Electronic Supplementary Information

Spectral features extraction based on continuous wavelet transform and image segmentation for peak detection

Guofeng Yang,[a] Jiacai Dai,*[a] Xiangjun Liu, [a,b] Meng Chen, [a] and Xiaolong Wu [a]

[a] *School of Geoscience and Technology, Southwest Petroleum University, Chengdu 610500, Sichuan Province, China*

[b] *State Key Laboratory of Oil and Gas Reservoir Geology and Exploitation, Southwest Petroleum University, Chengdu 610500, Sichuan Province, China*

1. The demo of CWT-IS

This section provides a demo of the CWT-IS peak detection algorithm. CWT-IS is designed and implemented efficiently in MATLAB. The user can save the spectra data as “Spectrum.mat” and run this program. In appendix Ⅰ, we also provide data to test the algorithm.

```
clear
clc
close all
load Spectrum;
% CWT-IS peak detection
peak_list = CWTIS_peakfinder(Spectrum,1:0.15:7,5,0);
figure
if peak_list ~= 0
    plot(Spectrum(:,1), Spectrum (:,2));
    for i=1:size(peak_list,1)
        hold on
        plot(Spectrum (floor(peak_list(i)),1), Spectrum
(floor(peak_list(i)),2),'or');
    end
```

```
end
xlabel('Channel')
ylabel('Intensity')
title('CWT-IS peak detection')
```

2. The source codes of CWT-IS

This section provides the source codes of the CWT-IS peak detection algorithm. MATLAB software needs to be installed on the computer and the version used by the author is 2016a. The user can save the following functions as ".m file" and test the algorithm performance. The meaning of the main parameters can be seen in the comments of the functions.

```
function peak_list =
CWTIS_peakfinder(Spectrum,Scale,threshold,Compress_type,Clust_num)
% CWT-IS peak detection function(Version 1.0)
% A command-line peak detection program for spectral signals,
% written as Matlab functions in a package.
% Uses continuous wavelet transform and image segmentation to extract
% spectral peak features in the wavelet domain and determine the peak
% positons of the input spectrum.
% Input arguments:
% Spectrum -- input spectral signal matrix, which has channel values
in column 1
% and intensity values in column 2 (e.g.[x y])
% Scale -- the vector of continuous wavelet transform scales
% threshold -- the threshold of the potential peak region
% Compress_type -- the way of wavelet coefficients map to grayscale
values.
% =0 is linear compression and =1 is logistic compression
% Clust_num -- the number of categories used in fuzzy Otsu method,
the default is 2.
% Output arguments:
% peak_list -- the vector of the determined peak position indices
% Copyright (c) 2019 Guofeng Yang, Southwest Petroleum University,
Chengdu, China
if nargin==5
    disp('Processing begin...')
elseif nargin==4
    Clust_num = 2;
    disp('Processing begin...')
else
```

```
    error('Too many or too few input arguments');
end
signal = Spectrum(:,2);
signal2 = [ones(1024,1).*signal(1);signal;ones(1024,1).*signal(end)];
coefs_mexh = cwt(signal2,Scale, 'mexh');
coefs_mexh  = coefs_mexh(:,1024+1:size(signal2,1)-1024);
A = Compress(coefs_mexh, Compress_type);
H = zeros(1,256);
index = 0:255;
for i=1:size(A,1)
    for j=1:size(A,2)
        V = index==A(i,j);
        H(V==1) = H(V==1)+1;
    end
end
iMax = max(A(:));
iMin = min(A(:));
T = iMin:iMax;
[iRow, iCol] = size(A);
imagSize = iRow*iCol;
[~,Thmin] = max(H);
n = 1;
ThO = Otsu(A, Thmin);
for k = Thmin:ThO
    iBg = 0;
    iFg = 0;
    BgSum = 0;
    FgSum = 0;
    for i=1:iRow
        for j=1:iCol
            if A(i,j)>k
                iFg = iFg + 1;
                FgSum = FgSum + A(i,j);
            else
                iBg = iBg + 1;
                BgSum = BgSum + A(i,j);
            end
        end
    end
    w0 = iFg/imagSize;
    w1 = iBg/imagSize;
    u0 = FgSum/iFg;
    u1 = BgSum/iBg;
    ut = sum(sum(A))/imagSize;
```

```
    Tval(n) = w0*(u0-ut)^2 + w1*(u1-ut)^2;
    n = n + 1;
end
MapTh = mapminmax(Tval, 0, 1);
[~,U] = FCMCluster(MapTh' ,Clust_num);
for i=1:size(U,1)
    if U(i,1) > 0.5
        break;
    end
end
delt = find(U(i,:)>0.5, 1, 'last' );
Th = Thmin + delt;
D = A > Th;
bw=bwlabel(D);
for i=1:max(max(bw))
    if length(find(bw==i))>threshold
        bw(bw==i)=-1;
    end
end
bw=bwlabel(bw==-1);
figure
imagesc(Spectrum(:,1), Scale, bw);
out=zeros(size(coefs_mexh));
for i=1:max(max(bw))
    out=out+local_maxima(coefs_mexh,bw,i);
end
out=out.*bw;
hang = [];
lie = [];
index_all = [];
vally = [];
for i=1:max(max(bw))
    [hang_p, lie_p, index_p] = local_minima(coefs_mexh,bw,i);
    hang = [hang,hang_p];
    lie = [lie,lie_p];
    index_all = [index_all, index_p];
    if isempty(lie_p)~=1
        table = tabulate(lie_p);
        [~,idx] = max(table(:,2));
        vally = [vally, idx];
    end
end
for i=1:max(max(out))
    out_s=out==i;
```

```
    out_sum = sum(out_s);
    temp = find(out_sum==max(out_sum,[],2));
    if size(temp,2) ~= 1
        temp = mean(temp);
    end
    Ridge(i) = temp;
end
Ridge(index_all) = [];
cw = zeros(size(bw));
count = 1;
for k=1:length(index_all)
    for i=1:size(bw,1)
        for j=1:size(bw,2)
            if bw(i,j) == index_all(k) && j<vally(k)
                cw(i,j) = count;
            end
            if bw(i,j) == index_all(k) && j>vally(k)
                cw(i,j) = count + 1;
            end
        end
    end
    count = count + 2;
end
out=zeros(size(coefs_mexh));
for i=1:max(max(cw))
    out=out+local_maxima(coefs_mexh,cw,i);
end
out=out.*cw;
for i=1:max(max(out))
    out_s=out==i;
    out_sum = sum(out_s);
    temp = find(out_sum==max(out_sum,[],2));
    if size(temp,2) ~= 1
        temp = mean(temp);
    end
    Ridge = [Ridge,temp];
end

Ridge =sort(Ridge);
Ridge =Ridge';
peak_list = Ridge;
disp('processing complete');
end
```

```
function ThreshValue = Otsu(Imag,Thmin)
iMax = max(Imag(:));
iMin = min(Imag(:));
T = iMin:iMax;
Tval = zeros(size(T));
[iRow, iCol] = size(Imag);
imagSize = iRow*iCol;
for i = Thmin : length(T)
    TK = T(i);
    iFg = 0;
    iBg = 0;
    FgSum = 0;
    BgSum = 0;
    for j = 1 : iRow
        for k = 1 : iCol
            temp = Imag(j, k);
            if temp > TK
                iFg = iFg + 1;
                FgSum = FgSum + temp;
            else
                iBg = iBg + 1;
                BgSum = BgSum + temp;
            end
        end
    end
    w0 = iFg/imagSize;
    w1 = iBg/imagSize;
    u0 = FgSum/iFg;
    u1 = BgSum/iBg;
    ut = sum(sum(Imag))/imagSize;
    Tval(i) = w0*(u0-ut)^2 + w1*(u1-ut)^2;
end
[val, flag] = max(Tval);
ThreshValue = T(flag);
end


function [center,U] = FCMCluster(data,n)
data_n=size(data,1);
default_options=[2;100;1e-5];
options=default_options;
expo=options(1);
max_iter=options(2);
```

```
min_impro=options(3);
obj_fun=zeros(max_iter,1);
U=initfcm(n,data_n);
for i=1:max_iter
    [U,center,obj_fun(i)]=stepfcm(data,U,n,expo);
    if i>1
        if abs(obj_fun(i)-obj_fun(i-1))<min_impro
            break;
        end
    end
end

end

function U= initfcm(n,data_n)
U=rand(n,data_n);
col_sum=sum(U);
U=U./col_sum(ones(n,1),:);

end

function [U_new,center,obj_fun]=stepfcm(data,U,n,expo)
mf=U.^expo;
center=mf*data./((ones(size(data,2),1)*sum(mf'))');
dist=distfcm(center,data);
obj_fun=sum(sum((dist.^2).*mf));
tmp=dist.^(-2/(expo-1));
U_new=tmp./(ones(n,1)*sum(tmp));

end

function out=distfcm(center,data)
out=zeros(size(center,1),size(data,1));
for k=1:size(center,1)
    out(k,:)=sqrt(sum(((data-
ones(size(data,1),1)*center(k,:)).^2)',1));
end

end



function out=local_maxima(abs_coef,bw,ind)
% this function is used to search the local maxima value in a
% matrix
```

```
out=zeros(size(abs_coef));
sub=(bw==ind).*abs_coef;
MAX=max(sub,[],2);
for j=1:size(abs_coef,1)
    out(j,:)=abs_coef(j,:)==MAX(j);
end


function [hang, lie, index]=local_minima(abs_coef,bw,ind)
% this function is used to search the local minima value in a
% matrix
index = [];
out=zeros(size(abs_coef));
hang = [];
lie = [];
sub=(bw==ind).*abs_coef;
count = 1;
for i=1:size(abs_coef,1)
    [x,y] = find(bw(i,:)==ind);
    if isempty(x)~= 1 && isempty(y)~=1
        left(count) = min(y);
        right(count) = max(y);
        row(count) = i;
        count = count+1;
    end
end
count = 1;
for i=1:length(left)
    MIN(i) = min(sub(row(i),left(i):right(i)));
end
submax = max(max(sub))+1;
for i=1:length(left)
    sub(row(i),1:left(i)-1) = submax;
    sub(row(i),right(i)+1:end)=submax;
end
for i=1:length(left)
    [~,b] = find(sub(row(i),:)==MIN(i));
    if length(b) ~=1
        b = b( round(length(b)/2) );
    end
    if (b>left(i) && b<right(i))
        out(row(i),b) = 1;
        hang(count) = row(i);
        lie(count) = b;
```

```
        count = count + 1;
    end
end
wan = diff(lie);
[x,~] = find(wan==0);
if length(hang)<1/4*length(row)
    hang = [];
    lie = [];
elseif length(x)<1/5*length(row)
    hang = [];
    lie = [];
else
    index = ind;
end


function A = Compress( coefs, type )
if type == 0
    max_coef=max(max(coefs));
    min_coef=min(min(coefs));
    ext=max_coef-min_coef;
    A = 255*(coefs - min_coef) ./ ext;
    A = floor(A);
else
    m = mean(mean(coefs));
    s = std2(coefs);
    for i=1:size(coefs,1)
        for j=1:size(coefs,2)
            A(i,j) = floor(255./(1+exp(2*(-coefs(i,j)-m)/s)));
        end
    end

end
```

APPENDIX Ⅰ Test data

Channel	Intensity
1	130.858819771978
2	130.771553151472
3	132.124862946034
4	130.523782285338
5	128.759807779039
6	131.929920361009

7	131.696426750537
8	130.295500182207
9	131.029502732314
10	131.563585714052
11	131.335218541235
12	131.670804728573
13	131.108656493832
14	131.251802070167
15	131.436378901937
16	130.816424219680
17	132.668263397549
18	131.616736131379
19	132.243524142024
20	131.095332617747
21	131.008442918770
22	131.892846327090
23	131.872744176543
24	133.159191180302
25	132.844724677828
26	132.989079452717
27	131.653129372162
28	132.361695764747
29	133.720405964550
30	133.823471201295
31	135.821489104895
32	136.717448064413
33	136.220579031871
34	138.621581421399
35	140.467943661490
36	141.215828366666
37	144.009155262937
38	145.405039138817
39	146.597903491331
40	148.948583673056
41	150.089187672639
42	151.074494801793
43	152.885026144780
44	153.493180904569
45	153.697912772950
46	155.613018932044
47	154.321616583071
48	156.117358278187
49	153.455030672199
50	154.025838926923

51	155.009478252973
52	156.540038557372
53	155.793207094446
54	156.790655953475
55	160.663143211146
56	162.270347271221
57	167.035347526906
58	171.867525418799
59	177.488105873082
60	182.079320999186
61	185.236388491611
62	189.729226808477
63	191.599776308874
64	194.170593037540
65	195.732118910118
66	193.597789217214
67	191.250317892498
68	187.971975355106
69	184.877817525953
70	180.597668564849
71	176.007445898991
72	168.951520420018
73	163.707731340285
74	158.014858210926
75	155.187312719708
76	149.752900883341
77	145.945203769899
78	145.190771235618
79	141.802738739418
80	139.995096046572
81	137.397498510244
82	137.802620825818
83	137.571355459590
84	136.417510992664
85	137.691715921144
86	138.599937042643
87	140.904734312558
88	141.781034312591
89	146.136234501132
90	149.950978159234
91	152.086693233276
92	156.599303249389
93	161.625502150498
94	166.947656166380

95	171.086119626894
96	175.325957620422
97	178.348905506671
98	181.414207040244
99	183.889156640063
100	185.346354036360
101	183.813038327208
102	180.554262682737
103	178.960922125286
104	174.773243800526
105	170.270538638639
106	165.564457115658
107	160.226496236517
108	156.011404822077
109	151.309346269593
110	147.773269849877
111	144.819136139167
112	142.078016239830
113	139.512837523862
114	137.794990433843
115	136.594684537134
116	135.931137121425
117	135.861123435164
118	134.812811204169
119	133.722784412316
120	134.176684234341
121	133.329902318320
122	132.012789086419
123	131.635447782630
124	132.660682602653
125	135.418281737983
126	138.646629747970
127	144.649642353026
128	153.115434410910
129	159.536220171268
130	162.358691034932
131	160.440429587459
132	153.487347459164
133	146.842563475937
134	138.939912390137
135	135.096483242531
136	132.896222364740
137	132.555383783923
138	132.455655702705

139	131.017102008318
140	132.186107284685
141	132.984328525707
142	131.943033440649
143	132.837862032311
144	133.314103599989
145	133.506076727805
146	133.444919898510
147	131.262397234206
148	132.946539697137
149	134.541946222547
150	134.668412728413
151	135.011545564758
152	136.147876495664
153	137.639163897580
154	137.948061071467
155	138.463922275144
156	140.325475111053
157	141.845862783539
158	144.399212786095
159	145.784566816562
160	147.737206942210
161	149.294803794398
162	150.752165359954
163	152.895069055200
164	154.710416560439
165	154.943798479042
166	157.514767937658
167	159.161744914728
168	159.079823914753
169	160.845387546317
170	160.108444639935
171	159.043292883429
172	158.629074565539
173	157.576115177221
174	156.733845997815
175	155.628695967183
176	153.060862365437
177	151.767359570713
178	150.802079784487
179	148.086454321014
180	145.945372858321
181	142.630037507022
182	140.953827300954

183	139.824253195262
184	137.993210015916
185	135.454465769972
186	134.501707209255
187	135.808681146394
188	134.004212508686
189	136.269094945099
190	133.208103747824
191	136.099426714082
192	136.838598174193
193	137.612984143908
194	140.113456984804
195	140.802733078751
196	143.714166944557
197	144.466984311002
198	144.153300977809
199	144.129045110921
200	145.923567341558
201	146.597577476321
202	144.824351453704
203	143.254569533899
204	141.927405410689
205	138.185991116472
206	138.251907660054
207	134.216770092529
208	132.894008054872
209	130.692110877298
210	129.455118611116
211	126.384065656848
212	128.010023522805
213	126.304438976078
214	126.746883087349
215	125.702218387281
216	124.131825196758
217	123.017546168729
218	122.089886386210
219	123.252161384294
220	123.201309705079
221	123.668789382190
222	121.888687402892
223	123.108887250090
224	122.356883261125
225	121.508701984384
226	122.575009107547

227	121.995723941170
228	121.421789135601
229	122.937354959602
230	121.747443689758
231	120.995202722761
232	121.127574851771
233	120.796240238436
234	120.685275787938
235	120.727873425379
236	119.699585757561
237	119.360980921896
238	120.102119140028
239	121.135473349212
240	119.194542537854
241	118.441220831999
242	118.870763593844
243	118.483388151043
244	118.416524382669
245	118.865414387193
246	118.186396087088
247	117.619117374674
248	117.542586471916
249	116.641376281806
250	116.812383067022
251	116.168999062019
252	116.684811279710
253	116.026521522623
254	116.124673127391
255	116.243676363396
256	115.318047596960