Clinical Neurophysiology 117 (2006) 348-368 – Appendix (Supplementary Data)

Principal components analysis of Laplacian waveforms as a generic method for identifying ERP generator patterns: I. Evaluation with auditory oddball tasks

Jürgen Kayser [a,b,*], Craig E. Tenke [a,b]

[a] *Department of Biopsychology, New York State Psychiatric Institute, New York, NY, USA*
[b] *Department of Psychiatry, College of Physicians & Surgeons of Columbia University, New York, NY, USA*

Accepted 17 August 2005

Appendix (Supplementary Data) [*]

The computation of current source density (CSD) estimates based on spherical splines (Perrin et al., 1989, 1990) may be compactly coded in high-level numerical languages, such as the widely-popular popular MatLab language (The MathWorks, Inc., 2002, MatLab Version 6.5, Release 13 [http://www.mathworks.com]). All crucial computational steps to transform surface potentials to CSD waveforms are exemplified by the MatLab routines provided below in section A2. As an initial step, any EEG montage has to be expressed in spatial coordinates of the electrode sites with respect to the underlying CSD model, which is a sphere for the Perrin et al. (1989, 1990) algorithm. In section A1, we explain a general procedure to derive these spatial locations for a unit sphere, and table the coordinates for our 31-channel EEG montage.

A1. Spatial coordinates of a unit sphere for 10-20 system EEG montages

All notations are considered for a sphere with unit radius (1.0). For Cartesian coordinates, the x-y plane is marked by the great circle combining the locations Fpz, T7, Oz, and T8, with the x-axis running through T7 (-1.0) and T8 (+1.0), the y-axis running through Oz (-1.0) and Fpz (+1.0), and the z-axis running through the origin of the x-y plane (0) and Cz (+1.0). For spherical coordinates, the angle theta denotes the rotation of the x-axis towards the y-axis (positive poles, respectively), whereas phi denotes the angular displacement from the x-y plane towards the positive pole of the z-axis.

The spherical coordinates for locations where the three great circles of the x-y, x-z, and y-z planes intersect are straightforward. As defined by the logic of the international 10-20 system (e.g., Oostenveld and Praamstra, 2001), intermediate locations on these great circles are quarter (22.5°) or quintile (18.0°) sections of the respective rectangular arcs (e.g., for Fp1, theta is 90 + 1*18 = 108; for Fz, phi is 2 * 22.5 = 45). Similarly, locations on the small circle combining nasion and inion are a quarter rectangular arc downwards (i.e., phi equals -22.5°) and on quintile sections with respect x-y plane angular displacement (e.g., for P10, theta is 2 * -18 = -36).

The remaining locations, which are less straightforward, are located on small circles defined by two lateral and one midline base locations (Oostenveld and Praamstra, 2001). For instance, F3 is located on a small circle defined by the intersection of the sphere with the plane given by the base locations F7, F8, and Fz. Origin and radius of this small circle can easily be derived from the Cartesian coordinates of the three base locations. Halving the two vectors from this origin to F7 and Fz gives a mid-vector that

[*] MatLab code available at *http://psychophysiology.cpmc.columbia.edu/CN2006appendix.html*

intersects the sphere at F3. Once F3 has been derived, all mirror locations (i.e., F4, P3, and P4) are easily determined through their x-y plane displacements. By using these intermediate locations, this systematic can easily be extended to determine the coordinates of any montage based on the 10-20 system (see Functions 1 and 2, Example 1 for site F3, and Table A1 below).

The "true" spatial location of the nose tip is obviously outside the sphere, which is in a strict sense incompatible with the assumed model. However, the inclusion of the nose reference in the EEG montage provides additional information, which is most important for determining the contributions of nearby anterior regions. As similar geometric simplifications were applied to all scalp coordinates, we modeled the effective location of the nose on the surface of the sphere as half of a quarter extension beyond nasion.

Function 1: *SphericalMidPoint.m*

```
% SphericalMidPoint - Determine a point on a unit sphere by assuming
%                     equal distance to two spherical points
%
% (published in appendix of Kayser J, Tenke CE, Clin Neurophysiol 2006;117(2):348-368)
%
% Usage: [P] = SphericalPoint( Pl, Pr, Pm, P1, P2, lab);
%
% Implementation of simple linear geometric principles and algorithms
% (e.g., see documentation at http://mathworld.wolfram.com/Plane.html
%                             http://mathworld.wolfram.com/Sphere.html
%                             http://mathworld.wolfram.com/Line.html )
%
% Input parameters:
%   Pl,Pr,Pm =  point vectors with Cartesian x,y,z coordinates defining
%               a small circle on a unit sphere, with Pl and Pr located
%               on the x-y plane, and Pm on the y-z plane
%   P1,P2    =  point vectors indicating two locations on the small circle
%   lab      =  electrode label string
%
% Output parameter:
%   P     =  point vector/matrix with Cartesian x,y,z intersection(s)
%
% Copyright (C) 2003 by Jürgen Kayser (Email: kayserj@pi.cpmc.columbia.edu)
% GNU General Public License (http://www.gnu.org/licenses/gpl.txt)
% Updated: $Date: 2005/02/09 14:00:00 $ $Author: jk $
%
function [P] = SphericalPoint( Pl, Pr, Pm, P1, P2, lab);
M = Pl-(Pl-Pr)/2';          % use lateral points to determine their midpoint
d = abs(Pr(1)-Pl(1))/2;     % ...  in x-y plane on y-axis and its distance
V = Pm - M;                 % get vector from midline point Pm and midpoint M
m = sqrt(V(2)^2 + V(3)^2);  % ... (V(1)=0) and determine the vector length m
q = (d^2 - m^2) / 2*m;      % get extension q of vector V towards origin of
r = m + q;                  % ... small circle and compute its radius r
V = V * (m+q)/m;            % extend vector V
O = Pm - V;                 % determine origin O of small circle
M1 = P1 - O;                % translate first point vector to small circle
M2 = P2 - O;                % translate second point vector to small circle
N = O + (M1-(M1-M2)/2);     % determine mean vector and translate to unit sphere
P = LineWithSphere(N,O);    % get intersection of vector O-N with unit sphere
R = 1.0;                    % set unit sphere radius
for i = 1:size(P,1)         % test whether point P is really on the surface
    s = P(i,1).^2 + ...     % ... of the sphere (s must be zero)
        P(i,2).^2 + ...
        P(i,3).^2 - R.^2;
    disp(sprintf('%5s %8.5f %8.5f %8.5f  (off surface: %20.17f)', ...
         char(lab(:)),P(i,1),P(i,2),P(i,3),s));
end;
```

Function 2: *LineWithSphere.m*

```
% LineWithSphere - Determine intersection(s) of a line defined by
%                  two points in space with a sphere
%
% (published in appendix of Kayser J, Tenke CE, Clin Neurophysiol 2006;117(2):348-368)
%
% Usage: [P] = LineWithSphere ( P1, P2, Os, r );
%
% Implementation of simple linear geometric principles and algorithms
% (e.g., see documentation at http://mathworld.wolfram.com/Plane.html
%                             http://mathworld.wolfram.com/Sphere.html
%                             http://mathworld.wolfram.com/Line.html)
%
% Input parameters:
%   P1,P2 =  point vectors with Cartesian x,y,z coordinates
%   Os    =  point vector origin of sphere (default = [0 0 0])
%   r     =  sphere radius (default = 1.0)
%
% Output parameter:
%   P     =  point vector/matrix with Cartesian x,y,z intersection(s)
%
% Copyright (C) 2003 by Jürgen Kayser (Email: kayserj@pi.cpmc.columbia.edu)
% GNU General Public License (http://www.gnu.org/licenses/gpl.txt)
% Updated: $Date: 2005/02/09 14:00:00 $ $Author: jk $
%
function [P] = LineWithSphere ( P1, P2, Os, r );
if nargin < 4;  r = 1.0;     end;    % use unit sphere (radius = 1) by default
if nargin < 3; Os = [0 0 0]; end;    % use natural origin by default
x1 = P1(1); y1 = P1(2); z1 = P1(3);  % x,y,z-coordinates for line point 1
x2 = P2(1); y2 = P2(2); z2 = P2(3);  % x,y,z-coordinates for line point 2
x3 = Os(1); y3 = Os(2); z3 = Os(3);  % x,y,z-coordinates for origin of sphere
a = (x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2;
b = 2 * ( (x2-x1)*(x1-x3) + (y2-y1)*(y1-y3) + (z2-z1)*(z1-z3) );
c = x3^2 + y3^2 + z3^2 + x1^2 + y1^2 + z1^2 - 2 * (x3*x1 + y3*y1 + z3*z1) - r^2;
T = b^2 - 4 * a * c;
if T < 0                             % no intersection
   P = [NaN NaN NaN];
   return
end
if T == 0                            % one intersection
   u = -b / (2*a);
   P(1) = x1 + u * (x2 - x1);
   P(2) = y1 + u * (y2 - y1);
   P(3) = z1 + u * (z2 - z1);
   return
end
u = (-b + sqrt(T)) / (2 * a);        % first intersection
P(1,1) = x1 + u * (x2 - x1);
P(1,2) = y1 + u * (y2 - y1);
P(1,3) = z1 + u * (z2 - z1);
u = (-b - sqrt(T)) / (2 * a);        % second intersection
P(2,1) = x1 + u * (x2 - x1);
P(2,2) = y1 + u * (y2 - y1);
P(2,3) = z1 + u * (z2 - z1);
```

Example 1: ***GetF3CoordinatesWithMirrorLocations.m***

```
% Example code to determine the spherical and Cartesian coordinates of
% four intermediate 10-20 locations (F3, F4, P3, P4) from a small circle
% given by three known 10-20 locations (F7, Fz, F8)
%
% (published in appendix of Kayser J, Tenke CE, Clin Neurophysiol 2006;117(2):348-368)
%
ThetaPhi = [144.000    0.000;               % F7      a priori known
             90.000   45.000;               % Fz      spherical coordinates
             36.000    0.000] ;             % F8      [theta phi]
ThetaPhiRad = (2 * pi * ThetaPhi) / 360;    % convert degrees to radians
[X,Y,Z] = sph2cart(ThetaPhiRad(:,1), ...    % theta [radians] is in column 1
                   ThetaPhiRad(:,2), ...    % phi [radians] is in column 2
                   1.0 );                   % use unit radius
XYZ = [X Y Z];                              % create Cartesian matrix
F7 = XYZ(1,:);                              % recover point vectors with
FZ = XYZ(2,:);                              % ... Cartesian x,y,z coordinates
F8 = XYZ(3,:);                              % ... for the three electrodes
P = SphericalMidPoint(F7,F8,FZ,F7,FZ,'F3');  % determine F3 as F7-Fz mid point
F3 = P(2,:);                                % use intersection above x-y plane
F4 = F3 .* [-1  1  1];                      % mirror x-coordinate for F4
P3 = F3 .* [ 1 -1  1];                      % mirror y-coordinate for P3
P4 = F3 .* [-1 -1  1];                      % mirror x-,y-coordinates for P4
XYZext = [XYZ; F3; F4; P3; P4];             % extend XYZ matrix with new sites
[th,ph,ra] = cart2sph(XYZext(:,1), ...      % convert Cartesian xyz coordinates
                      XYZext(:,2), ...      % ... to spherical coordinates
                      XYZext(:,3));         % ... [radians]
ThetaPhiExt = [ [th * 360 / 2 / pi] ...     % convert spherical coordinates from
                [ph * 360 / 2 / pi]];       % ... radians to degrees
Elab = ['F7';'Fz';'F8'; ...                 % create electrode label array
        'F3';'F4';'P3';'P4'];
disp(sprintf('%8s  %10s %10s %12s %12s %12s', ...
    'Site','Theta','Phi','X','Y','Z'));
for i = 1:size(Elab,1)                      % table coordinates ...
  disp(sprintf('%8s  %10.3f %10.3f %12.5f %12.5f %12.5f', ...
    char(Elab(i,:)), ...                    % ... electrode label
    ThetaPhiExt(i,1), ThetaPhiExt(i,2), ...  % ... Theta, Phi [°]
    XYZext(i,1),XYZext(i,2),XYZext(i,3)) );  % ... Cartesian x,y,z coordinates
end;
```

Table A1. Spherical and Cartesian coordinates of 31-channel montage based on 10-20 system

	Spherical Coordinates [degrees]		Cartesian Coordinates (Unit Sphere [radius = 1.0])		
Electrode Site	Theta	Phi	X	Y	Z
FP1	108.000	0.000	-0.30902	0.95106	0.00000
FP2	72.000	0.000	0.30902	0.95106	0.00000
F7	144.000	0.000	-0.80902	0.58779	0.00000
F3	129.254	29.833	-0.54891	0.67173	0.49747
Fz	90.000	45.000	0.00000	0.70711	0.70711
F4	50.746	29.833	0.54891	0.67173	0.49747
F8	36.000	0.000	0.80902	0.58779	0.00000
FT9	162.000	-22.500	-0.87866	0.28549	-0.38268
FC5	158.854	20.773	-0.87203	0.33729	0.35467
FC6	21.146	20.773	0.87203	0.33729	0.35467
FT10	18.000	-22.500	0.87866	0.28549	-0.38268
T7	180.000	0.000	-1.00000	0.00000	0.00000
C3	180.000	45.000	-0.70711	0.00000	0.70711
Cz	0.000	90.000	0.00000	0.00000	1.00000
C4	0.000	45.000	0.70711	0.00000	0.70711
T8	0.000	0.000	1.00000	0.00000	0.00000
TP9	-162.000	-22.500	-0.87866	-0.28549	-0.38268
CP5	-158.854	20.773	-0.87203	-0.33729	0.35467
CP6	-21.146	20.773	0.87203	-0.33729	0.35467
TP10	-18.000	-22.500	0.87866	-0.28549	-0.38268
P9	-144.000	-22.500	-0.74743	-0.54304	-0.38268
P7	-144.000	0.000	-0.80902	-0.58779	0.00000
P3	-129.254	29.833	-0.54891	-0.67173	0.49747
Pz	-90.000	45.000	0.00000	-0.70711	0.70711
P4	-50.746	29.833	0.54891	-0.67173	0.49747
P8	-36.000	0.000	0.80902	-0.58779	0.00000
P10	-36.000	-22.500	0.74743	-0.54304	-0.38268
O1	-108.000	0.000	-0.30902	-0.95106	0.00000
Oz	-90.000	0.000	0.00000	-1.00000	0.00000
O2	-72.000	0.000	0.30902	-0.95106	0.00000
Nose	90.000	-33.750	0.00000	0.83147	-0.55557

A2. Scalp current density estimates using spherical spline interpolation

Given a complete EEG montage with spherical coordinates (i.e., angles theta and phi) for all recording sites, the computation of scalp current density estimates is exemplified by the two MatLab functions listed below. Two ASCII input files describing the EEG montage (e.g., consisting of the data rows in Table A1) and providing the recorded ERP data (i.e., arranging rows and columns as an electrodes-by-samples matrix) with an identical sequence of electrodes are needed. The initial iterative computations for the Legendre polynomial, the interelectrode cosine distances matrix, and g- and h-functions (Perrin et al., 1989) are rather time consuming (and more efficiently executed when using a compiler language), but need to be performed only once for any given montage. The actual sample-by-sample CSD transformations using these initial computations are very fast, even in an interpreter language. It should be noted that the last code line in Function 3 (*GetCSD.m*) evokes Function 4 (*CSD.m*) with implicit default settings for the smoothing constant (10^{-5}) and a head radius matching a unit sphere (1.0). The latter default provides larger and therefore more practical CSD values ($\mu V/m^2$) compared with the smaller CSD values based on a more realistic head radius (10.0 cm; $\mu V/cm^2$).

Function 3: ***GetCSD.m***

```
% GetCSD - Compute Current Source Density (CSD) estimates using the spherical spline
%          surface Laplacian algorithm suggested by Perrin et al. (1989, 1990)
%
% (published in appendix of Kayser J, Tenke CE, Clin Neurophysiol 2006;117(2):348-368)
%
% Usage: [CE] = GetCSD(EEGmont, ERPdata);
%
% Implementation of algorithms described by Perrin, Pernier, Bertrand, and
% Echallier in Electroenceph Clin Neurophysiol 1989;72(2):184-187, and
% Corrigenda EEG 02274 in Electroenceph Clin Neurophysiol 1990;76:565.
%
% Input parameters:
%   EEGmont = name of EEG montage spherical coordinates file in ASCII format,
%             with rows consisting of seven columns: electrode label, two
%             spherical angles (theta, phi), and Cartesian coordinates x,y,z
%   ERPdata = name of ERP data file in ASCII format stored as
%             electrodes-by-samples (rows-by-columns) matrix
%
% Output parameter:
%        CE = CSD estimates as electrodes-by-samples matrix
%
% Copyright (C) 2003 by Jürgen Kayser (Email: kayserj@pi.cpmc.columbia.edu)
% GNU General Public License (http://www.gnu.org/licenses/gpl.txt)
% Updated: $Date: 2005/02/11 14:00:00 $ $Author: jk $
%
function [CE] = GetCSD(EEGmont, ERPdata)
ERP = load(ERPdata);                    % load electrodes-by-samples matrix
[Elab,Theta,Phi,X,Y,Z] = ...            % read EEG montage
   textread(EEGmont,'%s%f%f%f%f%f');
ThetaRad = (2 * pi * Theta) / 360;      % convert Theta and Phi to radians ...
PhiRad = (2 * pi * Phi) / 360;          % ... and Cartesian coordinates ...
[X,Y,Z] = sph2cart(ThetaRad,PhiRad,1.0); % ... for optimal resolution
nElec = length(Elab);                   % determine size of EEG montage
EF(nElec,nElec) = 0;                    % initialize interelectrode matrix ...
for i = 1:nElec; for j = 1:nElec;       % ... and compute all cosine distances
  EF(i,j) = 1 - ( ( (X(i) - X(j))^2 + ...
    (Y(i) - Y(j))^2 + (Z(i) - Z(j))^2 ) / 2 );
end; end;
m = 4; N = 50;                          % set m constant and N iterations
G(nElec,nElec) = 0; H(nElec,nElec) = 0;  % claim memory for G- and H-matrices
for i = 1:nElec; for j = 1:nElec;
  P = zeros(N);                         % compute Legendre polynomial
  for n = 1:N;
    p = legendre(n,EF(i,j));
    P(n) = p(1);
  end;
  g = 0.0; h = 0.0;                     % compute h- and g-functions
  for n = 1:N;
    g = g + ( (( 2.0*n+1.0) * P(n)) / ((n*n+n)^m    ) );
    h = h + ( ((-2.0*n-1.0) * P(n)) / ((n*n+n)^(m-1)) );
  end;
  G(i,j) =  g / 4.0 / pi;               % finalize cell of G-matrix
  H(i,j) = -h / 4.0 / pi;               % finalize cell of H-matrix
end; end;
CE = CSD(ERP, G, H);                    % compute CSD for ERP data
```

Function 4: *CSD.m*

```
% CSD - Current Source Density (CSD) transformation based on spherical spline
%       surface Laplacian as suggested by Perrin et al. (1989, 1990)
%
% (published in appendix of Kayser J, Tenke CE, Clin Neurophysiol 2006;117(2):348-368)
%
% Usage: [X, Y] = CSD(Data, G, H, lamb, head);
%
% Implementation of algorithms described by Perrin, Pernier, Bertrand, and
% Echallier in Electroenceph Clin Neurophysiol 1989;72(2):184-187, and
% Corrigenda EEG 02274 in Electroenceph Clin Neurophysiol 1990;76:565.
%
% Input parameters:
%   Data = surface potential electrodes-by-samples matrix
%      G = g-function electrodes-by-electrodes matrix
%      H = h-function electrodes-by-electrodes matrix
%   lamb = smoothing constant lambda (default = 1.0e-5)
%   head = head radius (default = no value for unit sphere [µV/m²])
%          specify a value [cm] to rescale CSD data to smaller units [µV/cm²]
%          (e.g., use 10.0 to scale to more realistic head size)
%
% Output parameter:
%      X = current source density (CSD) transform electrodes-by-samples matrix
%      Y = spherical spline surface potential (SP) interpolation electrodes-
%          by-samples matrix (only if requested)
%
% Copyright (C) 2003 by Jürgen Kayser (Email: kayserj@pi.cpmc.columbia.edu)
% GNU General Public License (http://www.gnu.org/licenses/gpl.txt)
% Updated: $Date: 2005/02/11 14:00:00 $ $Author: jk $
%        - code compression and comments
% Updated: $Date: 2007/02/07 11:30:00 $ $Author: jk $
%        - recommented rescaling (unit sphere [µV/m²] to realistic head size [µV/cm²])
%
function [X, Y] = CSD(Data, G, H, lamb, head)
[nElec,nPnts] = size(Data);             % get matrix dimensions
mu = mean(Data);                        % get grand mean
Z = (Data - repmat(mu,nElec,1));        % compute average reference
Y = G;  X = H;                          % claim memory for output matrices
if nargin < 5; head = 1.0; end;         % initialize scaling variable [µV/m²]
head = head * head;                     % or rescale data to head sphere [µV/cm²]
if nargin < 4; lamb = 1.0e-5; end;      % initialize smoothing constant
for e = 1:size(G,1);                    % add smoothing constant to diagonale
  G(e,e) = G(e,e) + lamb;
end;
Gi = inv(G);                            % compute G inverse
for i = 1:size(Gi,1);                   % compute sums for each row
  TC(i) = sum(Gi(i,:));
end;
sgi = sum(TC);                          % compute sum total
for p = 1:nPnts
  Cp = Gi * Z(:,p);                     % compute preliminary C vector
  c0 = sum(Cp) / sgi;                   % common constant across electrodes
  C = Cp - (c0 * TC');                  % compute final C vector
  for e = 1:nElec;                      % compute all CSDs ...
    X(e,p) = sum(C .* H(e,:)') / head; % ... and scale to head size
  end;
  if nargout > 1; for e = 1:nElec;      % if requested ...
    Y(e,p) = c0 + sum(C .* G(e,:)');    % ... compute all SPs
  end; end;
end;
```