```
function [D_GATP,R_GATP]=DRS_GATP_JU(x,fs,speed,load,nfft)

x=x(:)';
dt=1/fs;

fi=cumsum(speed)*dt;
fi=fi-fi(1); % Phase starts from zero

[xn,xr,xref,do,fi2,t1,t2]=inst_norm_JU(x,fs,speed,load);

f=linspace(0,do,length(xr));

[stft,t,fn]=STFT_FILTERBANK_JU(xn,do,nfft);

stftS=stft(2:end,:);
stft(end,:)=[];

G1=stft.*conj(stftS);
G2=abs(stft.^2);

G=mean(G1,1)./mean(G2,1);

G=interp1(fn,G,f,'linear','extrap');
D=real(ifft(fft(xn).*abs(G)));
R=xn-D;

Ddn=D.*xref;
Rdn=R.*xref;

D_GATP = interp1(t2,Ddn,t1,'spline');
R_GATP = interp1(t2,Rdn,t1,'spline');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [xn,xr,xref,do,fi2,t1,t2]=inst_norm_JU(x,fs,speed,load)

dt = 1/fs;

x=x(:)';

% angular resampling
fi=cumsum(speed)*dt;
fi=fi-fi(1); % Phase starts from zero
[xr,do,fi2,t1,t2]=JU_resample_phase(x,fi,fs);

% envelope
xre = abs(xr+1i*hilbert(xr));

% filtered envelope
ds = (diff(speed)/dt)./speed(1:end-1);
dl = (diff(load)/dt)./load(1:end-1);
LU = max([ds dl]);

xref = dig_filter_LP_JU(xre,fs,LU);

%normalization
xn=xr./xref;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [X_res,do,fi2,t1,t2]=JU_resample_phase(x,ph,fs)

%OUTPUT
% X_res - resampled signal
% do - new sampling frequency (angular)

%INPUT
% x - original vibration signal
% ph - phase signal (in rads)
% fs - sampling frequency

ph=ph(:)';
x=x(:)';
ph(diff(ph) == 0) = [];

ph=ph-ph(1);
N_rot=ph(end)/(2*pi);



N=length(x);
do=N/N_rot;

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

fi2=linspace(ph(1),ph(end),N);
t2 = interp1(ph,t1,fi2,'spline'); %nowa oś czasu
X_res = interp1(t1,x,t2,'spline'); %nowy stgnał



function [stft,t,f,fs_n,x,step]=STFT_FILTERBANK_JU(x,fs,n_bins)

%Jacek Urbanek 2011
%
% The script computes Short-Time Fourier Transform array
% that can be understood as a downsampled filterbank.
% The filter cutoff frequency is slighly higher than desired in order to
% acheve integer amount of samples per time step for STFT algorythm.
% Signal is being zero padded in order to "fit in" integer numbers of time
% windows in STFT algorythm and not lose the information.
%
%INPUT:
% x - time signal
% fs - sampling frequency [Hz]
% upper_freq - filter cutoff frequency [Hz]
% n_bins - number of fft bins
%OUTPUT:
%
% stft - computed short time Fourier transform
% t - time vector
% f - frequency vector
% fs_n - new sampling frequency
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %for display:
%
%
```

```
% figure; imagesc(f,t,abs(stft));title('Spectrogram'); ylabel('Time[s]','fontsize',12);
xlabel('frequency[Hz]','fontsize',12);
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x=x(:)';
N=length(x);
T=N/fs;
win=n_bins;
step=n_bins/2;

n_steps=ceil((N-win)/step); %new samples number for calculated new sampling frequency

x(end+1:step*n_steps+win)=0; %zero padding
N_n=length(x); %new signal length
T_n=N_n/fs;

fs_n=n_steps/T_n ;%new sampling frequency - highier than desired for integer number of
samples in time step

%h=waitbar(0,'computing filterbank');


stft=zeros(n_steps,n_bins);


window=hanning(win);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i=1:n_steps
    fft_xi=fft((x(1+step*(i-1):step*(i-1)+win)).*window',n_bins);
    stft(i,:)=fft_xi(1:n_bins)/n_bins;

%waitbar(i/n_steps,h);
end
%close(h);

%stft(:,end/2:end)=0;



df=fs/n_bins;
f=linspace(df/2,fs-df/2,n_bins);
t=(1:n_steps)/fs_n;



function [filX]=dig_filter_LP_JU(x,fs,LU)

x=x(:)';
x(isnan(x)==1)=0;

N=length(x);
df=fs/N;

fft_x=fft(x);
```

```
fft_mask=zeros(1,N);
win=hanning(2*length(1:ceil(LU/df)));

fft_mask(1:ceil(LU/df))=win(end/2+1:end);
fft_mask(1)=0.5;

filX=real(2*ifft(fft_x.*fft_mask));
```