Appendix

Research Project

Development of an automated microseismic event detection method and analysis of a microseismic dataset from hydraulic fracture stimulation to characterise natural and induced fracture networks within a tight reservoir

Marc Gerona Sendino

Appendix A: Code in Matlab of the automated microseismic event detection method

```
function automated_microseismic_event_detection_method(xcorr,plot)

% Continuous data folders

pname{1} = 'E:\Research project Bristol/Creelman/Creelman/ContinuousData/Stage1/Cont';
pname{2} = 'E:\Research project Bristol/Creelman/Creelman/ContinuousData/Stage2/Cont';

% Open text file to write the events detected

fid = fopen('eventsnew.txt', 'wt');

for id = 1:2

    % Continuous data folder and number of files

    d = dir([pname{id} '/*.dat']);
    L = length(d);

    for k = 1:L

        % Read the file

        fname = [pname{id} filesep d(k).name];
        [FDStrings, nTraces, TDStrings, TraceData] = seg2_read(fname);

        % Parameters and cells to start

        dt = 0.00025; % Sample interval time
        corner = [75 350]; % Bandpass filter limits
        TraceDataSumcell = cell(1,12); % Cell to store the sum of the three traces of every geophone
        RRcell = cell(1,12); % Cell to store STA/LTA ratio functions
        Eventscell = cell(1,24); % Cell to store start and end of events
```

```
count1 = 1;
count2 = 1;

% Loop to detect events in every geophone

for n=1:3:nTraces

    % Filter the three traces of the geophone

    [H1,H2,V]=filter(TraceData{n},TraceData{n+1},TraceData{n+2},corner,dt);

    % Store the sum of the three traces filtered

    TraceDataSumcell{1,count1} = H1+H2+V;

    % Calculating STA/LTA ratio for every trace and multiply them together

    Ndat=length(H1); % Number of points of the trace
    S=200; % Number of samples of the short time average
    L=2500; % Number of samples of the long time average

    [RR1]=recursive_STA_LTA(H1,Ndat,S,L);
    [RR2]=recursive_STA_LTA(H2,Ndat,S,L);
    [RR3]=recursive_STA_LTA(V,Ndat,S,L);
    RR=RR1.*RR2.*RR3;

    % Store the STA/LTA ratio function for the geophone

    RRcell{1,count1} = RR;

    % Apply trigger threshold to the STA/LTA ratio functions

    [evtstrtid,evtendid]=trigger_threshold(RR,Ndat,dt,xcorr);

    % Store the start and the end of the events picked
```

```
    Eventscell{1,count2} = evtstrtid;
    Eventscell{1,count2+1} = evtendid;

    % Move to the next geophone

    count1 = count1+1;
    count2 = count2+2;

end

% Condition to apply cross-correlation to the events

if xcorr==1

  % New Eventscell cell with false detections reduced applying cross-correlation

  [Eventscell]=cross_correlation(RRcell,Eventscell,dt);

end

% Unifies all the events picked trough the geophones giving a unique start time for each event

[Eventstart]=detected_event_start(Eventscell);

% Condition to plot the results of the automated method and the shorter windows around the events

if plot==1

   plotresults(TraceDataSumcell,Eventscell,Eventstart,dt)

end

% Write the events detected in the text file opened

if isnan(Eventstart)
```

```
        else

            for i=1:length(Eventstart)

            fprintf(fid, '%s %s %s %s \n',d(k).name,FDStrings{1,1},FDStrings{1,3},num2str(Eventstart(i)));

            end

        end

    end

end

% Close the text file opened

fclose(fid);

% Filtering function

function [H1,H2,V]=filter(Trace1,Trace2,Trace3,corner,dt)

if corner(1)==0
    corner=[0 inf];
end

% Filter parameters

if corner(1)>0
    n      = 3;
    ny = round(1 / dt / 2);
    Wn     = corner/ny;
    ftype = 'bandpass';
    [b,a] = butter(n,Wn,ftype);
    taper = tukeywin(size(Trace1,2), .05);
```

```
end

% Applying the filter to the traces

if corner(1)>0

   H1 = filtfilt(b, a, taper'.*Trace1);
   H2 = filtfilt(b, a, taper'.*Trace2);
   V = filtfilt(b, a, taper'.*Trace3);

else
   H1 = C1;
   H2 = C2;
   V = C3;
end

% Recursive STA/LTA algorithm

function [RR]=recursive_STA_LTA(TraceData,Ndat,S,L)

%Inputs to start

Csta=1/S;
Clta=1/L;
Icsta=1-Csta;
Iclta=1-Clta;
sta=0;
lta=1*10^-99; % To avoid division per zero
RR=zeros(1,Ndat); % STA/LTA ratio function

for i=1:Ndat
    sq=TraceData(i)^2;
    sta=Csta*sq+Icsta*sta;
    lta=Clta*sq+Iclta*lta;
    RR(i)=sta/lta;
```

```
    % Starts taking into account RR when we reach number of samples of the long term average

    if i<L
        RR(i)=0;
    end
end

% STA/LTA trigger algorithm

function [evtstrtid,evtendid]=trigger_threshold(RR,Ndat,dt,xcorr)

HighTresh=50; % High threshold value
LowTresh=0.5; % Low threshold value
mintriggap=7500; % Minimum gap between 2 events
maxtriggap=5000; % Maximum gap between the start and the end of an event


% Find where STA/LTA ratio function is above high treshold

[trigid]=find(RR>HighTresh);

% If the STA/LTA ratio is not higher than the threshold, start and end equal to 0

if isempty(trigid)
    evtstrtid=0;
    evtendid=0;
else

% If it is higher, count the number of events

j=1;
for i=2:length(trigid)

    % Identify the events applying the minimum sapce between 2 events
```

```
    if trigid(i)>(trigid(i-1)+mintriggap)
       j=j+1;
    end
end

% Create the vector depending on the number of events

evtstrtid=zeros(1,j);

% Give the values of the events starts to the vector, the first one must be a start of an event

evtstrtid(1)=trigid(1);
j=1;
for i=2:length(trigid)
    if trigid(i)>(trigid(i-1)+mintriggap)
       j=j+1;
       evtstrtid(j)=trigid(i);
    end
end

%The end vector must have the same lenght of the start vector

evtendid=zeros(1,j);
for i=1:length(evtstrtid)

    %Find the first point of the STA/LTA is under the low Treshhold start counting from the start of the
events

    [evtendidpos]=find(RR(evtstrtid(i):Ndat)<LowTresh,1);

    % If there is no point under = 0 or if the gap between the start and the end is so long, no events,
start and end equal to 0

    if isempty(evtendidpos)
        evtendid(i)=0;
        evtstrtid(i)=0;
    elseif evtendidpos+evtstrtid(i)-maxtriggap>evtstrtid(i)
```

```
        evtendid(i)=0;
        evtstrtid(i)=0;

    %If the conditions are false we have an event! we must sum the startevent position because the find
function is giving us the position between evtsartid:Ndat

    else
    evtendid(i)=evtstrtid(i)+evtendidpos;
    end
end

end

% Now we do the conversion to seconds multipling for the sample interval if xcorrelation is not going to be
applied

if xcorr==1
else
evtstrtid=evtstrtid*dt;
evtendid=evtendid*dt;
end

% Cross-correlation

function [Eventscell]=cross_correlation(RRcell,Eventscell,dt)

count=1; % to give the new values to the eventscell

% For every sensor we are going to use xcorr with all the sensors

npoints=length(RRcell{1,1});
for sensor=1:12

       % Repeat the xcorr process i times, number of events of the sensor

        for i=1:length(Eventscell{1,count})
```

```
            % If the vector has a 0 in the start position, no event, keep 0

            if Eventscell{1,count}(i)==0

            else
            % if not, we start the xcorr process, creating vector for the maximums of the xcorr between the
sensors

               xcorrvector=zeros(1,12);

            % Now we create small windows (start1,end1) (RR of the event) and big windows (start2,end2) for
the remaining RR

                     if round(Eventscell{1,count}(i)-1000)<1
                        start1=1;
                     else
                        start1=round(Eventscell{1,count}(i)-1000);
                     end
                     if round(Eventscell{1,count+1}(i)+1000)>npoints
                        end1=npoints;
                     else
                     end1=round(Eventscell{1,count+1}(i)+1000);
                     end
                     if round(Eventscell{1,count}(i)-2500)<1
                        start2=1;
                     else
                     start2=round(Eventscell{1,count}(i)-2500);
                     end
                     if round(Eventscell{1,count+1}(i)+2500)>npoints
                         end2=npoints;
                     else
                     end2=round(Eventscell{1,count+1}(i)+2500);
                     end

                     % We compute the auto-correlation with the RR of the event
```

```
                [c,lags]=xcorr(RRcell{1,sensor}(start1:end1),RRcell{1,sensor}(start2:end2));
                [y,v]=max(c);
                xcorrvector(1,sensor)=y;
                autolag=abs(lags(v));

            % Now we create a loop to evaluate the similarity between RR of the sensor to all the RR of
the remaining sensors applying cross-correlation

            for sensors=1:12

                if sensors~=sensor

                 [c,lags]=xcorr(RRcell{1,sensor}(start1:end1),RRcell{1,sensors}(start2:end2));

                 % we get the maximum for every sensor xcorr

                 [y,v]=max(c);

                 % the maximum is valid only if the lag is less than 100samples*abs((N° Geophone (event)
- N° Geophone))

                 if abs(lags(v))-autolag<100*abs(sensors-sensor)

                    xcorrvector(1,sensors)=y;

                 else

                    xcorrvector(1,sensors)=0;

                 end
                end
            end

            %Now we take the maximum of the auto-correlation and normalize the vector, we get numbers
between 0 and 1
```

```
                xcorrvectornorm=xcorrvector./xcorrvector(1,sensor);

                % Fix a threshold to declare which sensors have similar RR

                [n]=find(xcorrvectornorm>0.6);

                % we declare that the studied event is real if it has at least 4 sensors above the threshold,
if not equal to 0, we rase the event

                if length(n)>3

                   Eventscell{1,count}(i)=Eventscell{1,count}(i)*dt;
                   Eventscell{1,count+1}(i)=Eventscell{1,count+1}(i)*dt;

                else

                   Eventscell{1,count}(i)=0;
                   Eventscell{1,count+1}(i)=0;

                end


            end

        end

    % we have a count +2 because we want to study the next sensor, where his start and end events are in 3th
and 4th position in the vector Eventscell

    count = count+2;

end

% Start of the events detected
```

```
function [Eventstart]=detected_event_start(Eventscell)

%create a vector for the final cuts of every sensor

cutevents=cell(1,12);

%Start counting sensors

sensor=1;

%loop to get out the zeros of the start and end vectors of the Eventscell

for p=1:2:24

    count=0;

    % count if there is some number different to 0

    for i=1:length(Eventscell{1,p})

        if Eventscell{1,p}(i)==0
        else
        count=count+1;
        end

    end

    % if count is 0, all the numbers are 0, so final cut in this sensor is Nan

    if count==0
    cutevents{1,sensor}=NaN;
    else

    % if count is different to 0, create a vector of zeros with the number of events we have counted in the
vector
```

```
    cutevent=zeros(1,count);
    count=1;

    % Then give values to this vector only the start number

    for i=1:length(Eventscell{1,p})
        if Eventscell{1,p}(i)==0
        else
        cutevent(count)=Eventscell{1,p}(i);
        count=count+1;
        end
    end

    % Now we attribute this vector to the first cell of cutevents

    cutevents{1,sensor}=cutevent;

    end

    %To apply the same to all the vectors

    sensor=sensor+1;
end

%loop to compare the events of every sensor to the rest of the sensors, get always the minimum cut of an
event

for sensor=1:12

    %create a vector to store the minimum cuts for every event of a sensor

    mincutevents=zeros(1,length(cutevents{1,sensor}));

    %If the vector of the sensor is empty, skip all the process

    if isnan(cutevents{1,sensor})
```

```
    else

    %If not repeat the process i times, number of events

    for i=1:length(cutevents{1,sensor})

    % we define the initial cut of the event

    cutevent=cutevents{1,sensor}(i);

    % start to compare with all the sensors

    for sensor1=1:12

        %If the sensor to compare is empty skip

        if isnan(cutevents{1,sensor1})
        else

        %If not we do the process for all the event of the sensor

        for j=1:length(cutevents{1,sensor1})

        % We check if the event of the sensor to compare is the same event of the original sensor

        if cutevents{1,sensor}(i)-1<cutevents{1,sensor1}(j) &&
cutevents{1,sensor1}(j)<cutevents{1,sensor}(i)+1

            %If it is true, we check if the cut of the event of the other sensor is lower than our original
cut, if its true we define a new cutevent, at the end we will have the lowest cut

            if cutevents{1,sensor1}(j)<cutevents{1,sensor}(i)
           cutevent=cutevents{1,sensor1}(j);
           end
        end
        end
```

```
        end
    end

    % Now we define the minimum cut for all the events of the sensor

    mincutevents(1,i)=cutevent;
    end

    %And then we define the cell of cutevents with that vector

    cutevents{1,sensor}=mincutevents;
    end
end

%Now we have the minimum cuts of all the events of every sensor, so we are going to take the largest one
(contains more events)

%We create a vector to store the lengths of the cells

lencut=zeros(1,12);

%Loop to calculate the lenght of every cell

for sensor=1:12

    if isnan(cutevents{1,sensor})
    lencut(1,sensor)=0;
    else
    lencut(1,sensor)=length(cutevents{1,sensor});
    end
end

% we get the maximum value of this vector and then we pick the first one that has this number

value=max(lencut);
[~,pos]=find(lencut==value,1);
```

```
%Now, we create a loop to check if we have missed an event in the other sensors, because it is possible that
the largest cell doesn't contain all the events of the file

for sensor=1:12
    if isnan(cutevents{1,sensor})
    else

    % Repeat the process for every sensor, j times, number of events of every sensor

    for j=1:length(cutevents{1,sensor})

        count=0;

        %we have to compare the events of every sensor to every event of our largest sensor

        for i=1:length(cutevents{1,pos})

            %we check if the event of the other sensor make true this statment one time for an event of our
largest vector (pos)

            if cutevents{1,pos}(i)-1.5<cutevents{1,sensor}(j) &&
cutevents{1,sensor}(j)<cutevents{1,pos}(i)+1.5
            else
                count=count+1;
            end
        end

        % if count is equal to the lenght of our largest event means that the statment was false for all of
them, so we have missed an event, and we have to add it in our vector

        if count==length(cutevents{1,pos})
           cutevents{1,pos}(length(cutevents{1,pos})+1)=cutevents{1,sensor}(j);
        end
    end
    end
```

```
end

% if value is NaN means that we hadn't an event in all the traces so we don't have a cut and the number of
events don't increase

if value==0
    Eventstart=NaN;

else

% at the end we have the final cut vector of the file

Eventstart=cutevents{1,pos};

end

% Plot of all the results for a file

function plotresults(TraceDataSumcell,Eventscell,Eventstart,dt)

    % Plot the picks in the continuous data

    BE  = [0   dt*(size(TraceDataSumcell{1},2)-1)];
    xx=BE(1):dt:BE(2);

    h1=figure(23);clf;axes('Units','Normalized','Position',[.07 0.07 .91
.85],'Ydir','reverse','Ytick',[1:12]) ;hold on
    set(gca,'ColorOrder',[0 .6 0;1 0 0; 0 0 1; 0  0 0])

    sensor = 1;

    for i=1:2:24

    plot(xx, TraceDataSumcell{sensor}/max(abs(TraceDataSumcell{sensor}))*.5+sensor)
    hold on
       for j=1:length(Eventscell{1,i})
```

```
    if Eventscell{1,i}(j)==0

    else
       x=linspace(Eventscell{1,i}(j),Eventscell{1,i}(j));
       y=linspace(-1,1);
       plot(x,y*.5+sensor,'r')
       hold on
       x=linspace(Eventscell{1,i+1}(j),Eventscell{1,i+1}(j));
       y=linspace(-1,1);
       plot(x,y*.5+sensor,'b')
       hold on
    end
   end
sensor = sensor+1;
end

% Plot short windows cut for every event

if isnan(Eventstart)
    else
       for i=1:length(Eventstart)
           if Eventstart(i)>8
               Eventsend=10;
           else
               Eventsend=Eventstart(i)+2;


           end

           if Eventstart(i)<1
               Eventstart(i)=0;
           else
           Eventstart(i)=Eventstart(i)-1;
           end
```

```
                BE  = [0   dt*(size(TraceDataSumcell{1}(round(Eventstart(i)/dt)+1:round(Eventsend/dt)),2)-
1)];
                xx=BE(1):dt:BE(2);

                for m=1:12

                h2=figure(i);
                plot(xx,
TraceDataSumcell{m}(round(Eventstart(i)/dt)+1:round(Eventsend/dt))/max(abs(TraceDataSumcell{m}(round(Eventst
art(i)/dt)+1:round(Eventsend/dt))))*.5+m)
                hold on

                end


            end

    end

    % Close all windows to start with a new continuous data file

    waitfor(h1)
    waitfor(h2)
```

Appendix B: Code in Matlab created by James Verdon in 2012 to estimate *b*-values

```
function [b,a,berr,ML_min,pcdf,mcdf]=BvalueKS(ML,E,method,eval,iplot)
%-------------------------------------------------------------------------
%
%  This software is distributed under the term of the BSD free software
%  license.
%
%  Copyright:
%     (c) 2012, James Verdon
%
%  All rights reserved.
%
%   * Redistribution and use in source and binary forms, with or without
%     modification, are permitted provided that the following conditions are
%     met:
%
%   * Redistributions of source code must retain the above copyright notice,
%     this list of conditions and the following disclaimer.
%
%   * Redistributions in binary form must reproduce the above copyright
%     notice, this list of conditions and the following disclaimer in the
%     documentation and/or other materials provided with the distribution.
%
%   * Neither the name of the copyright holder nor the names of its
%     contributors may be used to endorse or promote products derived from
%     this software without specific prior written permission.
%
%
%   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
%   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
%   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
%   A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT
%   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
%   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
%   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
%   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
```

```
%   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
%   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
%   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
%
%------------------------------------------------------------------------

% Check arguments, and set default values where necessary
if ~exist('E', 'var')
  E = 0.05;  % Allowable residual when fitting
end
if ~exist('method', 'var')
  method = 'ks';  % Calculation method
end
if ~exist('iplot', 'var')
  iplot = 0;   % Plotting tag
end
if ~exist('eval', 'var')
  eval = NaN;
end
if isfinite(eval)
    if eval >= E
        error('Error value cannot be larger than E');
    end
end

ierror=0;
if isempty(ML)
    ierror=1;
end
if size(ML,2) > size(ML,1)
    ML=ML';
end


% Set the K-S test significance number
% Find the significance level to use
signum=[1.07 1.14 1.22 1.36 1.63];
sigval=[0.2 0.15 0.1 0.05 0.01];
```

```
sig=signum(E==sigval);
if isempty(sig)
    error('Significance must be 20%, 15%, 10%, 5% or 1% only')
end



% Create real distribution and normalised cumulative real distribution
 [nall,xall]=hist(ML,sort(ML));
 pcdf=cumulative_magnitude(nall,0);

% Sort event magnitudes
MLs=sort(ML);



% Loop over increasing cut-off magnitude, find minimum magnitude at which
% straight line fit is accepted
for i=1:length(MLs)
    MLc=MLs(i:end);
    ML_min=min(MLc);

    if length(MLc) < 35
        %fprintf('%s \n','No straight line fit has been found, b is not calculated');
        ierror = 1;
        %break
    end

    % Compute b for cut-off distribution
    switch method
        case 'ks'
            [b,a,P]=bvalue_ks(MLc);


        case 'explicit-ks'
            [b,a,P]=bvalue_explicit_ks(MLc);
```

```
        otherwise
            error('B-value calculation method not recognised')

    end


    % Test reconstructed b-value with data with K-S
    nrecon=10.^(-b*MLc);
    nrecon=nrecon/max(nrecon);


    % Test differences between observed and reconstructed profile
    d=max(abs(P - nrecon));
    sig_n=sig/sqrt(length(MLc));

    % If d fits the profile, accept the results finish loop
    if d < sig_n
        break
    end

end

% If an error has been encountered (not enough events, or no straight line
% fit) set values to NaNs, make a plot and exit

if ierror == 1;
    b=NaN; a=NaN; berr=[NaN NaN]; ML_min=NaN; pcdf=NaN; mcdf=NaN;
%     if iplot == 1
%         figure
%         hold on
%         box on
%         plot(xall,pcdf,'go-')
%     end
    return
```

```
end

% Compute errors limits based on pre-calculated value of Mmin
if isfinite(eval)
    [berr,ef]=bvalue_ks_limits(b,MLc,eval);
else
    berr = [NaN NaN];
end

% Final stage - back calculate distributions and make a plot

% Construct modelled cdf from finalised b values
mcdf=10.^(a - b*MLs) ;  % Expected values computed for all magnitudes


if iplot == 1

    hold on
    box on
    plot(xall,pcdf,'go-')
    plot(xall(xall>ML_min),pcdf(xall>ML_min),'bo-')
    plot(xall(xall>ML_min),mcdf(xall>ML_min),'r','LineWidth',2)
    set(gca,'YScale','log')

    % Plot error lines if they have been computed
    if isfinite(eval)
        plot(xall(xall>=ML_min),ef(:,1),'r--')
        plot(xall(xall>=ML_min),ef(:,2),'r--')
    end

end
```

Appendix C: Code in Matlab created by James Verdon in 2012 to estimate D_C-values

```
function [dc,dcerr,cr,rmax]=correlation_dimension(x,y,z,E,eval,r,iplot)
%
% [dc,dcerr,cr,rmax]=correlation_dimension(x,y,z,E,eval,r,iplot);
%
% Computes the two-point correlation dimension for a population of seismic
% events. This describes the 'dimensionality' of the event spatial
% distribution. For example, if events are distributed along a linear
% feature, Dc = 1, if they fall on a plane, Dc = 2, and if they are filling
% up a 3D volume, Dc = 3.
%
% Dc is computed from the gradient of log(C(r))/log(r), where C(r) is
% 2N(R<r)/(n(n-1)): the fraction of event pairs separated by a distance
% less than r (so N(R<r) is the number of event pairs closer than r, and n
% is the total number of events).
%
%
% If the locations are fractally distributed, log(C(r))/log(r) will be
% linear. At larger r, this will break down as saturation occurs (r becomes
% larger than all the event pairs, so dC(r)/dr goes to 0). This routine
% searches for the maximum r at which a linear fit is appropriate to the
% data, and computes the gradient only below this saturation point.
%
%
% INPUTS:   x,y,z - spatial coordinates of each event
%           E - residual value to accept straight line fit
%           r - vector listing r bins for cr calculation
%           iplot - tag to plot log(C(r)) vs log(r)
%
% OUTPUTS:  dc - correlation dimension
%           dcerr - error in the correlation dimension
%           cr - the C(r) values as a function of r
%           rmax - the rmax cut-off value
%
% Written by J.P. Verdon, University of Bristol, 2012
%
```

```
%
%

%-------------------------------------------------------------------------
%
%  This software is distributed under the term of the BSD free software
%  license.
%
%  Copyright:
%     (c) 2012, James Verdon
%
%  All rights reserved.
%
%   * Redistribution and use in source and binary forms, with or without
%     modification, are permitted provided that the following conditions are
%     met:
%
%   * Redistributions of source code must retain the above copyright notice,
%     this list of conditions and the following disclaimer.
%
%   * Redistributions in binary form must reproduce the above copyright
%     notice, this list of conditions and the following disclaimer in the
%     documentation and/or other materials provided with the distribution.
%
%   * Neither the name of the copyright holder nor the names of its
%     contributors may be used to endorse or promote products derived from
%     this software without specific prior written permission.
%
%
%   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
%   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
%   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
%   A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT
%   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
%   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
%   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
%   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
%   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
```

```
%   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
%   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
%
%------------------------------------------------------------------------



%% Check which variables exist, and assign those that don't with default
% values
if ~exist('iplot', 'var')  % iplot is the tag to plot C(r) v r
  iplot = 0;               % Set to 1 to force the plot
end
ir=0;

if exist('r','var') % r is the vector of r bins over which C(r) is computed
    if isfinite(r)  % If not set, C(r) will be computed at each r value of
        ir=1;       % the event pair distances, which might take a long time
    end
end

if ~exist('E','var') % E is the allowable residual between model and
  E = 0.05;          % observation when fitting the straight line
end                  % If not set, use 5%

if exist('eval','var')
    if eval <= E
        error('Error value cannot be smaller than E');
    end
else
    eval = NaN;
end
dcerr=[NaN NaN];

%% Stage 1: Compute and sort event-event distances

% Check x, y and z have same lengths
if length(x) ~= length(y) || length(x) ~= length(z) || length(y) ~= length(z)
```

```
    error('Lengths of spatial vectors x, y and z are not equal')
end

ne=length(x);  % Number of events
np=ne*(ne-1)/2; % Number of event pairs:

ed = NaN(np,1);
c=0;

% Loop over pairs to compute the event distance
for i=1:ne
    for j=i+1:ne
        c=c+1;
        ed(c)=event_distance(3,x(i),x(j),y(i),y(j),z(i),z(j));
    end
end
eds=sort(ed); % Sort event pair distances into ascending order


% If r has not been set, calculate C(r) at each event pair distance
if ir == 0
    r=eds;
end
if length(r) > 1e6
    message('r is longer than 1 million. This could take a while. Id recommend a trip to Neros....')
end



% Make sure r is a column
if size(r,1) < size(r,2);
    r=r';
end
r=sortrows(r,-1); % Sort radii into descending order
```

```
%% Stage 2 - Compute C(r)

% Compute C(r) - number of points closer than a given radius r
cr = NaN(length(r)-1,1);
for i=1:length(r)-1
    n_lt_r = length(eds(eds<r(i)));
    cr(i)=2*n_lt_r/np;
end
r=r(1:end-1);

% Take logarithm of both cr and radius values
lr=log10(r);
lcr=log10(cr);


%% Stage 3, decrease rmax until a linear fit is found
for i=1:length(r);
    % If we have reached to within 10 points of the minimum r, give up and
    % get out, as Dc will not be stable
    if i >= length(r)-10;
     %   warning('Fractal:NoStraightLine','Not straight line fit has been found, Dc is not calculated')
%         fprintf('%s \n','No straight line fit has been found, Dc is not calculated');
        dc=NaN;
        rmax=NaN;
%         if iplot == 1
%             figure
%             hold on
%             box on
%             plot(lr,lcr,'go-')
%             xlabel('Log_{10}R')
%             ylabel('Log_{10}N(r<R)')
%         end
        return
    end

    grad_method = 'slope-fit';
```

```
% Find gradient by averaging the gradient along the C(r) curve
if strcmp(grad_method,'grad_average')
    c=0;
    grad=NaN(1,length(i:length(r)-2));
    for j=i:length(r)-2
        c=c+1;
        grad(c)=(lcr(j)-lcr(j+2))/(lr(j)-lr(j+2));
    end

    dc=mean(grad); % Dc is the mean of the gradient values

    cint=mean(lcr(i:end))-dc*mean(lr(i:end)); % Intersect value


% Or find the gradient using a slope-fitting function
elseif strcmp(grad_method,'slope-fit')
    [cint,dc,~,~]=slope(lr(i:end),lcr(i:end));
end


% Test to see if linear fit is appropriate
fit=polyval([dc,cint],lr);  % Best fit line in log space
recon=10.^fit;  % ^10 for best fit in linear space

% Compare fit with observation
resid=sum(abs(cr(i:end)-recon(i:end)))/sum(cr(i:end));


% Test if residual between model and observed is below cut-off threshold
% Exit loop (and crack open a beer) if it is
if resid <= E
    ifin=i;
    rmax=r(i);
    bfit=fit;
    break
end
```

```
end



%% Stage 5: Keeping fixed rmax, search over Dc to find error limits
if isfinite(eval)
    ddc=0.01;

    % Upper limit first
    iexit=0;
    tdc=dc;
    while iexit == 0;
        tdc = tdc + ddc;

        % Determine cint from the mean point
        rmean=mean(lr(ifin:end));
        cmean=mean(lcr(ifin:end));
        cint=cmean-tdc*rmean;

        % Test to see if linear fit is appropriate
        fit=polyval([tdc,cint],lr);  % Best fit line in log space
        recon=10.^fit;  % ^10 for best fit in linear space

        % Compare fit with observation
        resid=sum(abs(cr(ifin:end)-recon(ifin:end)))/sum(cr(ifin:end));


        % Once the residual is exceeded, take the previous result as the
        % maximum allowed dc
        if resid > eval
            dcerr(1)=tdc-ddc;
            efit(:,1)=fit;
            break
        end
    end

    % Now lower limit
```

```
    tdc=dc;
    while iexit == 0;
        tdc = tdc - ddc;

        % Determine cint from the mean point
        rmean=mean(lr(ifin:end));
        cmean=mean(lcr(ifin:end));
        cint=cmean-tdc*rmean;

        % Test to see if linear fit is appropriate
        fit=polyval([tdc,cint],lr);  % Best fit line in log space
        recon=10.^fit;  % ^10 for best fit in linear space

        % Compare fit with observation
        resid=sum(abs(cr(ifin:end)-recon(ifin:end)))/sum(cr(ifin:end));


        % Once the residual is exceeded, take the previous result as the
        % maximum allowed dc
        if resid > eval
            dcerr(2)=tdc+ddc;
            efit(:,2)=fit;
            break
        end
    end
else
    dcerr=[NaN NaN];
end



%% Stage 6: Make a picture if iplot == 1

if iplot == 1

    hold on
    box on
```

```
    plot(lr,lcr,'go-')
    plot(lr(ifin:end),lcr(ifin:end),'bo-')
    plot(lr(ifin:end),bfit(ifin:end),'r-','LineWidth',2)
    if isfinite(eval)
        plot(lr(ifin:end),efit(ifin:end,:),'r--')
        plot(lr(ifin:end),efit(ifin:end,2),'r--')
    end
    xlabel('Log_{10}(r)')
    ylabel('Log_{10}(C(r))')

end
```