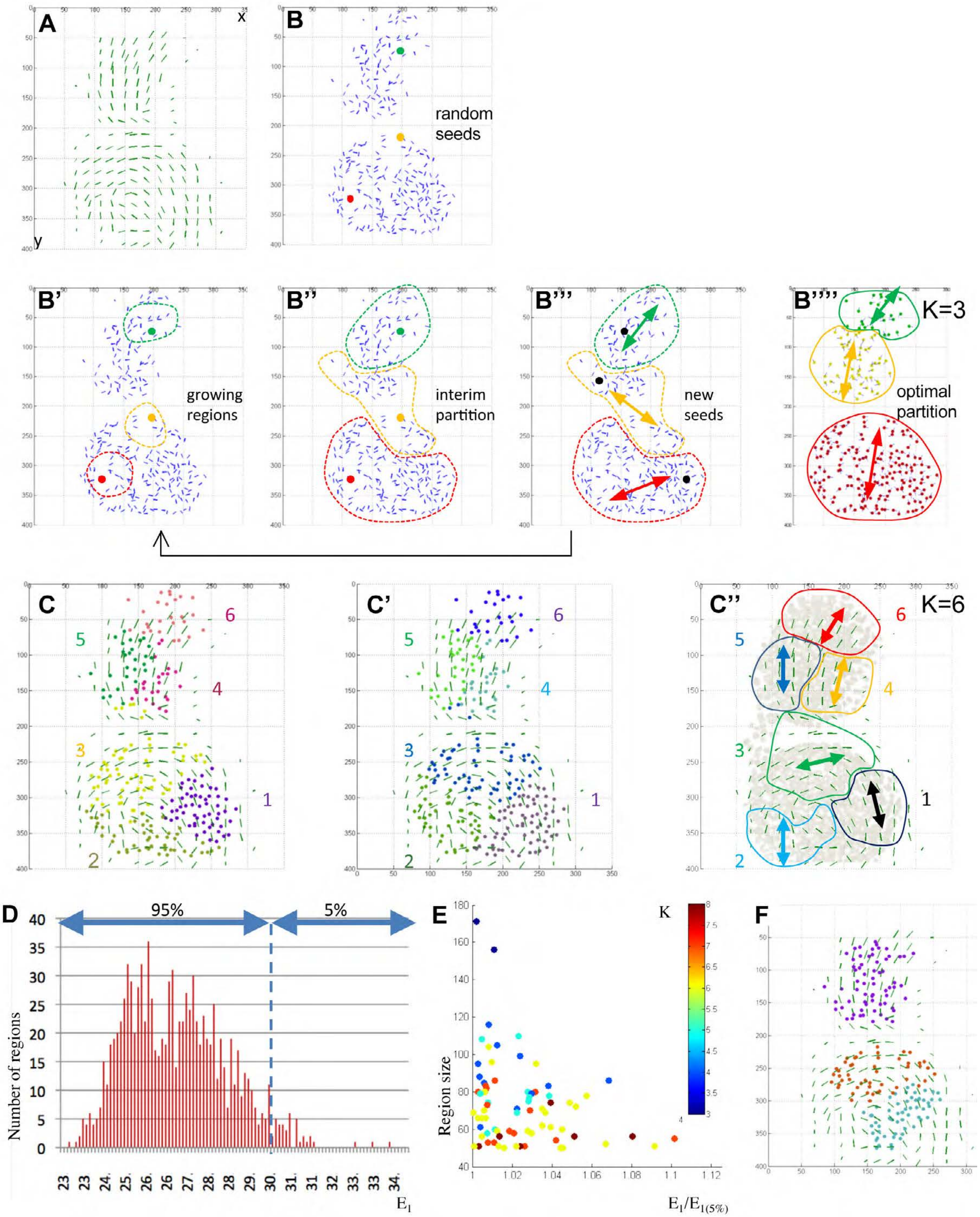

A
x
y
B
random seeds
B'
growing regions
B''
interim partition
B'''
new seeds
B''''
K=3
optimal partition
C
C'
C''
K=6
1
2
3
4
5
6
D
95%
5%
Number of regions
E_1
E
Region size
K
$E_1/E_{1(5\%)}$
F

Fig. S1. K-means clustering algorithm. (**A**) Map of local average directions (green bars) of the centrosome-nucleus axes in the embryonic heart shown in Fig. 3D. (B-B′′′′) Summary of the main steps of the K-means clustering algorithm, with K (number of regions)=3 and using the planar components of the axes. (**B**) Three seeds (green, red, yellow) are chosen randomly among the raw data set (blue axes); (**B′**) the three regions, outlined by colored dotted lines, are grown around these seeds by progressively allocating the neighbors with an axis most parallel to that of the seed; (**B′′**) at the end of the process, a partition of the whole data set into connex regions is reached; (**B′′′**) new seeds (in black) are computed for each region, by finding the cell with the axis closest to the average direction of the region (green, red, yellow double arrows). (**B′′′′**) This leads to a new iteration and, finally, after a few thousand iterations, to a stable optimal partition. (**C**,**C′**) When varying the parameter K, we obtained different partitions. Examples of two different partitions, both with K=6, and 5000 iterations, but different initial seeds. Dots of the same color belong to the same region with the best axial coordination. Although the algorithm does not strictly converge, when repeating the procedure with different initial seeds, it leads to very similar partitions for a given value of K (regions numbered 1 to 6). (**C′′**) Schematic in which a region is the intersection of the two corresponding regions of the partitions C and C′. (**D**) Bootstrap method for evaluating the threshold of the eigenvalue E_1, above which axes of a distribution were considered sufficiently parallel. As an example, the distribution of 1000 random regions of 50 axes from the data set of A is shown. A threshold is defined such that only 5% of the random regions have an eigenvalue E_1 above it: here $E_{1(5\%)}$=30 for regions of 50 axes. (**E**) Influence of the parameter K (color coded) on the characteristics of regions generated by the K-means clustering algorithm on a data set of 347 axes (for each K-value, the algorithm was run 100 times, with 100 iterations at each run). Lower K tended to produce larger regions with a lower degree of axial coordination ($E_1/E_{1(5\%)}$). (**F**) Output map of the clustering algorithm after selection of the regions with the best axial coordination, independently of K. These three best-oriented regions are color coded on the map of the local average directions per box (green). Further validation by a statistical test is required and shown in Fig. 3D. 3D maps are projected on the *xy* plane.

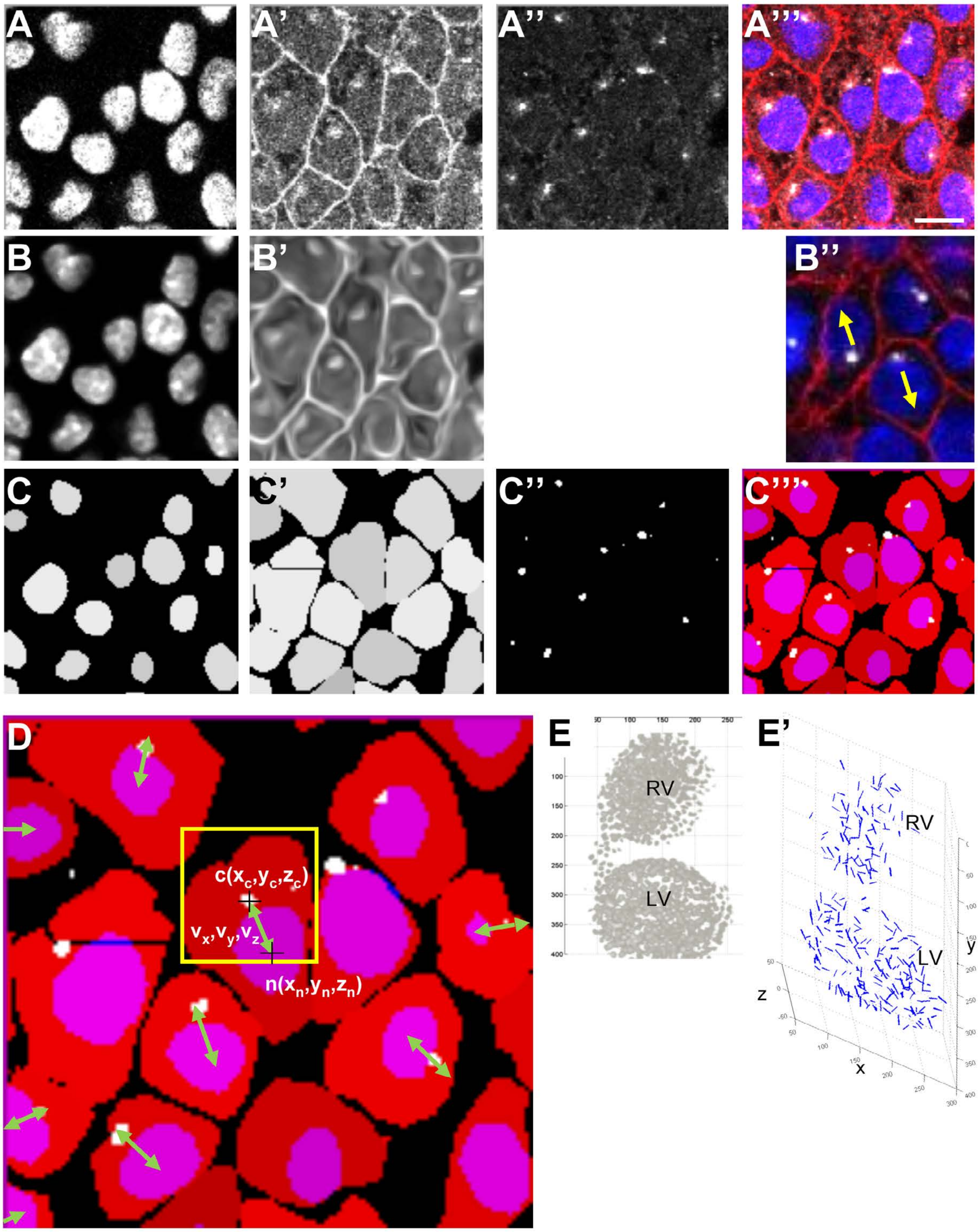
A
A'
A''
A'''
B
B'
B''
C
C'
C''
C'''
D
c(x_c,y_c,z_c)
v_x,v_y,v_z
n(x_n,y_n,z_n)
E
RV
LV
E'
RV
LV
x
y
z

Fig. S2. Automatic detection of the centrosome-nucleus axes of cell polarity. (**A-A‴**) Example image of embryonic myocardial cells, with labeled nuclei (A and blue in A‴), membranes (A′ and red in A‴) and centrosomes (A″ and white in A‴). Scale bar: 10 μm. (**B**,**B′**) Filtered images of the nuclei (B) and membranes (B′) that are used as input for the segmentation process. (**B″**) Yellow arrows indicate neighboring cells with anti-parallel orientations of their centrosome-nucleus vectors. (**C-C‴**) Results of the automatic segmentation of the nuclei (C and magenta in C‴), the cells (C′ and red in C‴) and the centrosomes (C″ and white in C‴). (**D**) For the automatic pairing between nuclei and centrosomes, the nearest voxel belonging to a nucleus (n, magenta) was searched within a box (yellow) of about one cell diameter wide (11×11×11 μm) around each centrosome (c, white). The 3D centrosome-nucleus axis (v_x, v_y, v_z), shown as a green double-headed arrow, was computed from the centroids (x_c,y_c,z_c) and (x_n,y_n,z_n) of the centrosome and nucleus, respectively. (**E**,**E′**) In a scan of 594 nuclei (E), a 3D data set of 250 centrosome-nucleus axes was extracted. Each blue bar corresponds to the axis of one cell (E′). LV, left ventricle; RV, right ventricle.

Appendix S1. MATLAB code of the K-means clustering algorithm.

The input of this code is an N x 6 array, called “data”, where N is the number of axes to be clustered and the 6 columns contain the location coordinates of each axis (X, Y, Z) and the coordinates of its unit vector (V_x, V_y, V_z). An N x 2 array, called “Table” should also be given as input, containing for each region size (from 1 to N: first column) the 5% threshold E_l eigenvalue (second column), as computed by the bootstrap method. The aim of the clustering algorithm is to minimize, for a given value of “RegionNumber” (K in the main text, defining the number of regions in the partition), the total approximation error, i.e. the sum of individual deviations of axes from the average direction of their region.

The main output of this code is “SignifRegionsLeagueTable”. This is an array listing all the significant regions found by the algorithm, i.e. regions containing more than “SizeLimit” axes (the minimum size according to Ringrose and Benn, 1997) and having an E_l eigenvalue above the 5% threshold. For each region (one per line), this table gives the E_l eigenvalue (column 1), how many times this region was found (column 2), the size of the region (column 3), the eigenvector V_1 of the region (columns 4-6) , and the list of all axes (identified by their rank in the “data” array) in the region (column 7 onward).

For the searches presented in the main text we ran this algorithm with various values of the parameter “RegionNumber” (K), and added the “SignifRegionsLeagueTable” from these successive runs before sorting the regions according to their E_l eigenvalue. The range of values of K is between K=2 up to a value for which the algorithm can no longer find regions containing “SizeLimit” or more points. For typical data sets, in which the number of axes was between 200 and 400, and “SizeLimit” = 50, the maximum K value was 7 or 8.

The algorithm first creates a list of the “NeighborNumber” (NN) nearest neighbours of each data point, where NN is chosen empirically as the smallest number allowing all the data points to be allocated to a region at the end of the clustering process. Typically, for our data sets, NN=7 proved sufficient. K seeds are then randomly chosen among the data points. A queue of length K*NN is then created including all the neighbours of the K seeds, sorted in ascending order of deviation of their axis from that of the seed. This queue is used to allocate each data point to one of the K regions: the first point in the queue is allocated to the region of the neighbouring seed and deleted from the queue; all the neighbours of this point are then added at the end of the queue. With this addition at the end, independently of the orientation of the new neighbours, the regions remain compact. Points are tagged so that they may only be allocated to the same region as their neighbour, which was previously allocated : this ensures the connexity of a region. The procedure is reiterated with the next first point in the queue until all points have been allocated. At the end of the allocation process, a partition of all the data points in K regions is obtained. The next iteration proceeds by replacing the previous seeds by the best proxy for each region, i.e. the data point associated with the axis most closely parallel to the average direction of the region. A new partition is built from these new seeds, leading to a further iteration. The number of iterations is set as “LoopNumber” (100-200 have proven sufficient). In order to escape local minima, the basic algorithm is run many times (“IterNumber”), starting with different seeds, and improved by inserting and deleting regions at regular intervals (“Divisor”) during iterations.

MATLAB code:

```
% clustering algorithm
% input data is a Nx6 array where N is the number of axes,
% and the 6 columns are: X,Y,Z (coord of axis location),Vx,Vy,Vz (coord of
axis vector)
% input Table is a Nx2 array giving the 5% significance threshold of any
region as a function of the number of axes in the region
N=250; % number of axes
RegionNumber=3; %number of regions
NeighborNumber=7; % number of neighbors to be fed into the queue
ThresholdRatio=1.0; % if 1 means that the 5% threshold will be used
SizeLimit=35; % only regions including more than SizeLimit axes will be
included in the SignifRegionsLeagueTable array
IterNumber=100; % number of trials with different initial seeds
LoopNumber=100; % number of iterations starting from a set of initial seeds
Divisor=51; % every Divisor loops, 2 regions are fused, and a new one is added
MaxDim=1000; % maximum dimension of the grid

% computing neighbor list of each axis, fed into AllNeighborList array
AllNeighborList=zeros(NeighborNumber,N);
NeighborSource=ones(N,N);
for i=1:N
    NeighborSource(i,i)=0;
end;
k=ones(N,1); % will count the number of neighbors fed into each column of
AllNeighborList, stopping at NeighborNumber
for distance=1:1:MaxDim
    for i=1:N

        for j=1:N
```

```
            if k(j,1)~=(NeighborNumber+1) && (NeighborSource(i,j)~=0)...
               && sqrt((data(j,1)-data(i,1))^2+(data(j,2)-
data(i,2))^2+(data(j,3)-data(i,3))^2)<distance;
               AllNeighborList(k(j,1),j)=i;
               NeighborSource(i,j)=0;
               k(j,1)=k(j,1)+1;

            end;
        end;
    end;
    if sum(k)==((NeighborNumber+1)*N)
        break;
    end;
end;

% trials starting each time with new seeds
iter=0;
    SignifRegionsRank=1;
    SignifRegions=zeros(N,2*IterNumber); % array listing the member-axes of
the significant regions at the end of each trial
    SignifV3=zeros(3,2*IterNumber); % array giving the 3rd eigenvector of each
region
    SignifThreshold=zeros(3,2*IterNumber); % array giving the threshold and
count (how many times it appeared at the end of a loop) of each region

    % initialization of the best regions as defined by the criterium chosen
    % (see line 357)
    BestProd = 0;
    BestSum=0;
    BestRegions=zeros(1,RegionNumber);
    BestList=zeros(N,N);
    BestThreshold=zeros(RegionNumber,1);
    BestNbSignifPoints=0;
    BestRegionsMembers=zeros(N,RegionNumber);

    maximum=-1;

while iter<IterNumber

Regions=zeros(1,RegionNumber); % array where the seeds of regions are listed
RegionsList=zeros(N,N); % array where all members of the region are listed
% random choice of seeds
RegionIndex=randperm(N);
for i=1:RegionNumber
    RegionsList(RegionIndex(i),RegionIndex(i))=1; % The RegionsList array has
ones in the diagonal when it intersects the ranks of the seeds in the data
array

    Regions(1,i)=RegionIndex(i);
end;

% Optimization loop proceeding from the same initial seeds
threshold=zeros(RegionNumber,1);
loops=1;
```

```
while  loops<LoopNumber

    % module of regions deletion/addition
    if mod(loops,Divisor) == 0
        [MinThreshold,Index]=min(threshold); % Index is the index of the
region with the worst score
        WorstRegionProxy=LastRegions(1,Index); % WorstRegionProxy is the proxy
of the worst region
        worstrating=1;
        for i=1:N % looking for the worst axis in the worst region (which axis
will later be introduced as the added seed of the additional region)
            if LastRegionsList(i,WorstRegionProxy)~=0 % computes the
distorsion between this axis and its proxy
               rating=abs(dot(data(i,4:6),data(WorstRegionProxy,4:6)) );
               if rating < worstrating
                   worstrating=rating;
                   worstpoint=i; % this is the axis that will be used as a new
seed for the next loop (to be done by changing RegionsList)
               end;
            end;
        end;

        % searching the two regions to be fused = neighboring regions,
        % which when fused will give the lowest distortion
        BestDistortion=0;
                % this will create a table, called LastRegionsMembers, with
                % a list of member-points in each region of LastRegions
                LastRegionsMembers=zeros(N,RegionNumber);
                for m=1:1:RegionNumber
                    n=1;
                    for i=1:1:N
                        if LastRegionsList(i,LastRegions(1,m))~=0
                            LastRegionsMembers(n,m)=i;
                            n=n+1;
                        end;
                    end;
                end;
        tf=0; % truth function = 0 if regions j and k are not contiguous, and
1 if they are
        for j=1:RegionNumber % first region

            for k=j+1:RegionNumber % second region

                    for n=1:N % search over all member axes of the first
region
                        if LastRegionsMembers(n,j)~=0 % axis n of the first
region
                            tf=ismember(LastRegionsMembers(n,j),
AllNeighborList(:,k)); % checks whether regions j and k are contiguous : is
axis n of the first region a neighbor of any of the axes of the second region
                            if tf==1 % these 2 regions are contiguous
                                % computes the distortion of these two regions
when fused together
                                FusedRegions=zeros(N,6);
                                rank=1;
                                for i=1:N % builds the data array for all the
axes in the fused region
                                    if LastRegionsMembers(i,j)~=0

FusedRegions(rank,4:6)=data(LastRegionsMembers(i,j),4:6);
                                    rank=rank+1;
                                    end;
                                end;
                                for i=1:N
                                    if LastRegionsMembers(i,k)~=0

FusedRegions(rank,4:6)=data(LastRegionsMembers(i,k),4:6);
                                    rank=rank+1;
                                    end;
                                end;
                                % computes the average axis-vector in the
region
                                dircos=zeros(3,3);
                                dircos(1,1)=sum(FusedRegions(:,4).^2);

dircos(1,2)=sum(FusedRegions(:,4).*FusedRegions(:,5));

dircos(1,3)=sum(FusedRegions(:,4).*FusedRegions(:,6));
                                dircos(2,1)=dircos(1,2);
                                dircos(2,2)=sum(FusedRegions(:,5).^2);

dircos(2,3)=sum(FusedRegions(:,5).*FusedRegions(:,6));
                                dircos(3,1)=dircos(1,3);
                                dircos(3,2)=dircos(2,3);
                                dircos(3,3)=sum(FusedRegions(:,6).^2);

                                % comparing the third eigenvalue to the 5%
significance Table
                                [V,D]=eig(dircos);
                                distortion=D(3,3)/Table(rank,2);
                                transpV = V';
                                if distortion>BestDistortion
                                    BestDistortion=distortion;
                                    Fused1=LastRegions(1,j); % the two regions
to be fused
                                    Fused2=LastRegions(1,k);
                                end;
                                break; % since the two regions are contiguous,
there is no need to further explore the n axes of the first region
                            end;% end of computation of fused regions
distortion
                        else % the end of the list of axes in the first region
has been reached
                            break;
                        end;
                    end;
            end;
        end;
        % computation of the proxy of the new fused region
                bestdotproduct=0;
                for i=1:N
```

```
                    if LastRegionsList(i,Fused1)~=0
                        dotproduct=abs(dot(transpV(3,:),data(i,4:6)));
                        if dotproduct > bestdotproduct
                            bestdotproduct=dotproduct;
                            bestproxy=i;

                        end;
                    end;
                end;
                for i=1:N

                    if LastRegionsList(i,Fused2)~=0
                        dotproduct=abs(dot(transpV(3,:),data(i,4:6)));
                        if dotproduct > bestdotproduct
                            bestdotproduct=dotproduct;
                            bestproxy=i;

                        end;
                    end;
                end;

    % if the added distortion is less than half the distortion of the worst
region
    if (BestDistortion -0.5*(threshold(j,1)+threshold(k,1)))< 0.5*MinThreshold

    % creates the new list of seeds: deleting Fused1 and Fused2 (replaced
    % by bestproxy), and adding worstpoint

    Regions=LastRegions;
    Regions(1,Fused1)=bestproxy;
    Regions(1,Fused2)=worstpoint;
    RegionsList=zeros(N,N);
    for i=1:RegionNumber
    RegionsList(Regions(i),Regions(i))=1; % The RegionsList array has ones in
the diagonal when it intersects the ranks of the seeds in the data array
    end;
    end;

    end;
    % end of module of regions deletion/addition

% creating a queue of seeds-neighbors sorted in ascending order of
% "distortion" (angle between the axis-vectors of the neighbor and its
% seed)
% queue size is RN*NN,3: 1rst row=axes identity, 2nd row=seed identity, 3rd
row=distortion
queue=zeros(3,RegionNumber*NeighborNumber);
for i=1:1:RegionNumber
    queue(1,NeighborNumber*(i-1)+1:1:NeighborNumber*i)=
AllNeighborList(:,Regions(1,i));
    queue(2,NeighborNumber*(i-1)+1:1:NeighborNumber*i)=Regions(1,i);
end;
for i=1:1:RegionNumber*NeighborNumber
    queue(3,i)=abs(dot(data(queue(2,i),4:6),data(queue(1,i),4:6)));
end;
```

```
transpqueue=queue';
transpqueue=sortrows(transpqueue,3);
queue=transpqueue';

% allocating axes to the various regions, following the order of the queue
Allocated=zeros(N,1); % keeps track of axes already allocated

for i=1:1:RegionNumber
    Allocated(Regions(1,i),1)=1;
end;



while size(queue,2)~=0
    if  queue(1,size(queue,2))~=0 && Allocated(queue(1,size(queue,2)))~=1%
checks that the 1rst axis in the queue is not already allocated to a region
        RegionsList(queue(1,size(queue,2)),queue(2,size(queue,2)))=1;%
registers the 1rst axis in RegionsList: a 1 in line=axis,column=seed
        Allocated(queue(1,size(queue,2)))=1;
        neighbors3=AllNeighborList(:,queue(1,size(queue,2)));% find the
neighbors of this newly allocated axis
        Added=zeros(3,NeighborNumber-1);
        rank=1;
        for i=1:NeighborNumber
            if neighbors3(i)~=queue(2,size(queue,2)) &&
Allocated(neighbors3(i))~=1 % excl the seed and already allocated points from
neighbors to be added to the queue
               Added(1,rank)=neighbors3(i);
               Added(2,rank)=queue(2,size(queue,2)); % this neighbor is given
the same seed as its precursor

Added(3,rank)=abs(dot(data(Added(1,rank),4:6),data(Added(2,rank),4:6)));
               rank=rank+1;
            end;
        end;

        queue(:,size(queue,2))=[]; % deletes the last row (first in the queue)
        queue=[Added queue];
    else
        queue(:,size(queue,2))=[]; % deletes the last row (first in the queue)
    end;

end;

% finding the best center for each region
S=sum(RegionsList);
bestscore=zeros(RegionNumber,1); % list of scores of each region
bestpoint=zeros(RegionNumber,1); % list of best centers

k=1; % index of the region
V3Table=zeros(3,RegionNumber); % table of third eigenvectors
for region=1:RegionNumber % iterates over all regions
        rank=1;
        neighbormatrix=zeros(S(Regions(region)),3);
        for i=1:N % builds the data array for all the points in the region
            if RegionsList(i,Regions(region))~=0
```

```
                neighbormatrix(rank,:)=data(i,4:6);
                rank=rank+1;
            end;
        end;
        % computes the average axis-vector in the region
        dircos=zeros(3,3);
        dircos(1,1)=sum(neighbormatrix(:,1).^2);
        dircos(1,2)=sum(neighbormatrix(:,1).*neighbormatrix(:,2));
        dircos(1,3)=sum(neighbormatrix(:,1).*neighbormatrix(:,3));
        dircos(2,1)=dircos(1,2);
        dircos(2,2)=sum(neighbormatrix(:,2).^2);
        dircos(2,3)=sum(neighbormatrix(:,2).*neighbormatrix(:,3));
        dircos(3,1)=dircos(1,3);
        dircos(3,2)=dircos(2,3);
        dircos(3,3)=sum(neighbormatrix(:,3).^2);

        % comparing the third eigenvalue to the 5% significance Table
        [V,D]=eig(dircos);
        threshold(k)=D(3,3)/Table(S(Regions(region)),2);
        transpV = V';
        V3Table(:,k)=V(:,3);

        % identifying the axis with axis-vector closest to average vector
        for i=1:N

            if RegionsList(i,Regions(region))~=0
               score=abs(dot(transpV(3,:),data(i,4:6)));
               if score > bestscore(k)
                  bestscore(k)=score;
                  bestpoint(k)=i;

               end;
            end;
        end;
        k=k+1;

end;
LastRegions=Regions;
Regions=(bestpoint)';
LastRegionsList=RegionsList;
RegionsList=zeros(N,N);
% defines the new seeds as all the bestpoints
for i=1:RegionNumber
    RegionsList(Regions(i),Regions(i))=1; % The RegionsList array has ones in
the diagonal when it intersects the ranks of the seeds in the data array

end;
loops=loops+1;
end;

% Recording (and counting) all significant regions including more than
SizeLimit axes
for i=1:RegionNumber % filling the SignifRegions array
    if (threshold(i,1)>=ThresholdRatio)&&
(sum(LastRegionsList(:,LastRegions(1,i)))>=SizeLimit)
                k=1;
                for m=1:1:N
                    if LastRegionsList(m,LastRegions(1,i))~=0
                        SignifRegions(k,SignifRegionsRank)=m;
                        k=k+1;
                    end;
                end;

                SignifV3(:,SignifRegionsRank)=V3Table(:,i);
                SignifThreshold(1,SignifRegionsRank)=threshold(i);
                SignifThreshold(3,SignifRegionsRank)=k-1;
                SignifRegionsRank=SignifRegionsRank+1;
    end;

end;



% Optimisation module
SumThreshold = sum(threshold);
ProdThreshold = cumprod(threshold);

% computes the number of axes belonging to regions with threshold > 1
% (i.e. 5% significant regions)
NbSignifPoints=0;
for r=1:1:RegionNumber
    if threshold(r,1)>=ThresholdRatio
        NbSignifPoints=NbSignifPoints +
sum(LastRegionsList(:,LastRegions(1,r)));
    end;
end;

% here the criterium is NbSignifPoints (could be SumThreshold or
% ProdThreshold or any other criterium)
if NbSignifPoints>maximum % alternatively :
ProdThreshold(RegionNumber,1)>maximum etc...
    BestRegionsMembers(:,:)=0;
    maximum=NbSignifPoints; %ProdThreshold(RegionNumber,1);
    BestProd = ProdThreshold(RegionNumber,1);
    BestSum = SumThreshold;
    BestRegions=LastRegions;
    BestList=LastRegionsList;
    for j=1:1:RegionNumber
        k=1;
        for i=1:1:N
            if BestList(i,BestRegions(1,j))~=0
                BestRegionsMembers(k,j)=i;
                k=k+1;
            end;
        end;
    end;
    for j=1:1:RegionNumber

        for i=2:1:N
            if BestRegionsMembers(i,j)<BestRegionsMembers(i-1,j)
               for k=i:1:N
               BestRegionsMembers(k,j)=0;
               end;
            end;
        end;
    end;
    BestThreshold=threshold;
    BestV3Table=V3Table;
    BestNbSignifPoints=NbSignifPoints;
    BestIteration=iter;
end;
iter=iter+1;
end;

% counts the number of appearances of the various regions along successive
% iterations
for t=1:size(SignifThreshold,2)
    value=SignifThreshold(1,t);
    for u=1:size(SignifThreshold,2)
        if SignifThreshold(1,u)==value
           SignifThreshold(2,t)= SignifThreshold(2,t)+1;
        end;
    end;
end;
SignifThreshold=cat(1,SignifThreshold,SignifV3,SignifRegions);
A=SignifThreshold';
SignifRegionsLeagueTable=sortrows(A,-1);% sorts SRLT according to descending
order of the threshold value
line=2;
while ((SignifRegionsLeagueTable(line,1)~=0) ||
(line~=size(SignifRegionsLeagueTable,1)))
   if SignifRegionsLeagueTable(line,1)==SignifRegionsLeagueTable(line-1,1)
       SignifRegionsLeagueTable(line,:)=[]; % make sure that each region
appears only once
       line=line-1;
   end;

   line=line+1;
   if line==size(SignifRegionsLeagueTable,1)
       break;
   end;
end;
SignifRegionsLeagueTable=sortrows(SignifRegionsLeagueTable,-3);% sorts SRLT
according to descending order of nb of points


% processing the best solution (from all the iterations)
% computes E3 of each region
BestRegionsE3=zeros(3,RegionNumber);
BestRegionsD3=zeros(3,RegionNumber);

colordata=zeros(1,6);
for r=1:1:RegionNumber
    sizeregion=sum(BestList(:,BestRegions(1,r)));% computes nb of points in
each region
    regiondata=zeros(sizeregion,6);
    t=1;
    for s=1:1:N % fills a new dataset, called regiondata with position and
axis-vector of all members of the region
```

```
        if BestList(s,BestRegions(1,r))~=0
        regiondata(t,:)=data(s,:);
        t=t+1;
        end;
    end;
    % computes the eigenvectors of the regiondata dataset
    dircos=zeros(3,3);
    dircos(1,1)=sum(regiondata(:,4).^2);
    dircos(1,2)=sum(regiondata(:,4).*regiondata(:,5));
    dircos(1,3)=sum(regiondata(:,4).*regiondata(:,6));
    dircos(2,1)=dircos(1,2);
    dircos(2,2)=sum(regiondata(:,5).^2);
    dircos(2,3)=sum(regiondata(:,5).*regiondata(:,6));
    dircos(3,1)=dircos(1,3);
    dircos(3,2)=dircos(2,3);
    dircos(3,3)=sum(regiondata(:,6).^2);

    [V,D]=eig(dircos);
    BestRegionsE3(:,r)= V(:,3);
    BestRegionsD3(:,r)= [D(1,1);D(2,2);D(3,3)];
        % creates colordata, an Nx6 array assigning different colors to
        % axis-points according to their region
    regioncolordata=regiondata;
    colorcode=rand(1,3);
    if BestThreshold(r,1)<ThresholdRatio
        regioncolordata(:,4:6) = ones(sizeregion,3);
    else
        for i=1:1:sizeregion
        regioncolordata(i,4:6) = colorcode;
        % axes belonging to non-significant regions are coloured in black
%         if BestThreshold(r,1)<0.99
%             regioncolordata(i,4:6) = [0,0,0];
%         end;
        end;
    end;
    colordata=cat(1,colordata,regioncolordata);
end;

save sauvegarde;

colordata(1,:)=[];
scatter3(colordata(:,1),colordata(:,2),colordata(:,3),50,colordata(:,4:6),'fil
led');
hold on;
quiver3(data(:,1),data(:,2),data(:,3),data(:,4),data(:,5),data(:,6), 0.25);
hold off;

return;
```