

# Scale Space Cortical Morphometry Toolbox

Scale Space Cortical Morphometry Toolbox is a Matlab toolbox for local cortical surface morphometric feature (thickness, volume and surface area) extraction and scale space search based statistical analysis of the features using linear mixed effects models. The methods are described in:

1. Maxime Boucher, "Shape Analysis of Cortical Folds", PhD thesis, McGill University, 2011.
2. Zhao L, Boucher M, Rosa-Neto P and Evans AC, "Impact of scale space search on age- and gender-related changes in MRI-based cortical morphometry", Hum Brain Mapp., 34(9):2113-28, 2013.

## Installation

- Unzip the contents of ScaleSpaceCorticalMorphometry.zip into a local directory.
- Add the directory of the Scale Space Cortical Morphometry toolbox and its subfolders into Matlab path.
- Keith Worsley's SurfStat toolbox will be called by some of the functions, the user should install SurfStat before using the Scale Space Cortical Morphometry Toolbox.

Scripts in the following demo are made up for surfaces acquired using MNI CIVET pipeline (40962 vertices and 81920 triangles for a hemispheric surface), and results are saved and can be reloaded at any given stage to resume computations.

## Load Cortical Surfaces

To load registered CIVET gray/white surfaces into matlab and correct flipped triangles:

```
civet_dir='your CIVET output directory';
subjects = dir([civet_dir]);
if subjects(1).name=='.'
    subjects=subjects(3:end);
end

for i=length(subjects):-1:1
    disp(['processing subject: ',subjects(i).name]);
    current_path = [civet_dir,subjects(i).name];
    gray = dir([current_path,'/surfaces/*gray_surface_left_81920.obj']);
    white = dir([current_path,'/surfaces/*white_surface_left_81920.obj']);
    reg = dir([current_path,'/transforms/surfreg/*left_surfmap.sm']);
    [left_white(:, :, i), left_gray(:, :, i)] = correct_surface_map([current_path, ...
'/transforms/surfreg/', reg(1).name], [current_path, '/surfaces/', white(1).name], ...
[current_path, '/surfaces/', gray(1).name]);
    gray = dir([current_path, '/surfaces/*gray_surface_right_81920.obj']);
    white = dir([current_path, '/surfaces/*white_surface_right_81920.obj']);
    reg = dir([current_path, '/transforms/surfreg/*right_surfmap.sm']);
    [right_white(:, :, i), right_gray(:, :, i)] = correct_surface_map([current_path, ...
'/transforms/surfreg/', reg(1).name], [current_path, '/surfaces/', white(1).name], ...
[current_path, '/surfaces/', gray(1).name]);

end

save coords_registered left_gray right_gray left_white right_white
```

Warning, may take a while.

## Extract Surface Features (thickness, surface area and cortical volume at each vertex)

### Initialization

```
load coords_registered;  
mesh = create_tetra(7);
```

### Cortical Thickness

Use the "T-link" method:

```
left_tlink = squeeze(norm_vec(left_gray-left_white));  
right_tlink = squeeze(norm_vec(right_gray-right_white));
```

### Surface Area

Use the mixed Voronoi area around each vertex:

```
left_area=zeros(size(left_gray,3),40962);  
right_area=zeros(size(left_gray,3),40962);  
for i=1:size(left_gray,3);  
    [tmp,area] = cot_laplacian_operator(0.5*(left_gray(:,i)+left_white(:,i)),mesh(7).triangles.vertices);  
    left_area(i,:) = area;  
    [tmp,area] = cot_laplacian_operator(0.5*(right_gray(:,i)+right_white(:,i)),mesh(7).triangles.vertices);  
    right_area(i,:) = area;  
end
```

### Cortical Volume

Compute a local measure of volume with tetrahedron-based finite element method:

```
left_volume=zeros(size(left_gray,3),40962);  
right_volume=zeros(size(left_gray,3),40962);  
for i=1:size(left_gray,3);  
    coords_left = [left_white(:,i),left_gray(:,i)];  
    coords_right = [right_white(:,i),right_gray(:,i)];  
  
    tetra_left(1,:,:) = coords_left(:,mesh(7).tetra.vertices(2,:))-coords_left(:,mesh(7).tetra.vertices(1,:));  
    tetra_left(2,:,:) = coords_left(:,mesh(7).tetra.vertices(3,:))-coords_left(:,mesh(7).tetra.vertices(1,:));  
    tetra_left(3,:,:) = coords_left(:,mesh(7).tetra.vertices(4,:))-coords_left(:,mesh(7).tetra.vertices(1,:));  
  
    tetra_right(1,:,:) = coords_right(:,mesh(7).tetra.vertices(2,:))- coords_right(:,mesh(7).tetra.vertices(1,:));  
    tetra_right(2,:,:) = coords_right(:,mesh(7).tetra.vertices(3,:))-coords_right(:,mesh(7).tetra.vertices(1,:));  
    tetra_right(3,:,:) = coords_right(:,mesh(7).tetra.vertices(4,:))-coords_right(:,mesh(7).tetra.vertices(1,:));  
  
    vol_left = repmat(squeeze(max(determinant(tetra_left)/24,0))',[4,1]);  
    vol_right = repmat(squeeze(max(-determinant(tetra_right)/24,0))',[4,1]);  
  
    left_volume(i,:) = accumarray(mod(mesh(7).tetra.vertices(:)-1,length(mesh(7).coords))+1,vol_left(:));  
    right_volume(i,:) = accumarray(mod(mesh(7).tetra.vertices(:)-1,length(mesh(7).coords))+1,vol_right(:));  
end  
  
save features left_tlink right_tlink left_area right_area left_volume right_volume
```

## Compute Scale Space of a feature vector

Compute a scale space decomposition of a feature vector, such as cortical thickness, volume and surface area computed above, using the function 'scale\_space\_lin\_mod.m'.

### Input/Output

#### Input:

- model: a  $N \times p$  matrix that describe the linear model used for the analysis;
- left\_feature, right\_feature:  $N \times 40962$  feature vectors for left/right hemisphere;
- coords:  $3 \times 40962$  matrix of coordinates of the surface model for surface diffusion;
- tri:  $3 \times 81920$  matrix of triangle vertices of the surface model for surface diffusion;
- start\_scale\_t: diffusion time for the finest scale level, the equivalent FWHM can be obtained with 'fwhm.m' as  $fwhm(diffusion\_time)$ ;
- end\_scale\_t: diffusion time for the coarsest scale level;
- octave\_step: define the step for sampling the scale range in the log space, i.e. the number of scale samples in the defined scale space is  $\{[(\log(end\_scale\_t) - \log(start\_scale\_t)) / (\log(2)/octave\_step)] + 1\}$ ;
- output\_dir: directory where the results will be saved;
- prefix: prefix for the outputs' filenames;

#### Output:

- files contained in the directory 'output\_dir' contains the scale space decomposition of the studied feature vector.
  - 'prefix\_model.mat' contains the linear model ( $model$ ), number of feature type ( $k$ ), number of data samples ( $n$ ), number of vertices ( $v$ ), degree of freedom ( $df$ ), and diffusion times at each scale level ( $times$ ).
  - 'prefix\_data\_model\_xxxx.mat' contains the normalized residuals at scale level xxxx.
  - 'prefix\_lin\_coef\_xxxx.mat' contains the coefficients of the linear model at the scale level xxxx.
  - 'prefix\_sd\_xxxx.mat' contains the square roots of the sum of squares of errors at the scale space level xxxx.

#### Script:

```
model = ['a  $N \times p$  matrix that describe the linear model used for the analysis'];  
load features; % load the cortical surface features computed above.
```

```
start_scale_t = 0.1; % fwhm(0.1) = 1.0 mm  
end_scale_t = 5910; % fwhm(5910) = 256.0 mm  
Model = 1+term(model);  
octave_step = 3;  
outputdir = 'directory where the results will be saved';  
surface_model_left = SurfStatReadSurf('surf_reg_model_left.obj');  
surface_model_right = SurfStatReadSurf('surf_reg_model_right.obj');
```

```
coords = surface_model_left.coord;  
tri = double(surface_model_left.tri);  
prefix = 'left';  
scale_space_lin_mod(coords, tri, left_feature, Model, start_scale_t, end_scale_t, octave_step, output_dir,  
prefix);
```

```

coords = surface_model_right.coord;
tri = double(surface_model_right.tri);
prefix = 'right';
scale_space_lin_mod(coords,tri,right_feature,Model,start_scale_t,end_scale_t,octave_step,output_dir,
prefix);

```

## Reconstruct smoothed feature data

Reconstruct the smoothed feature vector at the scale level xxxx from the scale space decomposition results.

```

outputdir = 'directory where the results will be saved';
load([outputdir,'left_model.mat']);

load([outputdir,'left_data_model_xxxx.mat']);
load([outputdir,'left_lin_coef_xxxx.mat']);
load([outputdir,'left_sd_xxxx.mat']);
feature_left_scale_xxxx = scalar_product(standard_dev,data_model) + double(model)*lin_mod;

load([outputdir,'right_data_model_xxxx.mat']);
load([outputdir,'right_lin_coef_xxxx.mat']);
load([outputdir,'right_sd_xxxx.mat']);
feature_right_scale_xxxx = scalar_product(standard_dev,data_model) + double(model)*lin_mod;

```

## Scale Space Search based statistical analysis

### Compute the t-statistics for an effect of interest

```

outputdir = 'directory where the results will be saved';
load([outputdir,'left_model.mat']);

prefix_left = 'left';
prefix_right = 'right';

% compute resel components at each scale level.
% warning, may take a while.
resel_left = ScaleSpaceResels(outputdir,prefix_left);
resel_right = ScaleSpaceResels(outputdir,prefix_right);
resels = resel_left + resel_right;

contrast = []; % the contrast for your effect of interest
Tmap_left = ScaleSpaceT([outputdir,prefix_left],contrast,df);
Tmap_right = ScaleSpaceT([outputdir,prefix_right],contrast,df);
Tmap = [Tmap_left, Tmap_right];

```

### Compute p-values corrected with unified-p (4D RFT)

```

start_scale = 1; % index of the starting scale
end_scale = length(times); % index of the end scale
Pmap = ScaleSpaceP(resels,start_scale,end_scale,v*2,df,Tmap);
% compute critical t-threshold corresponding to p-value p
p = 0.05;
t_threshold = find_threshold(resels,start_scale,end_scale,p,df,v*2);

```

## Compute critical t-threshold correcting 4D multiple comparisons with permutation test.

Alternatively, the 4D multiple comparison correction and critical t-threshold can be obtained with non-parametric permutation test.

Warning, this may be very time-consuming :-{

```
% reconstruct the feature data at each scale level
load([outputdir,'left_model.mat']);
feature = zeros(n,v*2,end_scale);
for scale = start_scale : end_scale
    load([outputdir,'left_data_model_000',num2str(scale-1),'.mat']);
    load([outputdir,'left_lin_coef_000',num2str(scale-1),'.mat']);
    load([outputdir,'left_sd_000',num2str(scale-1),'.mat']);
    left_feature = scalar_product(standard_dev,data_model) + double(model)*lin_mod;

    load([outputdir,'right_data_model_000',num2str(scale-1),'.mat']);
    load([outputdir,'right_lin_coef_000',num2str(scale-1),'.mat']);
    load([outputdir,'right_sd_000',num2str(scale-1),'.mat']);
    right_feature = scalar_product(standard_dev,data_model) + double(model)*lin_mod;

    feature(:,scale)=[left_feature, right_feature];
end

model = double(model);

% reorder the factors, to put the factor of interest at the end of the model. For instance,
% assume the linear model contains 5 factors, and the targeted one is the 2nd, then
model = model(:,[1 3 4 5 2]);

num_permutation = 10000;
tmax_permutation = permutation_test(data,model,num_permutation);
tmax_permutation = sort(tmax_permutation,'descend');
t_threshold = tmax_permutation(500); % threshold for p=0.05
```