2, menos las correspondientes a DC y Frecuencia Nyquist (fs/2). A su vez, los valores en amplitudes deben ser pasados a valor eficaz, dividiendo por sqrt(2) (menos el correspondiente a DC)

Entonces, DC---> igual, fs/2 ---> / sqrt(2).

Las restantes ---> \* 2 / sqrt(2) = \* sqrt(2)

Las señales de energía no se modifican.

samdirAudio = samdir\_fft(samdirAudio)

**samdir\_ifft**

Calcula la información en el tiempo de una señal dado su su espectro normalizado. Se revierte el efecto de la normalización, se agregan las frecuencias negativas para reconstruir el espectro y luego se realiza la IFFT, devolviendo el valor en el tiempo.

samdirAudio = samdir\_ifft(samdirAudio)

**samdir\_plot**

Grafica la señal en el dominio del tiempo. Utiliza el campo 'comment' para ponerle título a la gráfica, y modifica los ejes para una mejor visualización.

**samdir\_plot\_freq**

Grafica la señal en el dominio de la frecuencia. Por ahora utiliza la información de amplitud en dB y grafica utilizando la función semilogx () de Matlab, haciendo que el eje horizontal esté en forma logarítmica.

**samdir\_plot\_freq\_total**

Grafica la señal en el dominio de la frecuencia, módulo y fase. Utiliza subplots, presentando el módulo arriba y la fase abajo. La fase en radianes la pasa a grados sexagesimales y luego gráfica. Falta modificar.