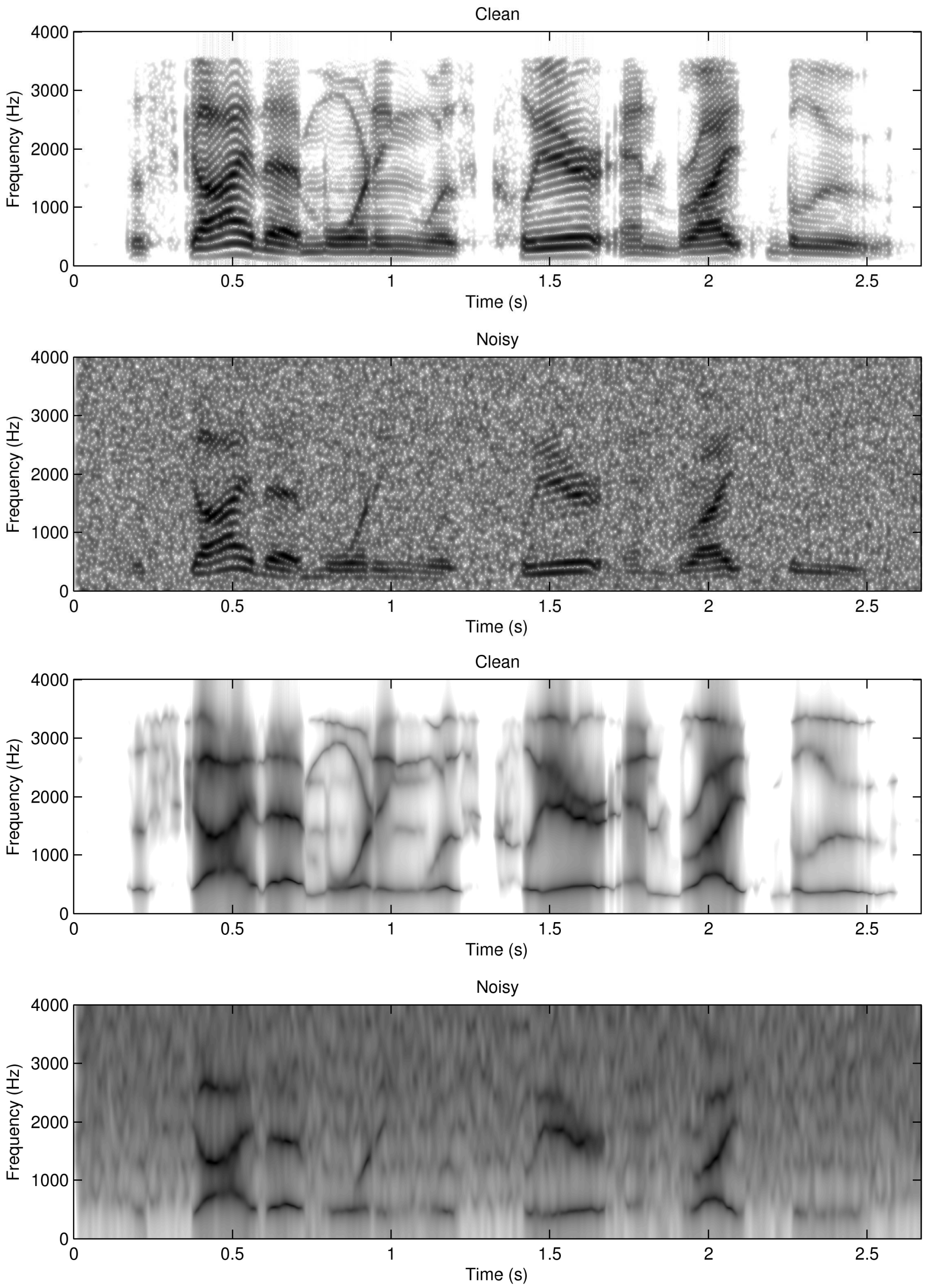
myspectrogram.m

Kamil Wojcicki

November 25, 2010

Usage:

1. Start Matlab
2. Run demo by typing: `test_myspectrogram`



```matlab
%------------------------------------------------------------------------------------------------------------------------
%
% @file         myspectrogram.m
% @date         17/09/2007
% @author       Kamil Wojcicki
% @affiliation  Signal Processing Laboratory, Griffith University, Nathan QLD4111, Australia
% @brief        Spectrogram function
%------------------------------------------------------------------------------------------------------------------------
%
% @inputs       speech  - time domain speech signal vector
%               fs      - sampling frequency (Hz), f.e. 8000
%               T       - vector of frame width, Tw, and frame shift, Ts, in milliseconds, i.e. [Tw, Ts]
%               w       - analysis window handle, f.e. @hamming
%               nfft    - fft analysis length, f.e. 1024
%               Slim    - vector of spectrogram limits (dB), i.e. [Smin Smax]
%               alpha   - fir pre-emphasis filter coefficients, f.e. [1 -0.97]
%               cmap    - color map ('default', 'gray', 'bone', 'copper', 'hot', 'jet')
%               cbar    - color bar (boolean)
%               type    - estimator or feature type ('lp', 'per')
%
% @output       handle  - plot handle
%------------------------------------------------------------------------------------------------------------------------
%
% @usage        [handle] = myspectrogram(speech, fs, T, w, nfft, Slim, alpha, cmap, cbar, type);
%
% @examples     [handle] = myspectrogram(speech, 8000, [18 1], @hamming, 1024, [-45 -2], false, 'default', false, 'lp');
%               [handle] = myspectrogram(speech, 8000, [18 1], @hamming, 1024, [-45 -2], [1 -0.97], 'default', true, 'per');
%------------------------------------------------------------------------------------------------------------------------
function [handle] = myspectrogram(s, fs, T, w, nfft, Slim, alpha, cmap, cbar, type)

    %--------------------------------------------------------------------------------------------------------------------
    % VALIDATE INPUTS, SET DEFAULTS
    switch nargin
    case 1, type='per'; cbar=false; cmap='default'; alpha=false; Slim=[-59,-1]; w=@hamming; T=[18,1]; nfft=1024; fs=8000;
    case 2, type='per'; cbar=false; cmap='default'; alpha=false; Slim=[-59,-1]; w=@hamming; T=[18,1]; nfft=1024;
    case 3, type='per'; cbar=false; cmap='default'; alpha=false; Slim=[-59,-1]; w=@hamming; T=[18,1];
    case 4, type='per'; cbar=false; cmap='default'; alpha=false; Slim=[-59,-1]; w=@hamming;
    case 5, type='per'; cbar=false; cmap='default'; alpha=false; Slim=[-59,-1];
    case 6, type='per'; cbar=false; cmap='default'; alpha=false;
    case 7, type='per'; cbar=false; cmap='default';
    case 8, type='per'; cbar=false;
    case 9, type='per';
    case 10
    otherwise, error('Invalid number of input arguments.');
    end

    %--------------------------------------------------------------------------------------------------------------------
    % DECLARE VARIABLES
    if(isstr(s)), [s, fs] = wavread(s); end; % read audio data from file
    Tw = T(1);                          % frame width (ms)
    Ts = T(2);                          % frame shift (ms)
    Nw = round(fs*Tw*0.001);            % frame width (samples)
    Ns = round(fs*Ts*0.001);            % frame shift (samples)
    N  = length(s);                     % length of speech signal (samples)
    Smin = Slim(1);                     % lower normalized dynamic range limit
    Smax = Slim(2);                     % upper normalized dynamic range limit
    if(isstr(w)), w = str2func(w); end; % obtain window function handle from string input

    %--------------------------------------------------------------------------------------------------------------------
    % SPEECH PREPROCESSING
    if(islogical(alpha) && alpha), s = filter([1 -0.95],1,s);   % apply a typical preemphasis filter
    elseif(¬islogical(alpha)) s = filter(alpha,1,s); end;       % apply custom preemphasis filter

    %--------------------------------------------------------------------------------------------------------------------
    % GET SPECTROGRAM DATA
    %[S,F,T] = spectrogram(s,w(Nw).',Nw-Ns,nfft,fs);            % MATLAB's new spectrogram function
    %[S,F,T] = specgram(s,nfft,fs,w(Nw).',Nw-Ns);               % MATLAB's depreciated spectrogram function
    [S,F,T] = toframes(s,w,T,fs,nfft,type);                     % Framing function, use this if you do not have the Signal Processing Toolbox

    %--------------------------------------------------------------------------------------------------------------------
    % SET DYNAMIC RANGE
    S = abs(S);                         % compute magnitude spectrum
    S = S/max(max(S));                  % normalize magntide spectrum
%    S(S<eps) = eps;                     % prevent zero spectral magnitude values
    S = 20*log10(S);                    % compute power spectrum in dB

    %--------------------------------------------------------------------------------------------------------------------
    % PLOT AND LABEL RESULTS
    handle = imagesc(T, F, S, [Smin Smax]);
    %handle = imagesc(T, F, S, 'CDataMapping', 'direct');
    axis('xy');
    axis([0 N/fs  0 fs/2]);
%    xlabel('time (s)', 'FontSize', 8, 'FontWeight', 'n');
%    ylabel('frequency (Hz)', 'FontSize', 8, 'FontWeight', 'n');
%    set(gca,'YDir','normal', 'FontSize', 6);

    if(cbar), colorbar('FontSize',6); end

    %--------------------------------------------------------------------------------------------------------------------
    % DEFINE CUSTOM COLOR MAPS
    switch(lower(cmap))
    case {'default'}
        colormap('gray');
        map=colormap;
        colormap(1-map);
    otherwise, colormap(cmap);
    end
```

```
%--------------------------------------------------------------------------------------------------------------------
%
% @author       Kamil Wojcicki
% @date         April 2007
% @revision     001
% @brief        Implements toframes part of analysis-modification-synthesis (AMS) framework
%--------------------------------------------------------------------------------------------------------------------
function [S,F,T] = toframes(s,w,T,fs,nfft,type)

    %----------------------------------------------------------------------------------------------------------------
    % VALIDATE INPUTS
    switch nargin
    case 1, type='per'; nfft=1024; fs=8000; T=[32 32/8]; w={@hamming, @hamming};
    case 2, type='per'; nfft=1024; fs=8000; T=[32 32/8];
    case 3, type='per'; nfft=1024; fs=8000;
    case 4, type='per'; nfft=1024;
    case 5, type='per';
    case 6
    otherwise, error('Invalid number of input arguments.');
    end

    %----------------------------------------------------------------------------------------------------------------
    % DEFINE VARIABLES
    if(isstr(s)) [s, fs, nbits] = wavread(s); else, nbits=16; end;

    s = s(:).';
%    s = s-mean(s);
    smax = max(abs(s))/0.999;
    s = s/smax;

    Tw = T(1);                                                   % frame length [ms]
    Ts = T(2);                                                   % frame frameshift [ms]
    N  = round(fs*Tw*0.001);                                     % frame length [samples]
    Z  = round(fs*Ts*0.001);                                     % frame shift [samples]
    D  = mod(length(s), Z);                                      % add N-D zeros to the end
    G  = (ceil(N/Z)-1)*Z;                                        % add G zeros to the beginning
    s  = [zeros(1,G) s zeros(1,N-D)];                            % zero pad signal to allow an integer number of segments
    ss = length(s);                                              % length of the signal for processing
    t  = [0:ss-1]/fs;                                            % time vector
    M  = ((ss-N)/Z)+1;                                           % number of overlapping segments
    if(isstr(w)), w=str2func(w); end;
    wa = w(N).';                                                 % analysis window A (for magnitude component)
    wsyn = 0.5-0.5*cos((2*pi*((0:N-1)+0.5))/N);                  % synthesis window


    %----------------------------------------------------------------------------------------------------------------
    % SPLIT SPEECH INTO OVERLAPPED FRAMES (EACH ROW IS A FRAME), AND WINDOW THE FRAMES
    indf = Z*(0:(M-1)).';                                        % indexes for frames
    inds = (1:N);                                                % indexes for samples
    refs = indf(:,ones(1,N)) + inds(ones(M,1),:);                % sample indexes for each frame

    segments_s = s(refs);                                        % split into overlapped frames (using indexing)
    segments_sm = segments_s .* wa(ones(M,1),:);                 % apply magnitude spectrum analysis window


    %----------------------------------------------------------------------------------------------------------------
    % PERFORM COMPLEX FOURIER SPECTRUM ANALYSIS
    F = [0:nfft-1]/(nfft-1)*fs;
    T = [0:M-1]/(M-1)*ss/fs;

    switch(lower(type))
    case {'per','periodogram'}
        S = fft(segments_sm, nfft, 2);                           % short-time Fourier analysis
        S = abs(S).^2/N;                                         % periodogram PSD estimates
        S = sqrt(S);                                             % magnitude spectrum (for consistency)
    case {'lp','lpc','ar'}
        p = 12;                                                  % order of AR model
        [A,E] = lpc(segments_sm.',p);
        S = repmat(sqrt(E),1,nfft)./abs(fft(A,nfft,2));          % LP-based (AR) magnitude spectrum
    otherwise
    end

    S = S.';
    F = F.';
    T = T;


%--------------------------------------------------------------------------------------------------------------------
%
% @author       Kamil Wojcicki
% @date         April 2007
% @revision     001
% @brief        Computes AR model parameters
%--------------------------------------------------------------------------------------------------------------------
function [apk, Gp2] = lpc(x,p)

    %----------------------------------------------------------------------------------------------------------------
    if nargin==1, p = length(x)-1; end;
    [nR,nC] = size(x);
    if min([nR nC])==1, x = x(:); end;

    %----------------------------------------------------------------------------------------------------------------
    if min([nR nC])==1
        apk = zeros(1,p+1);
        [apk(2:end) Gp2]=levinson_durbin(autocorr(x,p), p);
        apk(1) = 1;
    else
        apk = zeros(nC,p+1);
        Gp2 = zeros(nC,1);
        for n = 1:nC
            [apk(n,2:end) Gp2(n)]=levinson_durbin(autocorr(x(:,n),p), p);
        end
        apk(:,1) = ones(nC,1);
    end
```

```matlab
%------------------------------------------------------------------------------------------------------------------------
%
% @author       Kamil Wojcicki
% @date         Aug 2005
% @revision     001
% @brief        Uses Levinson-Durbin algorithm to efficiently solve Yule-Walker set of linear AR equations.
%
% @inputs       Rxx - autocorrelation values of AR random process x(n)
%               p   - order of AR model
%
% @outputs      apk - linear prediction coefficients
%               Gp2 - AR model gain squared (Gp^2)
%------------------------------------------------------------------------------------------------------------------------
%
% @usage        [apk, Gp2]=levinson_durbin(Rxx, p)
%------------------------------------------------------------------------------------------------------------------------
function [apk, Gp2]=levinson_durbin(Rxx, p)

    %--------------------------------------------------------------------------------------------------------------------
    % VALIDATE INPUT
    if(nargin≠2), error('Wrong number of input parameters.'); end

    %--------------------------------------------------------------------------------------------------------------------
    % LEVINSON DURBIN ALGORITHM
    a(1,1)=-Rxx(2)/Rxx(1);     % initialise algorithm
    P(1)=Rxx(1)*(1-a(1,1)^2);  % initialise algorithm

    for m=2:p
        a(m,m)=(Rxx(m+1)+a(m-1,1:m-1)*Rxx(m:-1:2))/-P(m-1);
        for i=1:m-1
            a(m,i)=a(m-1,i)+a(m,m)*a(m-1,m-i);
        end
        P(m)=P(m-1)*(1-a(m,m)^2);
    end

    apk=a(p,:);
    Gp2=P(p);

%------------------------------------------------------------------------------------------------------------------------
%
% @author       Kamil Wojcicki
% @date         Aug 2005
% @revision     001
% @brief        Computes autocorrelation coefficients of the input signal
%
% @inputs       x   - input signal (by default assumed of length N)
%               N   - how many samples of signal x to use for computations
%               p   - p+1 autocorrelation coefficients will be computed
%
% @outputs      Rxx     - autocorrelation coefficients
%               *none   - autocorrelation function gets plotted
%------------------------------------------------------------------------------------------------------------------------
%
% @usage        [Rxx]=autocorr(x, p)
%               [Rxx]=autocorr(x, N, p)
%               [varargout]=autocorr(x, varargin)
%------------------------------------------------------------------------------------------------------------------------
function [varargout]=autocorr(x, varargin)

    %--------------------------------------------------------------------------------------------------------------------
    % VALIDATE INPUTS
    switch nargin
    case 1, N=length(x); p=12; if(nargout==0), p=256; end
    case 2, N=length(x); p=varargin{1};
    case 3, N=varargin{1}; p=varargin{2};
    otherwise, error('Wrong number of input parameters.');
    end

    %--------------------------------------------------------------------------------------------------------------------
    % AUTOCORRELATION
    Rxx=zeros(p+1,1);
    for i=0:p
        for n=0:N-i-1
            Rxx(i+1)=Rxx(i+1)+x(n+1)*x(n+i+1);
        end
    end
    Rxx=Rxx/N;

    %--------------------------------------------------------------------------------------------------------------------
    % GENERATE THE OUTPUT VECTOR
    switch nargout
    case 0
        hfig = figure('Position',[50 50 1024 768],'PaperPositionMode','auto');
        plot([0:p],Rxx);
        xlabel('lag k'); ylabel('Rxx'); title('Autocorrelation function of signal x');
        axis([0 p 1.05*min(Rxx) 1.05*max(Rxx)]);
%       print('-depsc2', 'eps/autocorr.eps');
    case 1, varargout(1) = {Rxx};
    otherwise, error('Too many output parameters.');
    end


%------------------------------------------------------------------------------------------------------------------------
%
% @author       Kamil Wojcicki
% @date         Aug 2005
% @revision     001
% @brief        Hamming window function
%------------------------------------------------------------------------------------------------------------------------
function [w] = hamming(N)

    w = 0.54-0.46*cos(2*pi*[0:(N-1)].'/(N-1));

%------------------------------------------------------------------------------------------------------------------------
% EOF
```

```
%------------------------------------------------------------------------------------------------------------------------
% myspectrogram test framework by Kamil Wojcicki, 2010 (test_myspectrogram.m)
clear all; close all; % clc;

    titlecase = @(str) ( sprintf('%s%s',upper(str(1)),str(2:end))); % make a word title case

    file.clean = 'sp10.wav';
    file.noisy = 'sp10_white_sn5.wav';

    [speech.clean, fs, nbits] = wavread(file.clean);
    [speech.noisy, fs, nbits] = wavread(file.noisy);

    % speech.enhanced = some_enhancement_method(speech.noisy, "other method parameters go here...");

    methods = fieldnames(speech); % method names
    M = length(methods); % number of methods


    % PLOT SPECTROGRAMS (PERIODOGRAM SPECTRUM)
    figure('Position', [20 20 800 250*M], 'PaperPositionMode', 'auto', 'Visible', 'on');
    for m = 1:M % loop through treatment types and plot spectrograms
        method = methods{m};
        subplot(M,1,m);
        %myspectrogram(speech.(method), fs); % use the default options
        myspectrogram(speech.(method), fs, [22 1], @hamming, 2048, [-59 -1], false, 'default', false, 'per'); % or be quite specific about what you want
        title(titlecase(method));
        xlabel('Time (s)');
        ylabel('Frequency (Hz)');
    end
    print('-depsc2', '-r250', sprintf('%s.eps', mfilename));
    print('-dpng', sprintf('%s.png', mfilename));


    % PLOT SPECTROGRAMS (AUTOREGRESSIVE SPECTRUM WITH PREEMPHASIS)
    figure('Position', [20 20 800 250*M], 'PaperPositionMode', 'auto', 'Visible', 'on');
    for m = 1:M % loop through treatment types and plot spectrograms
        method = methods{m};
        subplot(M,1,m);
        myspectrogram(speech.(method), fs, [32 1], @hamming, 2048, [-59 -1], [1 -0.97], 'default', false, 'lp');
        title(titlecase(method));
        xlabel('Time (s)');
        ylabel('Frequency (Hz)');
    end
    print('-depsc2', '-r250', sprintf('%s_lp.eps', mfilename));
    print('-dpng', sprintf('%s_lp.png', mfilename));


    % WRITE TO AUDIO FILES
    for m = 1:M % loop through treatment types and write to wavs
        method = methods{m};
        audio.(method) = 0.999*speech.(method)./max(abs(speech.(method)));
        wavwrite(audio.(method), fs, nbits, sprintf('%s.wav',method));
    end

    fprintf('Enjoy! :)\n');

%------------------------------------------------------------------------------------------------------------------------
% EOF
```