Automatic Structured Noise Detection

```
A fixed 'Deviation Threshold' has to be selected
beforehand using single sample learning.

If not, please run the single_sample_threshold_selection.m
present in the same folder as this one to select the
'Deviation Threshold'.

Using that fixed threshold we will perform
Automatic Structured Noise Detection

Our method result is directly dependent on this
'Deviation threshold', which is our only parameter.

We have four other hyperparameters, whose varaition
has limited effect on the final result.
These hyperparameter values are fixed and have been
selected empirically

Other functions required:

     pbu_NaN_removed=remove_nan(pbu)

     pbu_interp=interpolate_data(pbu_NaN_removed, sampling_rate)

     [xlv,ylv]=get_limit(t_NaN_removed,X_NaN_removed)

     ssa_rc=ssa_decomposition(X_interp,window)

     C_T=original_to_interp_mapping_single_point(t_NaN_removed(i),t_interp)

     [sp,ep]=interp_to_original_mapping_start_end_point...
         (t_interp(start_end_pair(1,ww)),t_interp(start_end_pair(2,ww)),t_NaN_removed)
```

Farhan Asif Chowdhury, 2018 aumyfarhan@gmail.com

Contents

- clear previous figure, data
- Figure display and save options
- All Hyperparameters
- Directory creation for raw input data and output result
- Select Well PBU CSV data folder
- Read pressure derivative
- Read delta pressure
- Parameter Selection (Deviation Threshold)
- Performing the Structured Noise Detection

- Loads derivative data
- Nan Removal
- Linear interpolation
- Loads Pressure Data
- Obtains plotting axis limit
- SSA Decomposition
- Single Point Deviation based Structured Noise Detecton
- Structured Noise Grouping
- Translating noise index from interpolated datapoints to original datapoints
- Windowed averaged deviation based Structured Noise Detection
- Translating noise index from interpolated datapoints to original datapoints for windowed average
- Majority Voting
- Majority voting after windowed average
- Random Noise Detection
- labeling Structured Noise and Random Noise on the original Data for single point
- labeling Structured Noise and Random Noise on the original Data for Majority Voting
- labeling Structured Noise and Random Noise on the original Data for Windowed averaging
- labeling Structured Noise and Random Noise on the original Data for Majority Voting after windowed average
- Save Figure
- Append all the pdf in a single pdf

clear previous figure, data

```
clear all;
close all;
clc;
```

Figure display and save options

```
make_figure_visible='off';     % 'on' if want to display figure in matlab while running, 'off'
 otherwise
ghostscript_available=false(1,1); % true(1,1) if ghostscript has been installed, false(1,1) ot
herwise
```

All Hyperparameters

```
dev_win=11;          %window for deviaton avergaing

count_win=5;         %window for Majority voting

window=25;           %SSA window; expected length of the
%structure we want to detect

sampling_rate=0.01; %uniform sampling rate
```

Directory creation for raw input data and output result

```
code_dir=pwd;                                % directory where all the code exists

home_dir=strcat(code_dir,'/..');             % home directory, include code, data, result fold
er

pdf_maker_dir=strcat(code_dir,'/export_fig');% path directory for export_fig toolbox
addpath(genpath(pdf_maker_dir))

cd (home_dir)

if ~(exist ('DATA','dir'))                    % checks if there is already a DATA folder
    mkdir DATA;                               % if not, creates one
end

if ~(exist ('RESULT','dir'))                  % checks if there is already a RESULT folder
    mkdir RESULT;                             % if not, creates one
end
```

Select Well PBU CSV data folder

```
data_path=strcat(home_dir,'/DATA');           % path to the Well PBU data folder
cur_data_path = uigetdir(data_path);

[~,cur_data_folder_name] = fileparts(cur_data_path);   % name of the selected Well PBU data fo
lder

cd(cur_data_path);

disp('Running...');
```

Read pressure derivative

```
RM_fileList = dir('*_RM.csv');

L=length(RM_fileList);

PBU=cell(1,L);

for k=1:L

    filenames=RM_fileList(k).name;

    val=csvread(filenames,1,0);
    PBU{1,k}=val;

end
```

Read delta pressure

```
log10dp_fileList = dir('*_log10dp.csv');
```

```
L=length(log10dp_fileList);

log10dp=cell(1,L);

for k=1:L

    filenames=log10dp_fileList(k).name;

    val=csvread(filenames,1,0);
    log10dp{1,k}=val;

end

cd (home_dir);
cd ('RESULT');

cur_data_result_folder_name=strcat(cur_data_folder_name,'_RESULT');

if ~(exist (cur_data_result_folder_name,'dir'))           % checks if there is already a Result folder for the selected well
    mkdir (cur_data_result_folder_name);                  % if not, creates one
end

% cd (cur_data_result_folder_name);
%
% pbu_file_name=strcat(cur_data_folder_name,'.mat');
% save(pbu_file_name,'PBU','log10dp');                  % saves pbu pressure and derivative data in a mat file

cd (code_dir);
```

Parameter Selection (Deviation Threshold)

Ask the user for a threshold using a prompt input window

```
prompt=('Enter a threshold value for Deviation');
dlg_title=('Deviation Threshold Value');
num_lines=1;


defaultans={'0.08'};   %Default Deviation Threshold Value

%  Threshold value stored in variable 'dev_thresh_val'

dev_thresh_val=inputdlg(prompt,dlg_title,num_lines,defaultans);
dev_thresh=str2double(dev_thresh_val{1,1});


%  creating a cell variable 'file_name_cell'
% to store the individual PBU figure name
% in order to append them in future in a single file
file_name_cell=cell(1,length(PBU));
```

Performing the Structured Noise Detection

```
Inside a for loop, performing Structured Noise
Detection for all the PBU of the Well
```

```
for pbu_no=1:length(PBU)
```

Loads derivative data

loading both Time axis value and Derivative value in a Matrix. Size of matrix *pbu* is 2 by 'PBU-Time Series length'. First row contains Time axis value Second row contains Derivative value

```
    pbu=PBU{1,pbu_no};
```

Nan Removal

Removes the data points where derivative is -999 size of input and output matrix is 2 by 'PBU-Time Series length'. First row contains Time axis value Second row contains Derivative value

```
    pbu_NaN_removed=remove_nan(pbu);

    t_NaN_removed=pbu_NaN_removed(:,1);    % time axis values of NaN removed data

    X_NaN_removed=pbu_NaN_removed(:,2);    % derivative values of NaN removed data
```

Linear interpolation

Performs linear interpolation on the NaN removed Data to make the data uniformly sampled using given sampling rate size of input and output matrix is 2 by 'PBU-Time Series length'. First row contains Time axis value Second row contains Derivative value

```
    pbu_interp=interpolate_data(pbu_NaN_removed, sampling_rate);

    t_interp=pbu_interp(:,1);               % time axis values of interpolated data

    X_interp=pbu_interp(:,2);               % derivative values of interpolated data
```

Loads Pressure Data

loading both Time axis value and pressure value in a Matrix. size of the matrix *dp_val* is 2 by 'PBU-Time Series length'. First row contains Time axis value Second row contains pressure value

```
    dp_val=log10dp{pbu_no};

    dp_time=dp_val(:,1);                    % time axis value of pressur data

    dp_derv=dp_val(:,2);                    % pressure value
```

Obtains plotting axis limit

obtains the minimum, maximum of time axis value and derivative value for plot axis limit

```
    [xlv,ylv]=get_limit(t_NaN_removed,X_NaN_removed);
```

SSA Decomposition

Performs SSA decomposition on the NaN removed, interpolated derivative and returns the first five Reconstructed Components(RC) in a matrix *ssa_rc* of size 5 by 'PBU Time Series length' . First row of ssa_rc contains first RC, second row contains second RC and so on.

```
    ssa_rc=ssa_decomposition(X_interp,window);   % SSA decomposition to obtain first five RC

    S1=ssa_rc(1,:);                            % loading each RC into a separate row vector
    S2=ssa_rc(2,:);
    S3=ssa_rc(3,:);
    S4=ssa_rc(4,:);
    S5=ssa_rc(5,:);

    SA=(S1+S2+S3+S4+S5);                        % sum of RC1 through RC5

    S=abs(S2+S3+S4+S5);                         % absolute difference between RC1 and sum of R
C1 through RC5
    % diff of SA and S1 = SA-S1 = S1+S2+S3+S4+S5-S1 = S2+S3+S4+S5
```

Single Point Deviation based Structured Noise Detecton

```
    single_point_structured_noise_ind_temp=find((S-dev_thresh)>0); % checks against 'Deviation
 Threshold" to identify structured noise

    single_point_structured_noise_vector_temp=zeros(1,length(S));

    single_point_structured_noise_vector_temp(single_point_structured_noise_ind_temp)=1;
```

Structured Noise Grouping

groups together the stuctured noise indexes by giving all connected noise indexes the same label then finds the start and end index of a grouped connected noise segment and maps these indexes to original data points for labeling over original data

```
    label_no=0;
    label=zeros(1,length(single_point_structured_noise_vector_temp));
    prev_one=0;

    for w=1:length(single_point_structured_noise_vector_temp)    % grouping and giving same la
bel

        if (single_point_structured_noise_vector_temp(w)==1)
            if (prev_one==0)
                label_no=label_no+1;
                label(w)=label_no;
                prev_one=1;
            else
                label(w)=label_no;
            end

        else
            prev_one=0;
        end

    end

    start_end_pair=zeros(2,label_no);               % findind start/end indexes
    for ww=1:label_no
        cur_ind=find(label==ww);
        si=min(cur_ind);
        ei=max(cur_ind);
        start_end_pair(1,ww)=si;
        start_end_pair(2,ww)=ei;
    end
```

Translating noise index from interpolated datapoints to original datapoints

```
single_point_structured_noise_vector=zeros(1,length(X_NaN_removed));

for ww=1:label_no

    [sp,ep]=interp_to_original_mapping_start_end_point...
        (t_interp(start_end_pair(1,ww)),t_interp(start_end_pair(2,ww)),t_NaN_removed);

    single_point_structured_noise_vector(sp:ep)=1;

end

single_point_structured_noise_ind=find(single_point_structured_noise_vector==1);
```

Windowed averaged deviation based Structured Noise Detection

```
% an empty vector to store windowed averaged SSA deviation value
SW=zeros(1,length(S1));

% runs a for loop and calculates windowed average SSA deviation
% centering each point
for i=ceil(dev_win/2):length(SW)-floor(dev_win/2)
    SW(i)=sum(S(i-floor(dev_win/2):i+floor(dev_win/2)))/dev_win;
end

win_avg_structured_noise_ind_temp=find((SW-dev_thresh)>0);

win_avg_structured_noise_vector_temp=zeros(1,length(S));

win_avg_structured_noise_vector_temp(win_avg_structured_noise_ind_temp)=1;

label_no=0;
label=zeros(1,length(win_avg_structured_noise_vector_temp));
```

```
    prev_one=0;

    for w=1:length(win_avg_structured_noise_vector_temp)

        if (win_avg_structured_noise_vector_temp(w)==1)
            if (prev_one==0)
                label_no=label_no+1;
                label(w)=label_no;
                prev_one=1;
            else

                label(w)=label_no;

            end

        else
            prev_one=0;
        end

    end

    start_end_pair=zeros(2,label_no);
    for ww=1:label_no
        cur_ind=find(label==ww);
        si=min(cur_ind);
        ei=max(cur_ind);
        start_end_pair(1,ww)=si;
        start_end_pair(2,ww)=ei;
    end
```

Translating noise index from interpolated datapoints to original datapoints for windowed average

```
    win_avg_structured_noise_vector=zeros(1,length(X_NaN_removed));

    for ww=1:label_no

        [sp,ep]=interp_to_original_mapping_start_end_point...
            (t_interp(start_end_pair(1,ww)),t_interp(start_end_pair(2,ww)),t_NaN_removed);

        win_avg_structured_noise_vector(sp:ep)=1;

    end

    win_avg_structured_noise_ind=find(win_avg_structured_noise_vector==1);
```

Majority Voting

```
    major_voting_structured_noise_vector=zeros(1,length(single_point_structured_noise_vector))
;

    for i=count_win+1:length(major_voting_structured_noise_vector)-count_win
        if (sum(single_point_structured_noise_vector(i-count_win:i+count_win))...
                -single_point_structured_noise_vector(i))>=count_win
            major_voting_structured_noise_vector(i)=1;
        else
            major_voting_structured_noise_vector(i)=0;
        end
    end

    major_voting_structured_noise_ind=find(major_voting_structured_noise_vector==1);
```

Majority voting after windowed average

```
dil_ero_str=ones(1,2*count_win+1);

major_voting_win_avg_structured_noise_dil=imdilate(win_avg_structured_noise_vector...
    ,dil_ero_str);

major_voting_win_avg_structured_noise_ero=imerode(...
    major_voting_win_avg_structured_noise_dil,dil_ero_str);

major_voting_win_avg_structured_noise_ind=find(...
    major_voting_win_avg_structured_noise_ero==1);
```

Random Noise Detection

First, relates/matches each original time axis value with nearest/closest (based on time axis value) interpolated time axis value.

later we use this mapping to calculate the difference betweeen 'derivative value of original datapoints' and 'sum of RC1 through RC5' which are interpolated. As we want to take difference of two vectors whose no of sample(data points) are different,we need to perform this mapping to obtain two vectors of same no of data points.

```
% An empty vector to store the index of the mapping of
% original time axis value into
% interpolated time axis value.
C_T=zeros(1,length(t_NaN_removed));

% runs a for loop through the orignal time axis value
% and maps them into interpolated time axis value
for i=1:length(t_NaN_removed)
    C_T(i)=original_to_interp_mapping_single_point(t_NaN_removed(i),t_interp);
end

% using the time axis mapping, obtains corresponding
% 'sum of RC1 through RC5' value of SSA decomposed siganl derivative.
SA_mapped=SA(C_T);


% calculates the difference betweeen original derivative value
% and 'sum of RC1 through RC5' of SSA decomposed derivative;
% and checks with 'Deviation threshold' value for random noise detection
% and stores the random noise indexes in random_noise_ind_temp.
random_noise_ind_temp=find(abs(SA_mapped-X_NaN_removed')>dev_thresh);


% performs dilation followed by erosion for gap filling/ majority voting

% creates temporary array to store random noise indexes to perform
```

```
    % gap filing/majority voting on the currently detected random noise
    rand_val=zeros(1,length(X_NaN_removed));
    rand_val(random_noise_ind_temp)=1;

    dil_ero_str_rand=ones(1,count_win+1);           % defining 1D dilation/erosion structure

    rand_dil=imdilate(rand_val,dil_ero_str_rand);   % performing dilation

    rand_ero=imerode(rand_dil,dil_ero_str_rand);    % performing erosion

    random_noise_ind=find(rand_ero==1);        % random noise indexes after gap filing/ majori
ty voting
```

labeling Structured Noise and Random Noise on the original Data for single point

```
    figure('units','normalized','outerposition',[0 0 1 1],'visible',make_figure_visible);

    subplot(2,2,1),plot (dp_time,dp_derv,'.c'); % plot derivative value

    xlim(xlv),ylim(ylv); grid on,GridLineStyle=':';
    set(gca,'XTick', xlv(1)+.2:xlv(2)-.2,'YTick', ylv(1)+.2:ylv(2)-.2);
    xlabel('Log(delta time)','FontSize',8),...
        ylabel('Log(delta pressure) & Log(derivative)','FontSize',8);

    hold on;

    plot(t_interp,S1,'-g');                    % plot first RC component

    plot(t_interp,SA,'-k');                    % plot sum of RC1 through RC5

    plot(t_NaN_removed,X_NaN_removed,'.b');    % plot original Nan removed derivative

    if (isempty(random_noise_ind))
        plot(NaN,NaN,'.m');
    else
        plot(t_NaN_removed(random_noise_ind),X_NaN_removed(random_noise_ind),'.m');  % label r
andom noise over original derivative
    end

    if (isempty(single_point_structured_noise_ind))
        plot(NaN,NaN,'.r');
    else
        plot(t_NaN_removed(single_point_structured_noise_ind),...
            X_NaN_removed(single_point_structured_noise_ind),'.r');   % label structured noise
over original derivative
    end

    % title for the figure
    title_str=strcat('Single Point,PBU: ',num2str(pbu_no),' ;Threshold: ',num2str(dev_thresh))
;

    title(title_str,'FontSize',8);

    % legend for the figure
    legend ({'Delta Pressure','RC1','RC1-5','Signal Derivative',...
        'Random Noise','Structured Noise'},'Location','southwest','FontSize',5);
    legend('boxoff');

    hold off;
```

labeling Structured Noise and Random Noise on the original Data for Majority Voting

```
    subplot(2,2,2),plot (dp_time,dp_derv,'.c'); % plot derivative value

    xlim(xlv),ylim(ylv); grid on,GridLineStyle=':';
    set(gca,'XTick', xlv(1)+.2:xlv(2)-.2,'YTick', ylv(1)+.2:ylv(2)-.2);
    xlabel('Log(delta time)','FontSize',8),...
        ylabel('Log(delta pressure) & Log(derivative)','FontSize',8);

    hold on;

    plot(t_interp,S1,'-g');                       % plot first RC component

    plot(t_interp,SA,'-k');                       % plot sum of RC1 through RC5

    plot(t_NaN_removed,X_NaN_removed,'.b');     % plot original Nan removed derivative

    if (isempty(random_noise_ind))
        plot(NaN,NaN,'.m');
    else
        plot(t_NaN_removed(random_noise_ind),X_NaN_removed(random_noise_ind),'.m');  % label r
andom noise over original derivative
    end

    if (isempty(major_voting_structured_noise_ind))
        plot(NaN,NaN,'.r');
    else
        plot(t_NaN_removed(major_voting_structured_noise_ind),...
            X_NaN_removed(major_voting_structured_noise_ind),'.r');  % label structured noise
over original derivative
    end
    title_str=strcat('Majority Voting, PBU: ',num2str(pbu_no),' ;Threshold: ',num2str(dev_thre
sh));

    title(title_str,'FontSize',8);

    legend ({'Delta Pressure','RC1','RC1-5','Signal Derivative',...
        'Random Noise','Structured Noise'},'Location','southwest','FontSize',5);
    legend('boxoff');

    hold off;
```

labeling Structured Noise and Random Noise on the original Data for Windowed averaging

```
    subplot(2,2,3),plot (dp_time,dp_derv,'.c'); % plot derivative value

    xlim(xlv),ylim(ylv); grid on,GridLineStyle=':';
    set(gca,'XTick', xlv(1)+.2:xlv(2)-.2,'YTick', ylv(1)+.2:ylv(2)-.2);
    xlabel('Log(delta time)','FontSize',8),...
        ylabel('Log(delta pressure) & Log(derivative)','FontSize',8);
    hold on;

    plot(t_interp,S1,'-g');                    % plot first RC component

    plot(t_interp,SA,'-k');                    % plot sum of RC1 through RC5

    plot(t_NaN_removed,X_NaN_removed,'.b');    % plot original Nan removed derivative

    if (isempty(random_noise_ind))
        plot(NaN,NaN,'.m');
    else
        plot(t_NaN_removed(random_noise_ind),X_NaN_removed(random_noise_ind),'.m');  % label r
andom noise over original derivative
    end
    if (isempty(win_avg_structured_noise_ind))
        plot(NaN,NaN,'.r');
    else
        plot(t_NaN_removed(win_avg_structured_noise_ind),...
            X_NaN_removed(win_avg_structured_noise_ind),'.r');  % label structured noise over
original derivative
    end
    title_str=strcat('Windowed Average, PBU: ',num2str(pbu_no),...
        ' ;Threshold: ',num2str(dev_thresh));

    title(title_str,'FontSize',8);

    legend ({'Delta Pressure','RC1','RC1-5','Signal Derivative'...
        ,'Random Noise','Structured Noise'},'Location','southwest','FontSize',5);
    legend('boxoff');

    hold off;
```

labeling Structured Noise and Random Noise on the original Data for Majority Voting after windowed average

```
    subplot(2,2,4),plot (dp_time,dp_derv,'.c'); % plot derivative value

    xlim(xlv),ylim(ylv); grid on,GridLineStyle=':';
    set(gca,'XTick', xlv(1)+.2:xlv(2)-.2,'YTick', ylv(1)+.2:ylv(2)-.2);
    xlabel('Log(delta time)','FontSize',8),...
        ylabel('Log(delta pressure) & Log(derivative)','FontSize',8);

    hold on;

    plot(t_interp,S1,'-g');                    % plot first RC component

    plot(t_interp,SA,'-k');                    % plot sum of RC1 through RC5

    plot(t_NaN_removed,X_NaN_removed,'.b');    % plot original Nan removed derivative

    if (isempty(random_noise_ind))
        plot(NaN,NaN,'.m');
    else
        plot(t_NaN_removed(random_noise_ind),X_NaN_removed(random_noise_ind),'.m');  % label r
```

```
andom noise over original derivative
    end

    if (isempty(major_voting_win_avg_structured_noise_ind))
        plot(NaN,NaN,'.r');
    else
        plot(t_NaN_removed(major_voting_win_avg_structured_noise_ind),...
            X_NaN_removed(major_voting_win_avg_structured_noise_ind),'.r');  % label structure
d noise over original derivative
    end

    title_str=strcat('Majority Voting after Windowed Average, PBU: ',num2str(pbu_no)...
        ,' ;Threshold: ',num2str(dev_thresh));

    title(title_str,'FontSize',8);

    legend ({'Delta Pressure','RC1','RC1-5','Signal Derivative'...
        ,'Random Noise','Structured Noise'},'Location','southwest','FontSize',5);
    legend('boxoff');
    hold off;
```

Save Figure

```
    %name_str=strcat(data_folder_name,'Noise_marked_',num2str(pbu_no),'.jpg');


    fig_save_name=strcat(home_dir,'/RESULT/',cur_data_result_folder_name,...
        '/',cur_data_folder_name,'_AUTO_NOISE_T_',num2str(dev_thresh),'_PBU_',num2str(pbu_no),
'.pdf');

    %saveas(gcf,fig_save_name);

    set(gcf, 'Color', 'w');
    %export_fig (fig_save_name,'-nocrop');

    print(gcf,'-dpdf',fig_save_name,'-bestfit');

    file_name_cell{1,pbu_no}=fig_save_name;
```

```
end
```

Append all the pdf in a single pdf

Has dependency on the availability of Ghostscript

```
if (ghostscript_available)

    final_output_name=strcat(home_dir,'/RESULT/',cur_data_result_folder_name,...
        '/',cur_data_folder_name,'_AUTO_NOISE_T_',num2str(dev_thresh),'_ALL.pdf');

    if exist(final_output_name,'file')
        delete (final_output_name);
    end

    append_pdfs(final_output_name, file_name_cell{:});

    for w=1:length(file_name_cell)
        delete (file_name_cell{w})
    end

end

clc;
disp('Finished');
```

Published with MATLAB® R2018a