Supplementary Material

Matrix-based formulation of the iterative randomized stimulation and averaging (IRSA) method for recording evoked potentials

Angel de la Torre[1], Joaquin T. Valderrama[2,3], Jose C. Segura[1], and Isaac M. Alvarez[1]

[1]*Department of Signal Theory, Telematics and Communications, University of Granada, Granada, Spain.*
[2]*National Acoustic Laboratories, Sydney, Australia.*
[3]*Department of Linguistics, Maquarie University, Sydney, Australia.*
E-mail: atv@ugr.es (Angel de la Torre); joaquin.valderrama@nal.gov.au (Joaquin T. Valderrama); segura@ugr.es (Jose C. Segura); isamaru@ugr.es (Isaac M. Alvarez)

Contents

1 Code for conventional IRSA algorithm

The following MatLab / Octave code process the electroencephalogram and iteratively estimates the evoked response with the conventional IRSA algorithm.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [xi,t_run,t_iter] = IRSA_convent(y,m,I,J,alpha,OUTPUT)
% IRSA conventional: iterative with synchronized average
%  Input parameters:  y (Recorded EEG)
%                     m (Trigger vector)
%                     I (Number of iterations)
%                     J (Length of the averaging window in samples)
%                     alpha (Convergence-control parameter)
%                     OUTPUT (flag for presenting results at iterations)
%  Output parameters: xi (AEP estimate)
%                     t_run (time required for algorithm execution)
%                     t_iter (time required for each iteration)
% Angel de la Torre, Jose Carlos Segura, Joaquin Valderrama 2018
%     University of Granada (Spain)
%     National Acoustic Laboratories, Macquarie University (Australia)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xi,t_run,t_iter] = IRSA_convent(y,m,I,J,alpha,OUTPUT)
% Initialization
tic;                  % time-stamp beginning of function
K = length(m);        % Number of stimuli
xi = zeros(J,1);      % AEP initialization
z = zeros(J,1);                    % Residual initialization
for k=1:K                          % Loop for stimuli
    Sweep = m(k):(m(k)+J-1);       % Sweep selection
    z = z + y(Sweep)/K;            % Average of EEG
end
% Iterations
t_iter=toc;           % time-stamp for iterations
for i=1:I             % loop for iterations
    xi = xi+alpha*z;                   % AEP estimate
    % Residual EEG
    r = y;                             % Residual initialization
    for k=1:K                          % Loop for stimuli
        Sweep = m(k):(m(k)+J-1);       % Sweep selection
        r(Sweep) = r(Sweep)-xi;        % Removes all AEPs from y(n) to obtain residual
    end
    % Averaged residual estimate
    z = zeros(J,1);                    % Residual initialization
    for k=1:K                          % Loop for stimuli
        Sweep = m(k):(m(k)+J-1);       % Sweep selection
        z = z + r(Sweep)/K;            % Average residual estimation
    end
    if OUTPUT==1
        fprintf('Iteration %d:   res: %.16f\n',i,std(z));
    end
end
t_run=toc;                % total execution time
t_iter=(t_run-t_iter)/I; % execution time for each iteration
return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2 Code for matrix IRSA algorithm

The following MatLab / Octave code process the electroencephalogram and iteratively estimates the evoked response with the matrix IRSA algorithm.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [xi ,t_run,t_iter] = IRSA_matrix(y,m,I,J,alpha,OUTPUT)
% IRSA, matrix implementation:
%  Input parameters:  y (Recorded EEG)
%                     m (Trigger vector)
%                     I (Number of iterations)
%                     J (Length of the averaging window in samples)
%                     alpha (Convergence-control parameter)
%                     OUTPUT (flag for presenting results at iterations)
%  Output parameters: xi (AEP estimate)
%                     t_run (time required for algorithm execution)
%                     t_iter (time required for each iteration)
% Angel de la Torre, Jose Carlos Segura, Joaquin Valderrama 2018
%     University of Granada (Spain)
%     National Acoustic Laboratories, Macquarie University (Australia)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xi ,t_run,t_iter] = IRSA_matrix(y,m,I,J,alpha,OUTPUT)
% Initialization
tic;                % time-stamp beginning of function
% Initialization of z0 and x0
s=zeros(size(y));   % stimulation signal
s(m)=1;
Es=sum(s.*s);       % energy of the stimulation signal (number of stimuli)
z_tmp=xcorr(y,s,J-1)/Es; % cross-correlation between EEG and stim. signal
z0=z_tmp(J:end);    % z0 is the first averaged response
xi=zeros(J,1);      % initial estimation of the response
zi=z0;
% Initialization of autocorrelation matrix
rs=xcorr(s,J)/Es;   % autocorrelation function of the stimulation signal
Rs=zeros(J,J);      % autocorrelation matrix
for i=1:J
    j=1:J; idx=j-i+J+1;
    Rs(i,j)=rs(idx); % autocorrelation matrix
end
% Iterations
t_iter=toc;         % time-stamp for iterations
for i=1:I           % loop for iterations
    xi=xi+alpha*zi;  % AEP estimate
    zi=z0-Rs*xi;     % Average residual estimation
    if OUTPUT==1
        fprintf('Iteration %d:   res: %.16f\n',i,std(zi));
    end
end
t_run=toc;          % total execution time
t_iter=(t_run-t_iter)/I; % execution time for each iteration
return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3 IRSA solution expressed as a geometric series

The IRSA estimation of the evoked response can be expressed, using a geometric series of matrices, as:

$$\hat{\mathbf{x}}_i = \alpha \left(\sum_{j=0}^{i-1} (I - \alpha R_s)^j \right) \mathbf{z}_0 \tag{1}$$

where α is a convergence parameter (with $0 < \alpha \leq 1$), I is the $J \times J$ identity matrix, R_s is the $J \times J$ autocorrelation matrix of the stimulation sequence and $\mathbf{z}_0$ is the averaged residual estimation of the EEG at initialization (i.e., the averaged EEG).

This section provides a revision on geometric series and geometric series of matrices, and also the application of the results to the IRSA solution. This section can be useful for those not familiar with the mathematics related to geometric series and geometric series of matrices.

3.1 Solution of the geometric series

Let a be a real constant. The geometric series of ratio a is defined as:

$$S_i = 1 + a + a^2 + \ldots + a^i = \sum_{j=0}^{i} a^j \tag{2}$$

The derivation of the solution is obtained by subtracting $S_i - aS_i$:

$$aS_i = a + a^2 + a^3 + \ldots + a^{i+1} \tag{3}$$

$$S_i - aS_i = 1 - a^{i+1} \tag{4}$$

or equivalently:

$$(1 - a)S_i = 1 - a^{i+1} \tag{5}$$

and from this equation, the solution of the series is obtained as:

$$S_i = \frac{1 - a^{i+1}}{1 - a} \tag{6}$$

or, using the definition of the series:

$$\sum_{j=0}^{i} a^j = \frac{1 - a^{i+1}}{1 - a} \tag{7}$$

It should be noted that this expression is not valid if $a = 1$, because in that case the series has solution ($S_i = i + 1$) but the right side of the previous equation is an indeterminate.

3.2 Convergence of the geometric series

If the absolute value of the ratio a is smaller than 1 (i.e. if $|a| < 1$), the term a^{i+1} converges to zero, and therefore the geometric series converges to:

$$S_\infty = \lim_{i \to \infty} \sum_{j=0}^{i} a^j = \frac{1}{1 - a} \tag{8}$$

3.3 Solution of the geometric series of matrices

Let A be a $J \times J$ square matrix of real numbers. The corresponding geometric series of matrices is defined as:

$$S_i = I + A + A^2 + \ldots + A^i = \sum_{j=0}^{i} A^j \tag{9}$$

where the exponent j applied to the matrix means a matrix multiplication of A (repeated j times), and I is the $J \times J$ identity matrix. The series can be multiplied by the matrix A:

$$AS_i = S_i A = A + A^2 + \ldots + A^{i+1} \tag{10}$$

and the difference $S_i - AS_i = S_i - S_iA$ is:

$$S_i - AS_i = S_i - S_iA = I - A^{i+1} \tag{11}$$

or, equivalently:

$$(I - A)S_i = S_i(I - A) = I - A^{i+1} \tag{12}$$

and if $(I - A)$ is non singular (and can therefore be inverted), the equation can be multiplied by $(I - A)^{-1}$ at the left:

$$(I - A)^{-1}(I - A)S_i = IS_i = S_i = (I - A)^{-1}(I - A^{i+1}) \tag{13}$$

or at the right:

$$S_i(I - A)(I - A)^{-1} = S_iI = S_i = (I - A^{i+1})(I - A)^{-1} \tag{14}$$

resulting in the solution of the geometric series of matrices:

$$S_i = (I - A)^{-1}(I - A^{i+1}) = (I - A^{i+1})(I - A)^{-1} \tag{15}$$

or, using the definition of the series:

$$\sum_{j=0}^{i} A^j = (I - A)^{-1}(I - A^{i+1}) = (I - A^{i+1})(I - A)^{-1} \tag{16}$$

where it should be noted that the solution requires that $(I - A)$ is non singular.

3.4 Convergence of the geometric series of matrices

If the absolute values of all the eigenvalues of A are smaller than 1, then the term A^{i+1} converges to the null matrix and therefore the geometric series converges:

$$S_\infty = \lim_{i\to\infty} \sum_{j=0}^{i} A^j = (I - A)^{-1} \tag{17}$$

3.5 Application to the IRSA algorithm

According to the previous result, the geometric series of matrices involved in the IRSA solution can be expressed as:

$$\sum_{j=0}^{i-1} (I - \alpha R_s)^j = \left(I - (I - \alpha R_s)^i\right) \left(I - (I - \alpha R_s)\right)^{-1} = \left(I - (I - \alpha R_s)^i\right) (\alpha R_s)^{-1} \tag{18}$$

which requires that the matrix R_s is non singular (and can be inverted). With this result, the IRSA solution provided by the geometric series is:

$$\hat{\mathbf{x}}_i = \alpha \left(\sum_{j=0}^{i-1} (I - \alpha R_s)^j \right) \mathbf{z}_0 = \alpha \left(I - (I - \alpha R_s)^i\right) (\alpha R_s)^{-1} \mathbf{z}_0 = \left(I - (I - \alpha R_s)^i\right) R_s^{-1} \mathbf{z}_0 \tag{19}$$

If the absolute values of the eigenvalues of $(I - \alpha R_s)$ are smaller than 1, the term $(I - \alpha R_s)^i$ converges to the null matrix and therefore the IRSA solution converges to:

$$\hat{\mathbf{x}}_\infty = \lim_{i\to\infty} \hat{\mathbf{x}}_i = R_s^{-1} \mathbf{z}_0 \tag{20}$$

It can be noted that the convergence condition can be expressed in terms of the eigenvalues of R_s. Let μ_j (with $j = 1, \ldots, J$) be the eigenvalues of the autocorrelation matrix R_s. The eigenvalues of $(I - \alpha R_s)$ are $(1 - \alpha\mu_j)$. Since $\mu_j \geq 0$ (because autocorrelation matrices are Toeplitz-symmetric positive semi-definite matrices), and $\alpha > 0$, the condition of convergence can be written as:

$$1 - \alpha\mu_i > -1 \qquad \forall \quad j = 1, \ldots, J \tag{21}$$

or equivalently:

$$\alpha < \frac{2}{\mu_i} \qquad \forall \quad j = 1, \ldots, J \tag{22}$$

or:

$$\alpha < \frac{2}{\max\left(\mu_i\right)} \tag{23}$$

i.e. the convergence parameter should be small enough in order to avoid oscillations of the series, and the risk of oscillations increases when the autocorrelation matrix contains large eigenvalues (i.e. when the stimulation sequence contains resonances). This condition provides a well defined criterion to select a small enough convergence parameter.

On the other hand, the solution of the geometric series at convergence does not depends on the α value (a very small α would require more iterations, but the series converges to the same solution), and therefore, if we are concerned with the IRSA solution at convergence, the convergence parameter is irrelevant.

4 Code for matrix IRSA algorithm, optimized implementation of initialization

The following MatLab / Octave code process the electroencephalogram and iteratively estimates the evoked response with the matrix IRSA algorithm with an optimized implementation of the initialization.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [xi ,t_run,t_iter] = IRSA_matrix_opt(y,m,I,J,alpha,OUTPUT)
% IRSA, matrix implementation. Initialization optimized:
%  Input parameters:  y (Recorded EEG)
%                     m (Trigger vector)
%                     I (Number of iterations)
%                     J (Length of the averaging window in samples)
%                     alpha (Convergence-control parameter)
%                     OUTPUT (flag for presenting results at iterations)
%  Output parameters: xi (AEP estimate)
%                     t_run (time required for algorithm execution)
%                     t_iter (time required for each iteration)
% Angel de la Torre, Jose Carlos Segura, Joaquin Valderrama 2018
%     University of Granada (Spain)
%     National Acoustic Laboratories, Macquarie University (Australia)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xi ,t_run,t_iter] = IRSA_matrix_opt(y,m,I,J,alpha,OUTPUT)
% Initialization
tic;                % time-stamp beginning of function
Es=length(m);       % energy of the stimulation signal (number of stimuli)
z0=zeros(J,1); rs0=zeros(J,1); s=zeros(size(y)); Rs=zeros(J,J); xi=zeros(J,1);
s(m)=1;             % stimulation signal
for j=1:J
    idx=j+m-1;
    z0(j)=sum(y(idx));  % cross-corr between EEG and stim. signal
    rs0(j)=sum(s(idx)); % autocorrelation of stim. signal
end
z0=z0/Es; zi=z0;        % first averaged response
rs0=rs0/Es;             % normalized autocorrelation stim. signal
for i=1:J
    j=1:J; idx=abs(j-i)+1;
    Rs(i,j)=rs0(idx); % autocorrelation matrix
end
% Iterations
t_iter=toc;         % time-stamp for iterations
for i=1:I           % loop for iterations
    xi=xi+alpha*zi;  % AEP estimate
    zi=z0-Rs*xi;     % Average residual estimation
    if OUTPUT==1
        fprintf('Iteration %d:   res: %.16f\n',i,std(zi));
    end
end
t_run=toc;          % total execution time
t_iter=(t_run-t_iter)/I; % execution time for each iteration
return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5 Matrix product involving a Toeplitz matrix in the frequency domain

5.1 Toeplitz matrices

Let T be a $J \times J$ Toeplitz matrix:

$$T = \begin{pmatrix} T_{0,0} & T_{0,1} & T_{0,2} & \cdots & T_{0,J-1} \\ T_{1,0} & T_{1,1} & T_{1,2} & \cdots & T_{1,J-1} \\ t_{2,0} & t_{2,1} & T_{2,2} & \cdots & T_{2,J-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ T_{J-1,0} & T_{J-1,1} & T_{J-1,2} & \cdots & T_{J-1,J-1} \end{pmatrix} = \begin{pmatrix} t_0 & t_1 & t_2 & \cdots & t_{J-1} \\ t_{-1} & t_0 & t_1 & \cdots & t_{J-2} \\ t_{-2} & t_{-1} & t_0 & \cdots & t_{J-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{-J+1} & t_{-J+2} & t_{-J+3} & \cdots & t_0 \end{pmatrix} \tag{24}$$

that is, a matrix verifying that $T_{j_1,j_2} = t_{j_2-j_1}$, and let $\mathbf{x}$ be a J-component column vector:

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{J-1} \end{pmatrix} \tag{25}$$

The conventional matrix product $\mathbf{y} = T\mathbf{x}$ involves $J \times J$ products (with complexity $\mathcal{O}(J^2)$) . However, due to the properties of Toeplitz matrices, a Fast Fourier Transform (FFT) based implementation, with complexity $\mathcal{O}(J \log_2(J))$ is possible, which is very useful, particularly for large dimensionality. In this section we describe how the product $\mathbf{y} = T\mathbf{x}$ can be performed with a FFT-based implementation.

Since Toeplitz matrices represent convolutional processes, and due to the properties of the Fourier Transform (FT) regarding convolution (FT transforms convolution into algebraic product, or, equivalently, FT diagonalizes convolutions), matrix multiplication involving a Toeplitz matrix can be implemented in the frequency domain.

However, the product $\mathbf{y} = T\mathbf{x}$ is not a simple convolution, but a truncated convolution, i.e. a convolution of the discrete time signals t_j (with $j = -J+1, \ldots, J-1$) and x_j (with $j = 0, \ldots, J-1$) restricted to the samples y_j with $j = 0, \ldots, J-1$ (a complete convolution would provide non-null samples in the range $j = -J+1, \ldots, 2J-2$). In other words, FT (in this case discrete time Fourier Transform, because it is applied to discrete time signals) diagonalizes infinite Toeplitz matrices but not truncated (or finite) Toeplitz matrices (because truncation in time domain is equivalent to multiplying with zero some samples in the time domain, which is a diagonal operation in the time domain but not in the frequency domain).

5.2 Circulant matrices

The situation is different in the case of circulant matrices. A circulant matrix is a special case of Toeplitz matrix verifying that each row is similar to the previous one except for a circulant right shift of its elements. A circulant matrix C verifies that:

$$C = \begin{pmatrix} c_0 & c_1 & c_2 & \cdots & c_{J-1} \\ c_{J-1} & c_0 & c_1 & \cdots & c_{J-2} \\ c_{J-2} & c_{J-1} & c_0 & \cdots & c_{J-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_1 & c_2 & c_3 & \cdots & c_0 \end{pmatrix} \tag{26}$$

and the matrix product involving a circulant matrix ($\mathbf{y} = C\mathbf{x}$) is equivalent to a circular convolution.

A key property of circulant matrices is that they are diagonalized by the Discrete Fourier Transform (DFT): the eigenvectors of whatever circulant matrix C are the DFT modes, and the eigenvalues are the DFT components ($\mathbf{C} = \{C(j), j = 0, \ldots, J-1\} = \mathcal{DFT}(\mathbf{c})$) of the first row in the circulant matrix ($\mathbf{c} = \{c_j, j = 0, \ldots, J-1\}$).

This property simplifies the algebra procedures involving circulant matrices. For example the matrix product $\mathbf{y} = C\mathbf{x}$ can be easily performed in the frequency domain with this procedure:

1. Fourier Transform of the involved signals: $\mathbf{C} = \mathcal{DFT}(\mathbf{c})$; $\mathbf{X} = \mathcal{DFT}(\mathbf{x})$;
2. Product of each component in the frequency domain: $Y(j) = C(j) \cdot X(j)$ (with $j = 0, \ldots, J-1$);
3. Inverse Fourier Transform of the product: $\mathbf{y} = \mathcal{IDFT}(\mathbf{Y})$.

Similarly, matrix inversion of a circulant matrix becomes trivial with the DFT:

1. Fourier Transform of the first row $\mathbf{C} = \mathcal{DFT}(\mathbf{c})$;
2. Inverse of each component in the frequency domain: $A(j) = 1/C(j)$;
3. Inverse Fourier Transform of the inverted frequency components: $\mathbf{a} = \mathcal{IDFT}(\mathbf{A})$;
4. The circulant matrix A generated from the elements in $\mathbf{a}$ verifies that $AC = CA = I$ (or equivalently $A = C^{-1}$).

where the matrix inversion requires that all the frequency components $C(j)$ are non-null.

Usually, the DFT is implemented with the more efficient Fast Fourier Transform (FFT) algorithm, and FFT or Inverse FFT (IFFT) are used instead of DFT or IDFT. Additionally, when the involved signals are real, the IFFT should provide real signals (i.e. signals with null imaginary part), but, due to the limited numerical precision of the algorithm implementation, the imaginary part is usually very small but not null, and therefore in these procedures the imaginary part is usually discarded after the IFFT.

5.3 Circulant extension of a Toeplitz matrix

Let T be the $J \times J$ Toeplitz matrix generated by the sequence t_j (with $j = -J+1, \ldots, J-1$):

$$T = \begin{pmatrix} t_0 & t_1 & t_2 & \cdots & t_{J-1} \\ t_{-1} & t_0 & t_1 & \cdots & t_{J-2} \\ t_{-2} & t_{-1} & t_0 & \cdots & t_{J-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{-J+1} & t_{-J+2} & t_{-J+3} & \cdots & t_0 \end{pmatrix} \tag{27}$$

Using the elements of the generating sequence t_j, a new $2J$-length sequence c_j can be prepared:

$$\mathbf{c} = \{c_j, j = 0, \ldots, 2J-1\} = \{t_0, t_1, t_2, \ldots, t_{J-1}, 0, t_{-J+1}, t_{-J+2}, \ldots, t_{-2}, t_{-1}\} \tag{28}$$

and this sequence can be used to generate a $2J \times 2J$ circulant matrix C which is the circulant extension of the Toeplitz matrix T:

$$C = \left(\begin{array}{ccccc|ccccc} t_0 & t_1 & t_2 & \cdots & t_{J-1} & 0 & t_{-J+1} & t_{-J+2} & \ldots & t_{-1} \\ t_{-1} & t_0 & t_1 & \cdots & t_{J-2} & t_{J-1} & 0 & t_{-J+1} & \ldots & t_{-2} \\ t_{-2} & t_{-1} & t_0 & \cdots & t_{J-3} & t_{J-2} & t_{J-1} & 0 & \ldots & t_{-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{-J+1} & t_{-J+2} & t_{-J+3} & \cdots & t_0 & t_1 & t_2 & t_3 & \ldots & 0 \\ \hline 0 & t_{-J+1} & t_{-J+2} & \ldots & t_{-1} & t_0 & t_1 & t_2 & \cdots & t_{J-1} \\ t_{J-1} & 0 & t_{-J+1} & \ldots & t_{-2} & t_{-1} & t_0 & t_1 & \cdots & t_{J-2} \\ t_{J-2} & t_{J-1} & 0 & \ldots & t_{-3} & t_{-2} & t_{-1} & t_0 & \cdots & t_{J-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ t_1 & t_2 & t_3 & \ldots & 0 & t_{-J+1} & t_{-J+2} & t_{-J+3} & \cdots & t_0 \end{array}\right) \tag{29}$$

which can be written as a composition of the matrices T and S:

$$C = \left(\begin{array}{c|c} T & S \\ \hline S & T \end{array}\right) \tag{30}$$

where the matrix S is a Toeplitz matrix defined as:

$$S = \begin{pmatrix} 0 & t_{-J+1} & t_{-J+2} & \ldots & t_{-1} \\ t_{J-1} & 0 & t_{-J+1} & \ldots & t_{-2} \\ t_{J-2} & t_{J-1} & 0 & \ldots & t_{-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_1 & t_2 & t_3 & \ldots & 0 \end{pmatrix} \tag{31}$$

5.4 Matrix-vector multiplication with Toeplitz matrices using FFT

Let T be a $J \times J$ Toeplitz matrix, and $\mathbf{x}$ a J-component column vector. Given an arbitrary $J \times J$ matrix A, the matrix product $\mathbf{y} = T\mathbf{x}$ verifies that:

$$\mathbf{y} = T\mathbf{x} = \begin{pmatrix} I_J & O_J \end{pmatrix} \begin{pmatrix} T & A \\ A & T \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{o}_J \end{pmatrix} \tag{32}$$

where $\mathbf{o}_J$, I_J and O_J are, respectively, the J-component null vector, the $J \times J$ identity matrix and the $J \times J$ null matrix. This result is useful in the particular case when the extended Toeplitz matrix becomes a circulant matrix, i.e., when the matrix A is the matrix S in equation (31):

$$\mathbf{y} = T\mathbf{x} = \begin{pmatrix} I_J & O_J \end{pmatrix} \begin{pmatrix} T & S \\ S & T \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{o}_J \end{pmatrix} = \begin{pmatrix} I_J & O_J \end{pmatrix} C \begin{pmatrix} \mathbf{x} \\ \mathbf{o}_J \end{pmatrix} \tag{33}$$

In this case, since C is circulant, it is diagonalized by DFT (or FFT). This provides an efficient procedure for computing the matrix product $\mathbf{y} = T\mathbf{x}$ for Toeplitz matrices T:

1. Preparing the $2J$-length circulant sequence: $\mathbf{c} = \{t_0, t_1, t_2, \ldots, t_{J-1}, 0, t_{-J+1}, t_{-J+2}, \ldots, t_{-2}, t_{-1}\}$
2. Zero-padding the vector to be multiplied: $\mathbf{x}_e = \{x_0, x_1, x_2, \ldots, x_{J-1}, 0, 0, 0, \ldots, 0\}$, with length $2J$.
3. Computing the FFT for both sequences: $\mathbf{C} = \mathcal{FFT}(\mathbf{c})$; $\mathbf{X}_e = \mathcal{FFT}(\mathbf{x}_e)$.
4. Multiplying both FFTs to obtain the FFT of the extended result: $Y_e(j) = C(j)X_e(j)$, with $j = 0, \ldots, 2J-1$.
5. Transform the product to the time domain with IFFT: $\mathbf{y}_e = \mathcal{IFFT}(\mathbf{Y}_e)$.
6. Truncation of the extended matrix product: $\mathbf{y} = \{y_e(j), j = 0, \ldots, J-1\}$.

This procedure involves three $2J$-component FFT operations (two direct FFTs and one inverse FFT), but the order of the complexity is $\mathcal{O}(J\log_2(J))$ instead of $\mathcal{O}(J^2)$, which significantly saves computation, particularly for large dimensionality J.

It should be noted that a recursive multiplication (like that required in the IRSA algorithm) cannot be fully performed in the frequency domain with a simple recursive algebraic product, because truncation must be applied at each iteration. Truncation is easily applied in the time domain (it is equivalent to multiplying a portion of the signal with a null vector, i.e. it is a diagonal operation in the time domain), but it is not a simple operation in the frequency domain (it can be performed in the frequency domain as a circular convolution with the Fourier Transform of the truncating window, with complexity $\mathcal{O}(J^2)$). Therefore, in order to perform truncation in the time domain, it is preferable to apply IFFT, truncation and FFT at each iteration (with complexity $\mathcal{O}(J\log_2(J))$.

As previously discussed, due to the limited numerical precision, if the coefficients of T and $\mathbf{x}$ are real, it is convenient to discard the imaginary part of the result.

6 Code for matrix IRSA algorithm, FFT implementation

The following MatLab / Octave code process the electroencephalogram and iteratively estimates the evoked response with the matrix IRSA algorithm with a FFT based implementation of the matrix products.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [xi ,t_run,t_iter] = IRSA_matrix_fft(y,m,I,J,alpha,OUTPUT)
% IRSA, matrix implementation with matrix product in fft domain
%    (this is possible because Rs is a Toeplitz-symmetric matrix)
%  Input parameters:  y (Recorded EEG)
%                     m (Trigger vector)
%                     I (Number of iterations)
%                     J (Length of the averaging window in samples)
%                     alpha (Convergence-control parameter)
%                     OUTPUT (flag for presenting results at iterations)
%  Output parameters: xi (AEP estimate)
%                     t_run (time required for algorithm execution)
%                     t_iter (time required for each iteration)
% Angel de la Torre, Jose Carlos Segura, Joaquin Valderrama 2019
%     University of Granada (Spain)
%     National Acoustic Laboratories, Macquarie University (Australia)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xi ,t_run,t_iter] = IRSA_matrix_fft(y,m,I,J,alpha,OUTPUT)
% Initialization
tic;                % time-stamp beginning of function
Es=length(m);       % energy of the stimulation signal (number of stimuli)
z0=zeros(J,1); rs0=zeros(J,1); s=zeros(size(y));
s(m)=1;             % stimulation signal
for j=1:J
    idx=j+m-1;
    z0(j)=sum(y(idx));  % cross-corr between EEG and stim. signal
    rs0(j)=sum(s(idx)); % autocorrelation of stim. signal
end
z0=z0/Es;              % first averaged response
rs0=rs0/Es;            % normalized autocorrelation stim. signal
RS=real(fft([rs0; 0; flipud(rs0(2:end))]));  % FT of autocorrelation
Z0=fft([z0; zeros(J,1)]);             % FT of z0
Zi=Z0; Xi=zeros(2*J,1);               % FT of zi and xi initialization
% Iterations
t_iter=toc;         % time-stamp for iterations
for i=1:I           % loop for iterations
    Xi=Xi+alpha*Zi;    % AEP estimate in freq. domain
    P=RS.*Xi;          % matrix product in freq. domain
    P1=real(ifft(P));  % this two lines are important in order to truncate
    P=fft([P1(1:J); zeros(J,1)]); % the estimation of the P in time domain
    Zi=Z0-P;
    if OUTPUT==1
        fprintf('Iteration %d:   res: %.16f\n',i,std(real(Zi)));
    end
end
xi=real(ifft(Xi));    % the result is transformed to time domain
xi=xi(1:J);           % ...and truncated to remove the non-causal part
t_run=toc;               % total execution time
t_iter=(t_run-t_iter)/I; % execution time for each iteration
return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

7 Bounds for the eigenvalues of symmetric Toeplitz matrices and application for setting the convergence parameter

As previously discussed, for circulant matrices the estimation of the eigenvalues is a simple task (the FFT of the first row of a circulant matrix directly provides the eigenvalues). However, computing the eigenvalues is not so simple in general for Toeplitz matrices.

There are some results concerning circulant matrices embedding symmetric Toeplitz matrices that provide relationships between their respective eigenvalues. This section describes a procedure that allows a simple and accurate estimation of bounds for the largest and smallest eigenvalues of the autocorrelation matrix R_s in the IRSA method (which is a symmetric Toeplitz positive semi-definite matrix) as a function of the eigenvalues of an associated circulant matrix (which can easily be calculated with FFT). This result is very useful for setting the convergence parameter of the IRSA algorithm.

7.1 Bound for the eigenvalues of the symmetric Toeplitz matrix

Let T be a $J \times J$ symmetric Toeplitz matrix, defined by its first row $\mathbf{t} = \{t_0, t_1, \ldots, t_{J-1}\}$:

$$T = \begin{pmatrix} t_0 & t_1 & t_2 & \cdots & t_{J-1} \\ t_1 & t_0 & t_1 & \cdots & t_{J-2} \\ t_2 & t_1 & t_0 & \cdots & t_{J-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{J-1} & t_{J-2} & t_{J-3} & \cdots & t_0 \end{pmatrix} \tag{34}$$

and let C be its $2J \times 2J$ circulant extension, defined from the $2J$-length sequence:

$$\mathbf{c} = \{c_j, j = 0, \ldots, 2J-1\} = \{t_0, t_1, t_2, \ldots, t_{J-1}, 0, t_{J-1}, t_{J-2}, \ldots, t_1\} \tag{35}$$

i.e., the matrix:

$$C = \left(\begin{array}{ccccc|ccccc} t_0 & t_1 & t_2 & \cdots & t_{J-1} & 0 & t_{J-1} & t_{J-2} & \ldots & t_1 \\ t_1 & t_0 & t_1 & \cdots & t_{J-2} & t_{J-1} & 0 & t_{J-1} & \ldots & t_2 \\ t_2 & t_1 & t_0 & \cdots & t_{J-3} & t_{J-2} & t_{J-1} & 0 & \ldots & t_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{J-1} & t_{J-2} & t_{J-3} & \cdots & t_0 & t_1 & t_2 & t_3 & \ldots & 0 \\ \hline 0 & t_{J-1} & t_{J-2} & \ldots & t_1 & t_0 & t_1 & t_2 & \cdots & t_{J-1} \\ t_{J-1} & 0 & t_{J-1} & \ldots & t_2 & t_1 & t_0 & t_1 & \cdots & t_{J-2} \\ t_{J-2} & t_{J-1} & 0 & \ldots & t_3 & t_2 & t_1 & t_0 & \cdots & t_{J-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ t_1 & t_2 & t_3 & \ldots & 0 & t_{J-1} & t_{J-2} & t_{J-3} & \cdots & t_0 \end{array}\right) \tag{36}$$

Since C is a circulant matrix, it is diagonalized by the DFT (or FFT) and its eigenvalues λ_j (with $j = 0, \ldots, 2J-1$) are the components of the FFT applied to the first row of the matrix:

$$\{\lambda_0, \lambda_1, \ldots, \lambda_{2J-1}\} = \mathcal{FFT}(\{t_0, t_1, t_2, \ldots, t_{J-1}, 0, t_{J-1}, t_{J-2}, \ldots, t_1\} \tag{37}$$

Let μ_j (with $j = 0, \ldots, J-1$) be the eigenvalues of the matrix T. According to [Hertz 1992], the maximum and minimum eigenvalues of T are easily bounded by the maximum and minimum eigenvalues of C:

$$\max(\mu_j) \leq \max_{0 \leq j' < 2J}(\lambda_{j'}) \tag{38}$$

$$\min(\mu_j) \geq \min_{0 \leq j' < 2J}(\lambda_{j'}) \tag{39}$$

This result provides a very simple boundary of all the eigenvalues of the matrix T.

Similarly, in [Ferreira 1994] better bounds are proposed for the largest and smallest eigenvalues of T:

$$\max(\mu_j) \leq \frac{1}{2}\left(\max_{0 \leq j' < J}(\lambda_{2j'}) + \max_{0 \leq j' < J}(\lambda_{2j'+1})\right) \tag{40}$$

$$\min(\mu_j) \geq \frac{1}{2}\left(\min_{0 \leq j' < J}(\lambda_{2j'}) + \min_{0 \leq j' < J}(\lambda_{2j'+1})\right) \tag{41}$$

where $\lambda_{2j'}$ and $\lambda_{2j'+1}$ are, respectively, the even and odd eigenvalues of the circulant matrix C. Even the derivation of this result requires some manipulations of auxiliary matrices, its application is very simple, since the eigenvalues of the circulant matrix are easily calculated with the FFT and they directly provide the upper and lower bounds.

7.2 Application of the bound to the IRSA algorithm

The analysis of convergence of the IRSA algorithm establishes a maximum value of the convergence parameter α related to the maximum eigenvalue of the autocorrelation matrix R_s:

$$\alpha < \frac{2}{\max(\mu_j)} \tag{42}$$

Since R_s is a symmetric Toeplitz positive semi-definite matrix, the bound in equation (40) and the convergence condition provide a bound for the convergence parameter which guarantees stability in the IRSA algorithm. A value of α smaller than the bound but close to it provides stability and a convergence as fast as possible.

Therefore, in order to simultaneously provide fast convergence and stability of the algorithm, the convergence parameter has been selected, in the fast implementation of IRSA, with the value:

$$\alpha = \frac{1.9}{\frac{1}{2}\left(\max(\lambda_{2j'}) + \max(\lambda_{2j'+1})\right)} \tag{43}$$

i.e. the convergence parameter α has been selected as a 95% of the maximum value suggested by the bound for the eigenvalues of R_s provided by equation (40).

References:

- Hertz, D. (1992). Simple bounds on the extreme eigenvalues of Toeplitz matrices. IEEE Transactions on Information Theory 38(1), 175-176.
- Ferreira, P.J.S.G. (1994). Localization of the eigenvalues of Toeplitz matrices using additive decomposition, embedding in circulants, and the Fourier transform. In: M. Blanke, T. Sï¿½derstrï¿½m (Eds.), Proceedings of SysID'94, 10th IFAC Symposium on System Identification, vol III, Copenhagen, Denmark, July 1994, pp. 271-276.

8 Code for matrix IRSA algorithm, FFT implementation, fast convergence

The following MatLab / Octave code process the electroencephalogram and iteratively estimates the evoked response with the matrix IRSA algorithm with a FFT based implementation and fast convergence (thanks to the optimization of the convergence parameter α and by applying a predefined convergence criterion).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [xi ,t_run,t_iter] = IRSA_matrix_fft_fast(y,m,I,SNR,J,OUTPUT)
% IRSA, matrix implementation with matrix product in fft domain
%    (this is possible because Rs is a Toeplitz-symmetric matrix)
%    Fast version: alpha / converg. criterion optimized (120dB num. error)
%  Input parameters:  y (Recorded EEG)
%                     m (Trigger vector)
%                     I (maximum number of iterations)
%                     SNR (SNR for numerical error in convergence criterion)
%                     J (Length of the averaging window in samples)
%                     OUTPUT (flag for presenting results at iterations)
%  Output parameters: xi (AEP estimate)
%                     t_run (time required for algorithm execution)
%                     t_iter (time required for each iteration)
% Angel de la Torre, Jose Carlos Segura, Joaquin Valderrama 2019
%     University of Granada (Spain)
%     National Acoustic Laboratories, Macquarie University (Australia)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xi ,t_run,t_iter] = IRSA_matrix_fft_fast(y,m,I,SNR,J,OUTPUT)
% Initialization
tic;                % time-stamp beginning of function
Es=length(m);       % energy of the stimulation signal (number of stimuli)
z0=zeros(J,1); rs0=zeros(J,1); s=zeros(size(y));
s(m)=1;             % stimulation signal
for j=1:J % cross-corr between EEG and stim. signal and autocorr. stim. signal
    idx=j+m-1; z0(j)=sum(y(idx)); rs0(j)=sum(s(idx));
end
z0=z0/Es;              % first averaged response
rs0=rs0/Es;            % normalized autocorrelation stim. signal
RS=real(fft([rs0; 0; flipud(rs0(2:end))]));       % FT of autocorrelation
Z0=fft([z0; zeros(J,1)]); Zi=Z0; Xi=zeros(2*J,1); % FT of z0, zi and xi
lambda_1=max(RS(1:2:J)); lambda_2=max(RS(2:2:J));
max_mu=0.5*(lambda_1+lambda_2); % bound for max eigenvalue of autoc. matrix
alpha=1.9/max_mu;       % selected alpha (close to maximum value)
ener_z0=Z0'*Z0;         % energy of RSA solution used for convergence
thr_conv=10^(-SNR/20);  % threshold for convergence criterion
% Iterations
t_iter=toc;         % time-stamp for iterations
for i=1:I           % loop for iterations
    Xi=Xi+alpha*Zi;    % AEP estimate in freq. domain
    P=RS.*Xi;          % matrix product in freq. domain
    P1=real(ifft(P));  % this two lines are important in order to truncate
    P=fft([P1(1:J); zeros(J,1)]); % the estimation of the P in time domain
    Zi=Z0-P;
    ener_zi=Zi'*Zi;             % energy of the correction at current it.
    ratio=sqrt(ener_zi/ener_z0); % ratio of correction vs initialization
    if ratio<thr_conv, break; end; % convergence criterion (loop broken)
    if OUTPUT==1, fprintf('It.%d: ratio:%.16f alpha=%.5f\n',i,ratio,alpha); end;
end
xi=real(ifft(Xi));    % the result is transformed to time domain
xi=xi(1:J);           % ...and truncated to remove the non-causal part
t_run=toc;               % total execution time
t_iter=(t_run-t_iter)/i; % execution time for each iteration
return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

9 Code for the direct RSLSD algorithm (RSLSD at convergence)

The following MatLab / Octave code process the electroencephalogram and directly estimates the evoked response at convergence with the RSLSD algorithm. This algorithm requires no convergence parameter nor iterations.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [xi ,t_run] = RSLSD_inf(y,m,J)
% Randomized Stimulation with Least Squares Deconvolution: direct
% deconvolution (infinite iterations) with matrix inversion
%     xi =  Rs^(-1) z0         xi = Rs\z0;
%  Input parameters:  y (Recorded EEG)
%                     m (Trigger vector)
%                     J (Length of the averaging window in samples)
%  Output parameters: xi (AEP estimate)
%                     t_run (time required for algorithm execution)
% Angel de la Torre, Jose Carlos Segura, Joaquin Valderrama 2019
%     University of Granada (Spain)
%     National Acoustic Laboratories, Macquarie University (Australia)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xi ,t_run] = RSLSD_inf(y,m,J)
% Initialization
tic;                 % time-stamp beginning of function
Es=length(m);        % energy of the stimulation signal (number of stimuli)
z0=zeros(J,1); rs0=zeros(J,1); s=zeros(size(y)); Rs=zeros(J,J);
s(m)=1;              % stimulation signal
for j=1:J
    idx=j+m-1;
    z0(j)=sum(y(idx));  % cross-corr between EEG and stim. signal
    rs0(j)=sum(s(idx)); % autocorrelation of stim. signal
end
z0=z0/Es;               % first averaged response
rs0=rs0/Es;             % normalized autocorrelation stim. signal
for i=1:J
    j=1:J; idx=abs(j-i)+1;
    Rs(i,j)=rs0(idx); % autocorrelation matrix
end
xi=(Rs\z0);             % direct deconvolution by matrix inversion
t_run=toc;              % total execution time
return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

10 Code for the iterative RSLSD algorithm (RSLSD for a finite number of iterations)

The following MatLab / Octave code process the electroencephalogram and iteratively estimates the evoked response with the RSLSD algorithm (iterative version). Note that this version is computationally more complex than the direct RSLSD algorithm.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [xi ,t_run,t_iter] = RSLSD_iter(y,m,I,J,alpha,OUTPUT)
% Randomized Stimulation with Least Squares Deconvolution: matrix
% implementation using geometric series of matrices, until iteration i
%     xi = (I-(I-alpha Rs)^i) * Rs^(-1) * z0
%  Input parameters:  y (Recorded EEG)
%                     m (Trigger vector)
%                     I (Number of iterations)
%                     J (Length of the averaging window in samples)
%                     alpha (Convergence-control parameter)
%                     OUTPUT (flag for presenting results at iterations)
%  Output parameters: xi (AEP estimate)
%                     t_run (time required for algorithm execution)
%                     t_run_iter (time required for each iteration)
% Angel de la Torre, Jose Carlos Segura, Joaquin Valderrama 2019
%     University of Granada (Spain)
%     National Acoustic Laboratories, Macquarie University (Australia)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xi ,t_run,t_iter] = RSLSD_iter(y,m,I,J,alpha,OUTPUT)
% Initialization
tic;                % time-stamp beginning of function
Es=length(m);       % energy of the stimulation signal (number of stimuli)
z0=zeros(J,1); rs0=zeros(J,1); s=zeros(size(y)); Rs=zeros(J,J);
s(m)=1;             % stimulation signal
for j=1:J
    idx=j+m-1;
    z0(j)=sum(y(idx));  % cross-corr between EEG and stim. signal
    rs0(j)=sum(s(idx)); % autocorrelation of stim. signal
end
z0=z0/Es;               % first averaged response
rs0=rs0/Es;             % normalized autocorrelation stim. signal
for i=1:J
    j=1:J; idx=abs(j-i)+1;
    Rs(i,j)=rs0(idx); % autocorrelation matrix
end
xia=(Rs\z0);          % xia = Rs^(-1) z0
A=eye(J)-alpha*Rs;   % A = (I - alpha Rs)
xib=xia;
% Iterations
t_iter=toc;          % time-stamp for iterations
for i=1:I
    xib=A*xib;       %    (I-alpha Rs)^i * Rs^(-1) * z0
    if OUTPUT==1
        fprintf('Iteration %d:   res: %.16f\n',i,std(xib));
    end
end
xi=xia-xib;
t_run=toc;               % total execution time
t_iter=(t_run-t_iter)/I; % execution time for each iteration
return;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

11 Detailed results of IRSA implementations at 50 iteration, α=0.02

This section presents detailed results of the different IRSA implementations, using a convergence parameter α=0.02 and after running the iterative algorithms for 50 iterations. The results compared at these conditions are obtained with conventional IRSA, matrix IRSA, matrix IRSA with optimized initialization (matrix-opt), matrix IRSA with FFT-based implementation of the matrix product (matrix-fft) and IRSA implemented with the geometric series of matrices (RSLSD-iterative).

- Figure 1 shows the responses estimated with the conventional IRSA algorithm with α=0.02 and at 50 iterations. Each figure shows the responses for a subject, obtained with ISI configurations 480-960 ms, 240-480 ms, 120-240 ms, 60-120 ms, 30-60 ms and 15-30 ms (from top to bottom in the figures). Figures obtained with the other algorithms look identical. Different waves (including auditory brainstem responses, ABR, middle latency responses, MLR, and cortical auditory evoked potentials, CAEP) can be identified.
- Table 1 evaluates the differences among the responses provided by the different algorithms. The differences are evaluated with the SNR, defined as the ratio of the response energy to that of the difference, expressed in dB and using the responses provided by the conventional IRSA algorithm as reference. A SNR around 260 dB indicates similar results up to the 13^{th} significant digit, and an increment of 20 dB represents another decimal digit in accuracy. The high SNR obtained in table 1 is an indicator of the mathematical equivalence of the algorithm implementations, and the small differences among the resulting responses are associated to the limited precision of the numerical representation (around 16 significant digits for 64-bits double precision floating-point representation) and the propagation of small errors in the least significant digits in the numerical procedures.
- Tables 2, 3 and 4 evaluates the computational cost of the different algorithm implementations. The table 2 represents the total execution time (including the initialization and the iterations), the table 3 shows the time required for the algorithm initialization, and the table 4 shows the time required for each iteration in the iterative algorithms.

SNR referred to IRSA-conventional; α=0.02; 50 iterations				
ISI	IRSA (matrix)	IRSA (matrix-opt)	IRSA (matrix-fft)	RSLSD (iterative)
480-960 ms	309.70 (0.52) dB	312.75 (0.19) dB	308.49 (0.18) dB	278.99 (0.35) dB
240-480 ms	308.36 (0.19) dB	310.39 (0.16) dB	308.03 (0.15) dB	271.89 (0.48) dB
120-240 ms	307.23 (0.45) dB	308.58 (0.38) dB	307.38 (0.13) dB	264.39 (2.37) dB
60-120 ms	304.23 (1.31) dB	305.10 (1.21) dB	306.31 (0.55) dB	261.69 (1.23) dB
300-60 ms	300.44 (2.54) dB	301.00 (2.57) dB	303.95 (1.18) dB	256.69 (1.94) dB
15-30 ms	297.63 (4.70) dB	298.00 (4.70) dB	300.71 (3.31) dB	253.36 (5.06) dB
Average	**304.60 (4.87) dB**	**305.97 (5.66) dB**	**305.81 (3.06) dB**	**264.34 (8.91) dB**

Table 1: Comparison of the responses provided by the different IRSA algorithms at 50 iterations with α=0.02. The SNR evaluates the difference between the responses (ratio of the response energy to the energy of the difference, expressed in dB, using the responses provided by the conventional IRSA algorithm as reference). These results are mean values averaged with the 4 subjects (standard deviation in parenthesis).

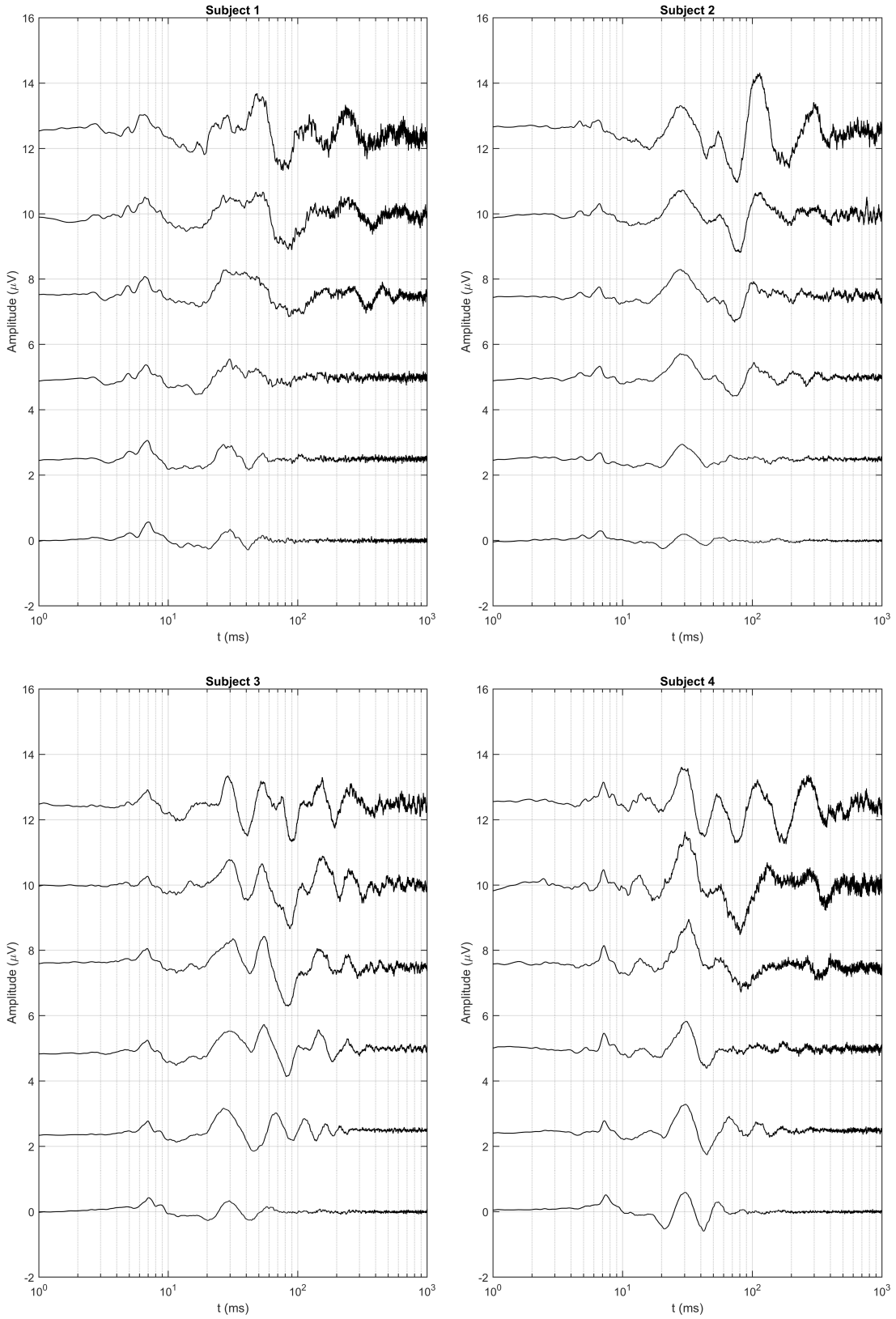

Figure 1: Responses estimated with the conventional IRSA algorithm with α=0.02 after 50 iterations, for the 4 subjects included in the study. From top to bottom, the responses correspond to the ISI configurations: 480-960 ms, 240-480 ms, 120-240 ms, 60-120 ms, 30-60 ms, and 15-30 ms.

Total execution time (α=0.02; 50 iterations)					
ISI	IRSA (conventional)	IRSA (matrix)	IRSA (matrix-opt)	IRSA (matrix-fft)	RSLSD (iterative)
480-960 ms	17.99 (0.75) s	17.83 (1.64) s	12.35 (0.78) s	0.69 (0.02) s	27.87 (1.97) s
240-480 ms	33.34 (0.59) s	17.54 (0.90) s	12.75 (0.44) s	1.11 (0.02) s	28.73 (2.97) s
120-240 ms	63.68 (0.60) s	17.54 (1.02) s	13.52 (0.50) s	1.89 (0.04) s	29.18 (2.06) s
60-120 ms	125.05 (2.40) s	17.22 (0.84) s	15.19 (0.85) s	3.41 (0.06) s	31.69 (3.32) s
300-60 ms	245.37 (1.51) s	16.84 (0.46) s	17.52 (0.40) s	6.26 (0.14) s	34.82 (5.00) s
15-30 ms	485.06 (1.87) s	16.61 (0.67) s	21.76 (0.34) s	10.58 (0.12) s	37.53 (2.70) s
All	**970.49 (5.30) s**	**103.58 (4.01) s**	**93.08 (2.41) s**	**23.94 (0.26) s**	**189.82 (13.60) s**

Table 2: Comparison of the computational cost associated to the different IRSA algorithms at 50 iterations with α=0.02. Total execution time, expressed in seconds, includes the initialization and the 50 iterations. These results are mean values per subject, averaged with the 4 subjects (standard deviation in parenthesis).

Execution time: initialization (α=0.02; 50 iterations)					
ISI	IRSA (conventional)	IRSA (matrix)	IRSA (matrix-opt)	IRSA (matrix-fft)	RSLSD (iterative)
480-960 ms	0.14 (0.02) s	12.41 (1.33) s	6.95 (0.47) s	0.61 (0.01) s	22.47 (1.69) s
240-480 ms	0.32 (0.05) s	12.07 (0.69) s	7.23 (0.16) s	1.03 (0.02) s	23.22 (2.39) s
120-240 ms	0.55 (0.03) s	12.11 (0.80) s	8.06 (0.10) s	1.81 (0.04) s	23.73 (1.64) s
60-120 ms	1.12 (0.03) s	11.86 (0.59) s	9.65 (0.26) s	3.33 (0.06) s	26.07 (2.68) s
30-60 ms	2.13 (0.04) s	11.61 (0.28) s	12.26 (0.16) s	6.18 (0.14) s	29.05 (3.78) s
15-30 ms	4.16 (0.07) s	11.49 (0.46) s	16.64 (0.16) s	10.50 (0.12) s	32.08 (2.32) s

Table 3: Comparison of the computational cost associated to the initialization of the different IRSA algorithms. These results are mean values averaged with the 4 subjects (standard deviation in parenthesis).

Execution time per iteration (α=0.02; 50 iterations)					
ISI	IRSA (conventional)	IRSA (matrix)	IRSA (matrix-opt)	IRSA (matrix-fft)	RSLSD (iterative)
480-960 ms	356.9 (14.7) ms	108.36 (6.26) ms	107.93 (6.53) ms	1.610 (0.025) ms	108.03 (5.60) ms
240-480 ms	660.3 (10.8) ms	109.37 (4.62) ms	110.37 (5.61) ms	1.630 (0.042) ms	110.17 (11.65) ms
120-240 ms	1262.6 (12.0) ms	108.57 (4.66) ms	109.24 (8.28) ms	1.620 (0.034) ms	109.02 (8.41) ms
60-120 ms	2478.5 (47.5) ms	107.23 (5.31) ms	110.74 (11.83) ms	1.660 (0.113) ms	112.37 (13.44) ms
30-60 ms	4864.7 (29.9) ms	104.61 (3.71) ms	105.20 (4.84) ms	1.608 (0.035) ms	115.33 (24.58) ms
15-30 ms	9617.9 (36.8) ms	102.29 (4.50) ms	102.31 (3.89) ms	1.590 (0.018) ms	108.87 (8.69) ms

Table 4: Comparison of the computational cost associated to each iteration of the different IRSA algorithms. These results are mean values averaged with the 4 subjects (standard deviation in parenthesis).

12 Convergence of the IRSA algorithm: results at 50, 100, 500, 1000, 5000, 10000 iterations and at convergence

This section describes the effect of the number of iterations in the IRSA algorithm. The responses have been computed using the matrix IRSA with the FFT-based implementation of the matrix products, using α=0.02 and running the iterative procedure for a number of iterations ranging from 50 to 10000. The responses have been compared with those obtained at convergence (estimated with the RSLSD algorithm, using version directly providing the estimation at convergence).

- Table 5 compares the responses at a given number of iterations with those estimated at convergence in terms of SNR.
- Figure 2 represents the responses for subject 1, at different iterations and at convergence.
- As can be observed from a comparison of the plots and the table, a significant difference is appreciated in the plots for SNRs below 30 dB, while the plots look identical when the SNRs are above 45 dB. Under this last condition, even though the numerical convergence is not achieved, the differences are irrelevant from an audiological point of view (i.e. in practice the responses can be considered identical for wave identification or for wave latency/amplitude measurements). As can be observed, the number of iterations required for convergence increases as the stimulation rate decreases. For the selected convergence parameter, 10000 iterations are enough for numerical convergence only at the 480-960 ms ISI configuration, even though this number of iterations is enough for practical convergence with all the ISI configurations.

SNR referred to RSLSD; IRSA-matrix-fast; α=0.02; between 50 and 10000 iterations						
ISI	50 iter.	100 iter.	500 iter.	1000 iter.	5000 iter.	10000 iter.
480-960 ms	8.36 (0.44) dB	15.94 (1.60) dB	48.23 (10.98) dB	76.12 (11.62) dB	277.77 (4.43) dB	264.96 (8.55) dB
240-480 ms	6.84 (1.05) dB	10.96 (2.84) dB	19.23 (5.06) dB	26.15 (5.71) dB	72.91 (6.67) dB	130.03 (6.95) dB
120-240 ms	4.43 (1.73) dB	6.65 (2.70) dB	12.60 (2.72) dB	17.68 (2.43) dB	52.73 (3.10) dB	93.99 (4.16) dB
60-120 ms	2.41 (0.63) dB	3.38 (0.90) dB	7.64 (1.28) dB	12.29 (1.45) dB	45.15 (2.34) dB	83.70 (3.20) dB
30-60 ms	1.13 (0.25) dB	1.71 (0.25) dB	5.35 (0.47) dB	9.42 (0.74) dB	38.98 (1.42) dB	74.73 (1.63) dB
15-30 ms	0.94 (0.39) dB	1.49 (0.49) dB	4.93 (0.76) dB	8.86 (0.92) dB	38.29 (1.58) dB	73.35 (2.43) dB
Average	**4.02 (2.98) dB**	**6.69 (5.61) dB**	**16.33 (16.06) dB**	**25.09 (24.55) dB**	**87.64 (87.73) dB**	**120.13 (69.10) dB**

Table 5: Comparison of the responses provided by RSLSD (at convergence) and the IRSA-matrix-fft with α=0.02, as a function of the number of iterations. The differences are expressed in terms of SNR. These results are mean values averaged with the 4 subjects (standard deviation in parenthesis).

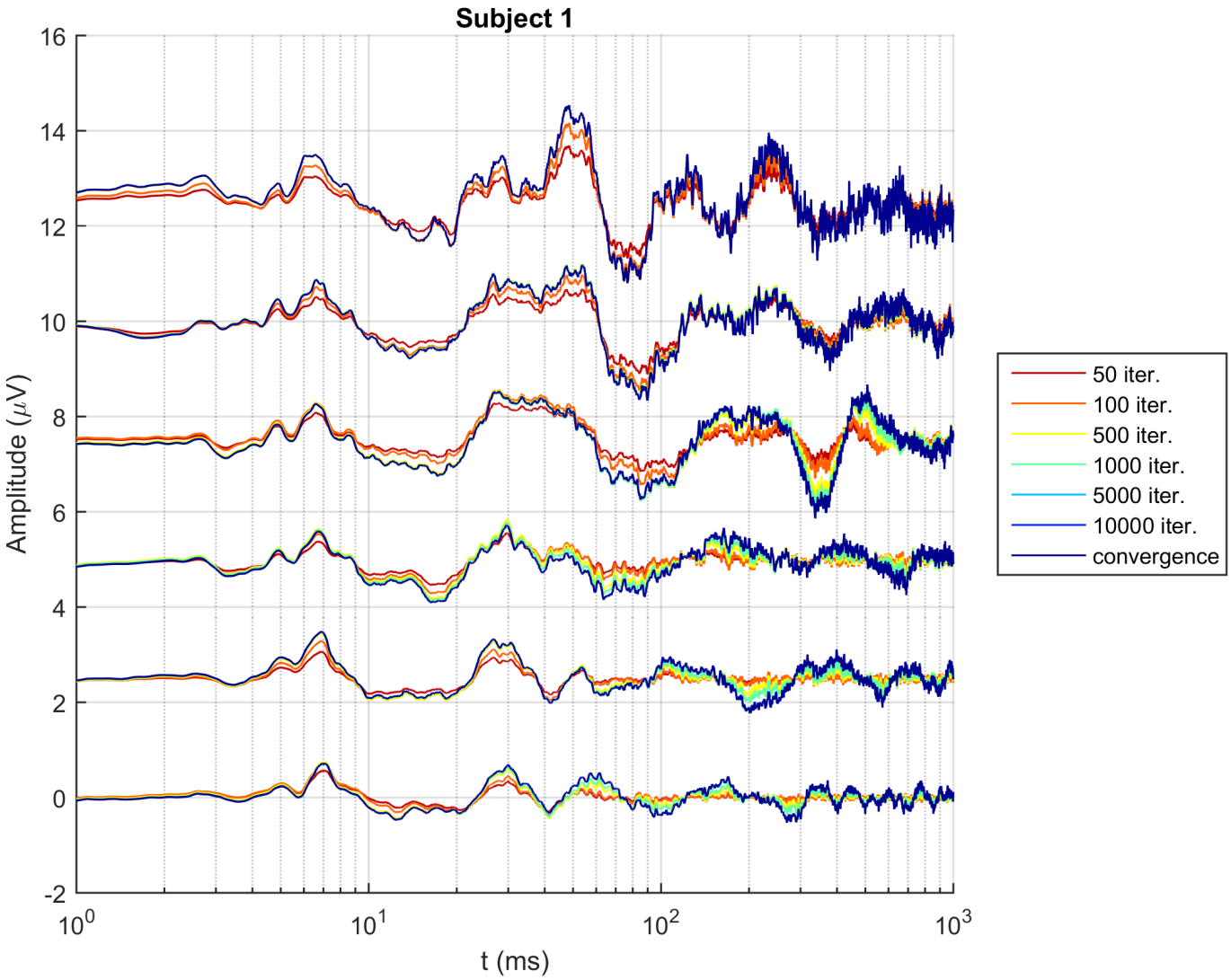

Figure 2: Responses estimated with the IRSA-matrix-fft algorithm with α=0.02 at 50, 100, 500, 1000, 5000 and 10000 iterations and with the RSLSD algorithm (direct estimation at convergence) for subject 1. From top to bottom, the responses correspond to the ISI configurations: 480-960 ms, 240-480 ms, 120-240 ms, 60-120 ms, 30-60 ms, and 15-30 ms.

13 Detailed results of the IRSA implementations at convergence

This section presents detailed results of the IRSA implementations at convergence. The results have been obtained with IRSA-matrix-fft (using a convergence parameter α=0.02 and 10000 iterations), with IRSA-matrix-fast (with convergence parameter selected automatically and convergence criterion configured at 290 dB and 120 dB) and with RSLSD (direct estimation of the response at convergence by matrix inversion). The convergence criterion of 290 dB guarantees numerical convergence, while that of 120 dB provides the accuracy required in practical situations.

- Figure 3 shows the responses estimated with the IRSA-matrix-fast algorithm using 120 dB as convergence criterion. Plots for each subject and for each ISI condition are provided. Figures obtained at convergence with the other algorithms or configurations look identical. Different waves, including ABR, MLR and CAEP can be identified.
- Table 6 evaluates the differences among the responses provided by the different IRSA algorithms at convergence. The differences are evaluated in terms of the SNR, using the responses provided by the RSLSD algorithm as reference. This table shows that in all the cases the IRSA algorithms provide a solution close enough to the convergence.
- Tables 7, 8 and 9 evaluate the computational cost of the different algorithm implementations. The table 7 represents the total execution time (including the initialization and the iterations), the table 8 shows the time required for the algorithm initialization, and the table 9 shows the time required for each iteration in the iterative algorithms. Finally, table 10 shows the convergence parameter used in the IRSA-matrix-fast implementations, and the number of iterations executed by this algorithm when it is configured with 290 dB and 120 dB as convergence criterion.

SNR referred to RSLSD (configuration for convergence)			
ISI	IRSA (matrix-fft) α=0.02, 10000 it.	IRSA (matrix-fast) 290 dB	IRSA (matrix-fast) 120 dB
480-960 ms	264.96 (8.55) dB	280.95 (1.29) dB	110.98 (0.74) dB
240-480 ms	130.03 (6.95) dB	266.78 (0.92) dB	96.59 (0.41) dB
120-240 ms	93.99 (4.16) dB	264.96 (1.28) dB	95.24 (1.83) dB
60-120 ms	83.70 (3.20) dB	263.69 (0.23) dB	93.84 (0.23) dB
300-60 ms	74.73 (1.63) dB	263.67 (1.51) dB	94.17 (1.63) dB
15-30 ms	73.35 (2.43) dB	146.25 (3.86) dB	94.48 (2.17) dB
Average	**120.13 (69.10) dB**	**247.72 (46.78) dB**	**97.55 (6.32) dB**

Table 6: Comparison of the responses provided by the different IRSA algorithms configured for convergence. The responses are compared with those provided by RSLSD (at convergence). The differences are expressed in terms of SNR. These results are mean values averaged with the 4 subjects (standard deviation in parenthesis).

Total execution time (configuration for convergence)				
ISI	IRSA (matrix-fft) α=0.02, 10000 it.	IRSA (matrix-fast) 290 dB	IRSA (matrix-fast) 120 dB	RSLSD (convergence)
480-960 ms	17.30 (0.22) s	0.81 (0.01) s	0.69 (0.02) s	23.59 (0.28) s
240-480 ms	17.61 (0.09) s	2.71 (0.06) s	1.59 (0.05) s	24.61 (0.44) s
120-240 ms	18.37 (0.13) s	6.23 (0.25) s	3.35 (0.11) s	24.76 (0.67) s
60-120 ms	20.08 (0.04) s	12.21 (0.38) s	6.35 (0.17) s	26.54 (0.70) s
300-60 ms	23.14 (0.16) s	24.69 (0.62) s	12.76 (0.44) s	29.62 (0.88) s
15-30 ms	28.17 (0.64) s	31.33 (1.13) s	24.03 (0.98) s	34.89 (1.13) s
All	**124.67 (0.95) s**	**77.98 (2.07) s**	**48.76 (1.48) s**	**164.01 (3.35) s**

Table 7: Comparison of the computational cost associated to the different IRSA algorithms configured for convergence. Total execution time (including initialization and iterations) expressed in seconds. These results are mean values per subject, averaged with the 4 subjects (standard deviation in parenthesis).

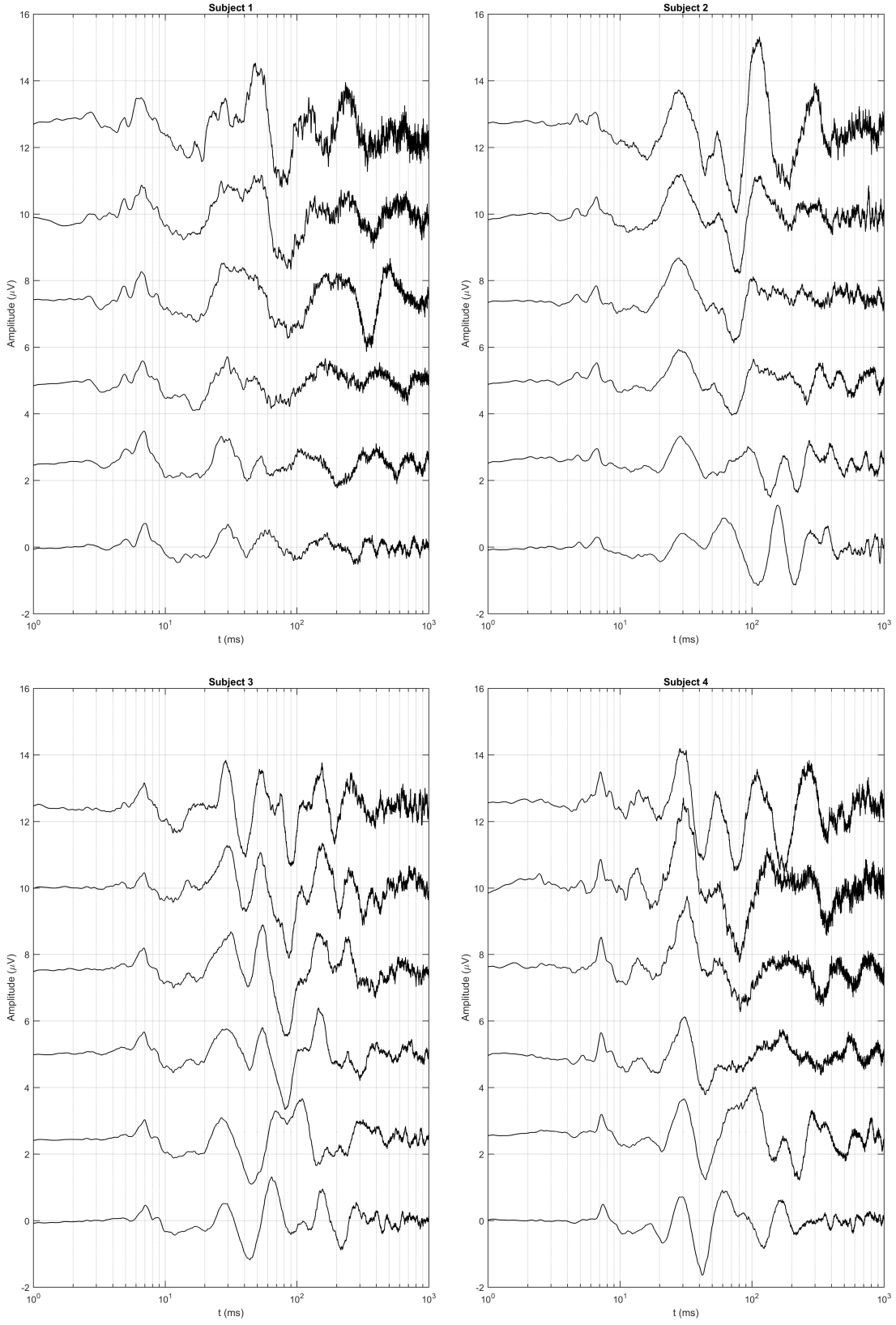

Figure 3: Responses estimated with the IRSA-matrix-fast algorithm, 120 dB as convergence criterion. From top to bottom, the responses correspond to the ISI configurations: 480-960 ms, 240-480 ms, 120-240 ms, 60-120 ms, 30-60 ms, and 15-30 ms.

Execution time: initialization (configuration for convergence)				
	IRSA (matrix-fft)	IRSA (matrix-fast)	IRSA (matrix-fast)	RSLSD (convergence)
ISI	α=0.02, 10000 it.	290 dB	120 dB	
480-960 ms	0.63 (0.01) s	0.62 (0.01) s	0.62 (0.01) s	23.59 (0.28) s
240-480 ms	1.07 (0.01) s	1.05 (0.01) s	1.06 (0.01) s	24.61 (0.44) s
120-240 ms	1.96 (0.01) s	1.94 (0.02) s	1.95 (0.01) s	24.76 (0.67) s
60-120 ms	3.57 (0.03) s	3.57 (0.01) s	3.54 (0.02) s	26.54 (0.70) s
30-60 ms	6.60 (0.08) s	6.65 (0.18) s	6.56 (0.06) s	29.62 (0.88) s
15-30 ms	11.72 (0.40) s	11.70 (0.43) s	11.67 (0.33) s	34.89 (1.13) s

Table 8: Comparison of the computational cost associated to the initialization of the different IRSA algorithms configured for convergence. These results are mean values averaged with the 4 subjects (standard deviation in parenthesis).

Execution time per iteration (configuration for convergence)				
	IRSA (matrix-fft)	IRSA (matrix-fast)	IRSA (matrix-fast)	RSLSD (convergence)
ISI	α=0.02, 10000 it.	290 dB	120 dB	
480-960 ms	1.667 (0.021) ms	1.991 (0.033) ms	1.901 (0.057) ms	0.000 (0.000) ms
240-480 ms	1.654 (0.009) ms	1.998 (0.034) ms	1.989 (0.049) ms	0.000 (0.000) ms
120-240 ms	1.641 (0.012) ms	1.961 (0.071) ms	1.963 (0.053) ms	0.000 (0.000) ms
60-120 ms	1.651 (0.002) ms	1.969 (0.044) ms	1.928 (0.057) ms	0.000 (0.000) ms
30-60 ms	1.654 (0.008) ms	1.982 (0.042) ms	1.974 (0.092) ms	0.000 (0.000) ms
15-30 ms	1.645 (0.024) ms	1.963 (0.073) ms	1.963 (0.063) ms	0.000 (0.000) ms

Table 9: Comparison of the computational cost associated to each iteration of the different IRSA algorithms configured for convergence. These results are mean values averaged with the 4 subjects (standard deviation in parenthesis).

Parameters of the fast implementation of IRSA			
	convergence criterion: 290 dB / 120 dB	converg. criterion: 290 dB	converg. criterion: 120 dB
ISI	selected α	number of iter.	number of iter.
480-960 ms	0.82852 (0.00817)	97.75 (7.68)	34.00 (4.90)
240-480 ms	0.52060 (0.00156)	829.50 (25.01)	267.25 (22.29)
120-240 ms	0.28299 (0.00151)	2185.00 (54.04)	713.50 (36.63)
60-120 ms	0.15548 (0.00026)	4387.25 (112.75)	1455.00 (57.38)
30-60 ms	0.08105 (0.00012)	9102.75 (152.43)	3139.75 (111.74)
15-30 ms	0.04164 (0.00003)	10000.00 (0.00)*	6291.00 (218.33)

Table 10: Selected convergence parameter α in the IRSA-matrix-fast implementations for each ISI configuration, and number of iterations required to achieve the selected convergence criterion (*note that for ISI 15-30 ms, when selected 290 dB for convergence, the convergence criterion is not achieved at the maximum number of iterations allowed in this test).

14 Effect of the response length J over the execution time

This section study the effect of the response length J (or the duration of the response window) over the total execution time.

- Table 11 shows the total execution time as a function of the response length for the different implementations of the IRSA algorithm. For those implementation more expensive in terms of computational cost (IRSA-conventional, IRSA-matrix, IRSA-matrix-opt and RSLSD-iterative) the algorithm has been executed for 50 iterations and the total execution time has been estimated for 10000 iterations from the results at 50 iterations taking into account the time required for both initialization and iterations.
- Figure 4 represents the total execution times as a function of the response length J for the different algorithm implementations. Figure 5 shows a detail of the previous one for the fastest algorithms.
- Figure 6 represents the responses estimated for subject 1 for different configurations of the response length, from J=147 (10 ms) to J=14700 (1 s). Interestingly, the responses obtained with different values of J are not identical: the effect of reducing the response length is, approximately, a truncation of the response, but in addition to the truncation, some differences are clearly observed. These differences are associated to the number of freedom degrees of the solution (the estimated response) involved in the optimization problem (i.e. the minimization of the residual) implemented with the IRSA (or RSLSD) algorithm.

Total execution time (for IRSA algorithms α=0.02, 50 iter., estimations for 10000 iterations)				
J (response duration)	IRSA (conventional)	IRSA (matrix)	IRSA (matrix-opt)	RSLSD (iterative)
14700 (1000 ms)	192421.4 (1038.1) s	6476.18 (241.78) s	6518.97 (323.41) s	6794.81 (589.06) s
7350 (500 ms)	107334.0 (126.6) s	1491.02 (29.07) s	1467.81 (17.24) s	1575.78 (10.21) s
3675 (250 ms)	50524.4 (26.0) s	403.11 (7.26) s	374.20 (2.77) s	397.90 (1.38) s
1470 (100 ms)	23199.4 (32.5) s	104.41 (0.39) s	59.51 (0.35) s	58.42 (0.91) s
735 (50 ms)	14368.6 (12.7) s	34.99 (0.23) s	5.49 (0.06) s	4.99 (0.12) s
368 (25 ms)	10064.4 (9.7) s	31.90 (0.07) s	2.24 (0.04) s	1.90 (0.03) s
147 (10 ms)	7465.74 (45.5) s	30.80 (0.22) s	1.00 (0.02) s	0.84 (0.02) s
Total execution time (for IRSA-matrix-fft, IRSA-matrix-fast and RSLSD)				
J (response duration)	IRSA (matrix-fft) α=0.02, 10000 it.	IRSA (matrix-fast) 290 dB	IRSA (matrix-fast) 120 dB	RSLSD (convergence)
14700 (1000 ms)	124.67 (0.95) s	77.98 (2.07) s	48.76 (1.48) s	164.01 (3.35) s
7350 (500 ms)	61.77 (0.09) s	28.48 (0.29) s	17.38 (0.11) s	32.27 (0.28) s
3675 (250 ms)	31.94 (0.05) s	10.38 (0.06) s	7.50 (0.06) s	9.86 (0.06) s
1470 (100 ms)	15.90 (0.04) s	3.42 (0.04) s	2.84 (0.03) s	3.00 (0.02) s
735 (50 ms)	9.19 (0.03) s	1.60 (0.01) s	1.45 (0.00) s	1.47 (0.00) s
368 (25 ms)	6.79 (0.02) s	0.94 (0.01) s	0.83 (0.00) s	0.80 (0.00) s
147 (10 ms)	4.28 (0.02) s	0.55 (0.01) s	0.47 (0.00) s	0.41 (0.00) s

Table 11: Comparison of the computational cost (execution time) associated to the different IRSA algorithms (configured for convergence), as a function of the response length. Results include the total execution time for all the ISI configurations. Results for IRSA-conventional, IRSA-matrix, IRSA-matrix-opt and RSLSD-iterative are estimations for 10000 iterations obtained from the results for 50 iterations. These results are mean values per subject, averaged with the 4 subjects (standard deviation in parenthesis).

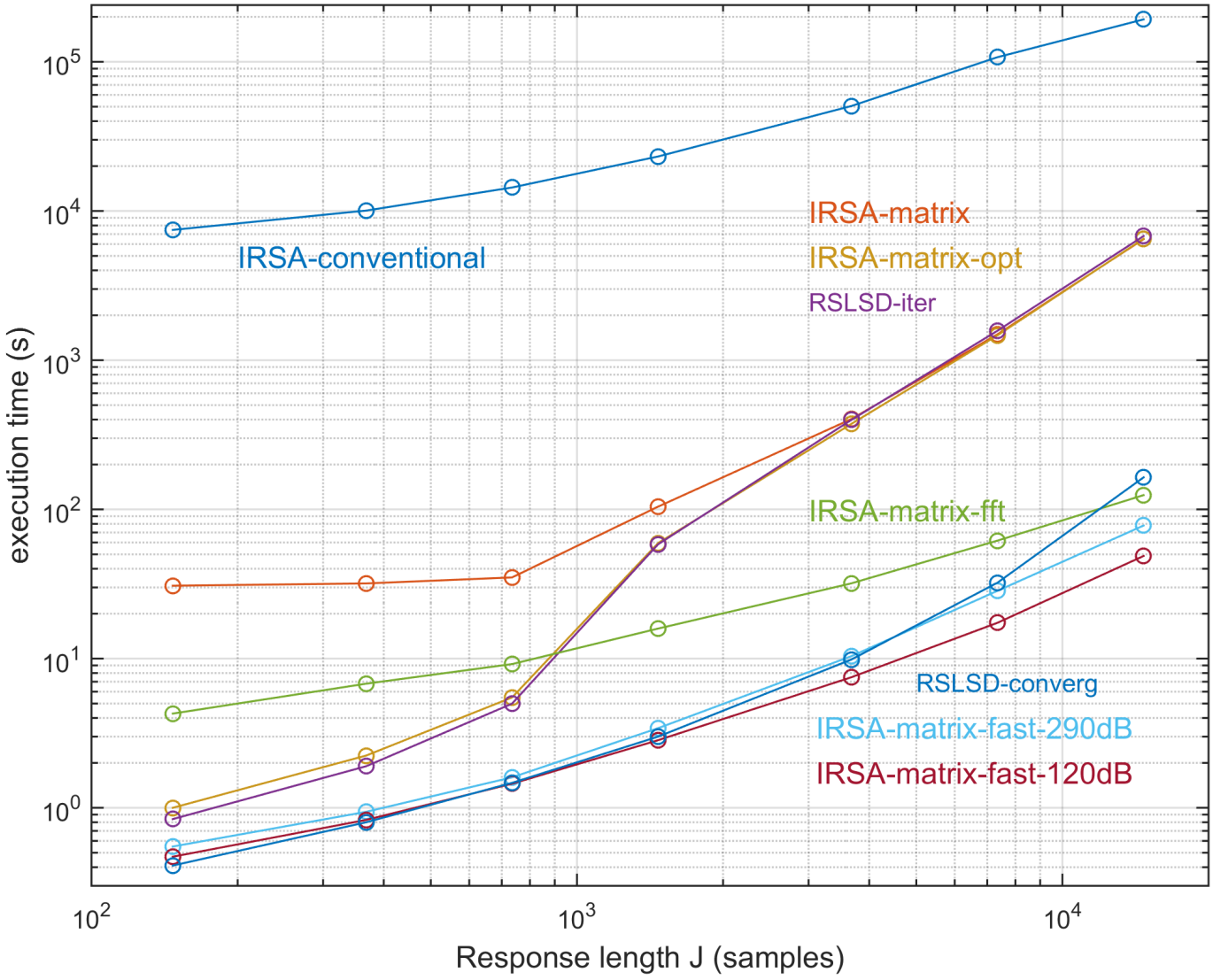

Figure 4: Execution time for the complete test (including all the ISI configurations) as a function of the response length J (in samples), for the different IRSA algorithms (configured for convergence). Results for IRSA-conventional, IRSA-matrix, IRSA-matrix-opt and RSLSD-iterative are estimations for 10000 iterations obtained from the results for 50 iterations. These results are mean values per subject.

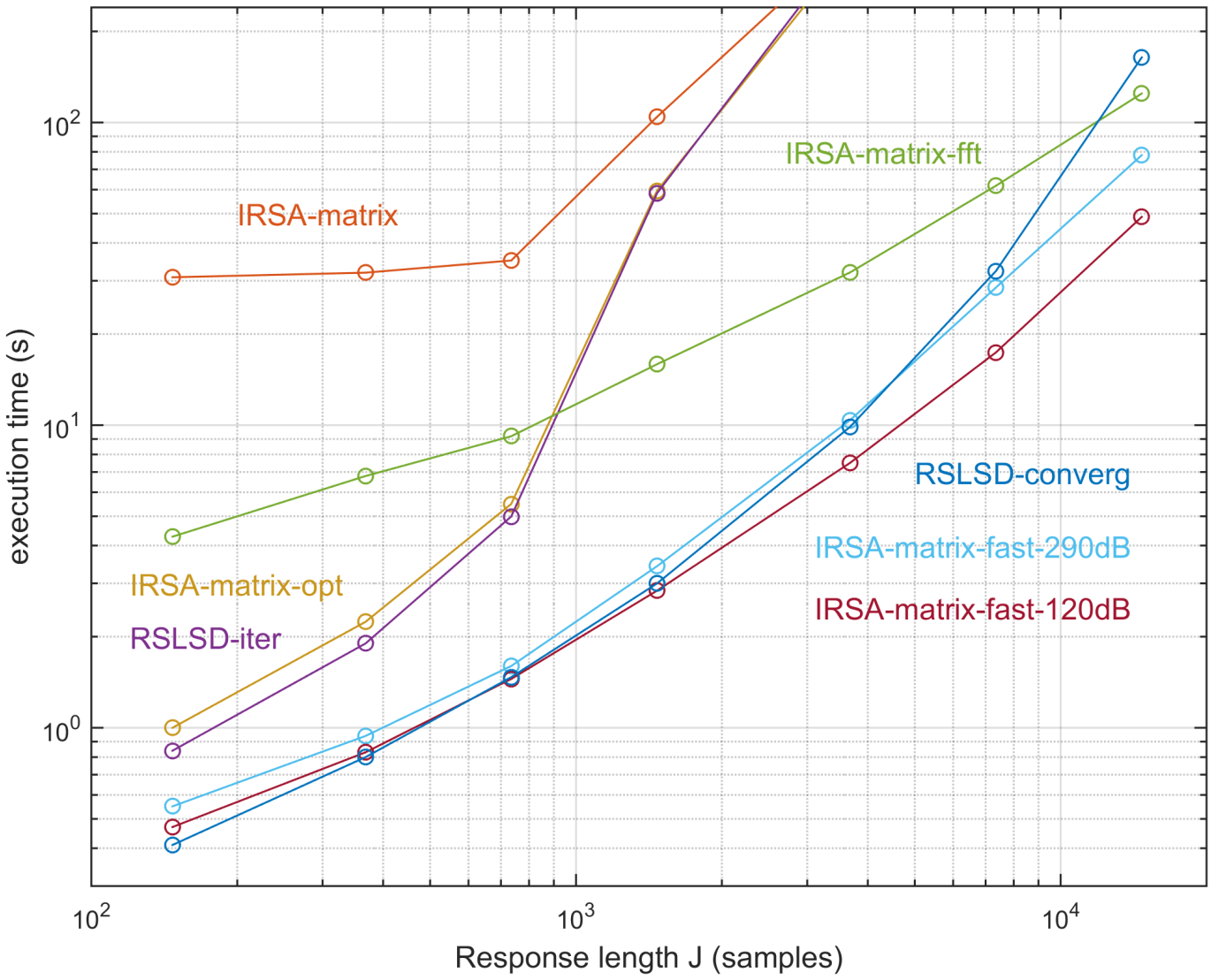

Figure 5: Execution time for the complete test (including all the ISI configurations) as a function of the response length J (in samples). Detail of the previous figure.

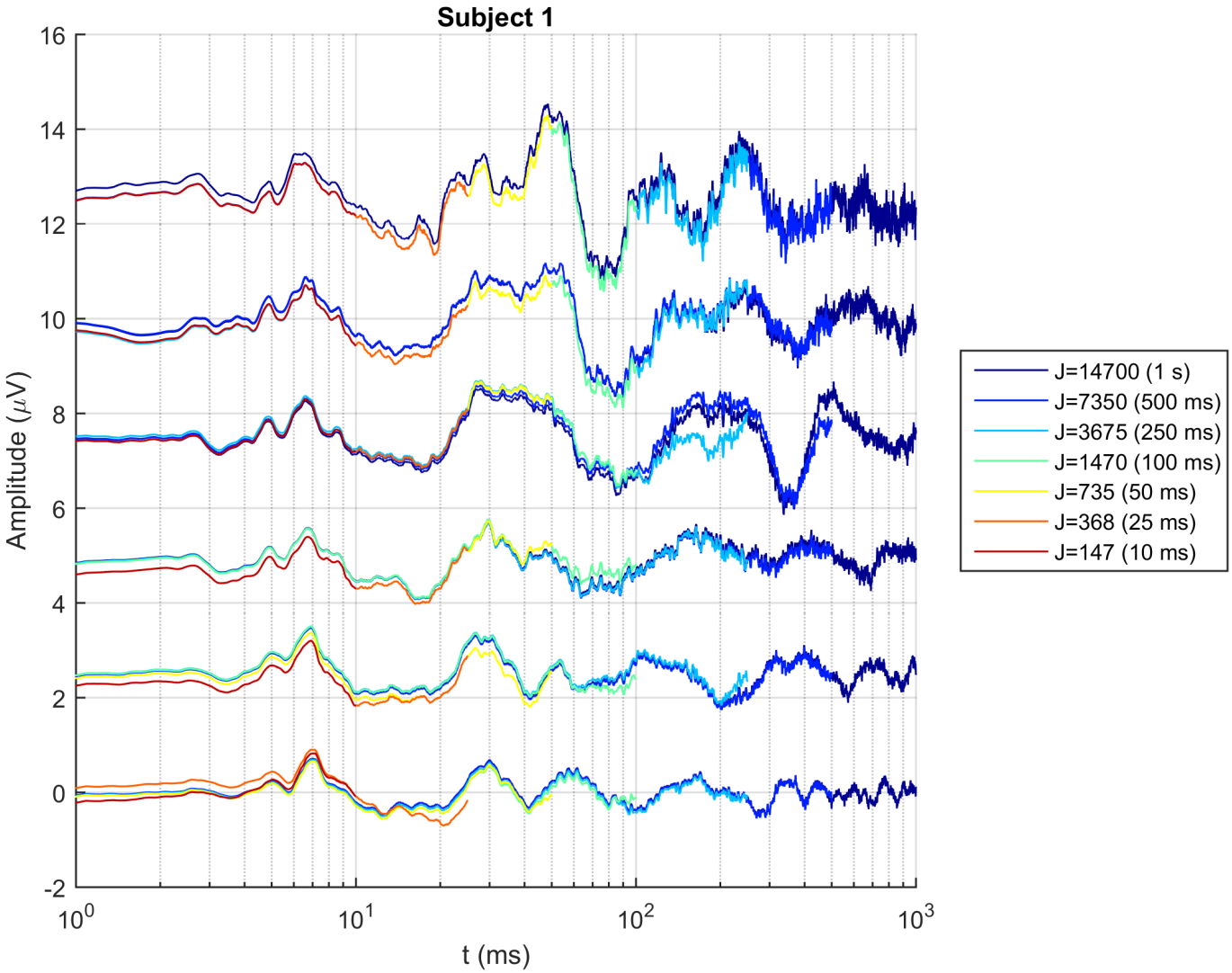

Figure 6: Responses estimated with the IRSA-matrix-fast algorithm, 120 dB as convergence criterion for different configurations of the response length J. This responses have been estimated for subject 1. From top to bottom, the responses correspond to the ISI configurations: 480-960 ms, 240-480 ms, 120-240 ms, 60-120 ms, 30-60 ms, and 15-30 ms.

15 Responses provided by IRSA

This section presents the responses obtained with IRSA at convergence. The responses have been estimated with the IRSA-matrix-fast algorithm with 120 dB as convergence criterion.

- Figures 7 and 8 show the responses for each subject (figure 7 for subjects 1 and 2; figure 8 for subjects 3 and 4). The plots in the left panels are the responses estimated from the complete EEG. Responses estimated with three EEG portions of 228 s have been also estimated for consistency verification of the evoked responses. The plots in the right panels represent the responses estimated with these short EEG portions. The responses for each subject and each condition are very consistent and the most relevant waves (including ABR, MLR and CAEP) can be identified in all the subjects.
- Figure 9 contains the evoked responses for the different ISI configurations. The left panel shows the grand average (estimated from all the subjects) while the right panel show the individual responses for each subject. The waves are very consistent in the grand average, even though some inter-individual differences can be appreciated. The most relevant waves (including ABR, MLR and CAEP: I, III, V, N_0, P_0, N_a, P_a, N_b, P_b/P_1, N_1, P_2, N_2, P_3) can easily be identified in the grand average responses.

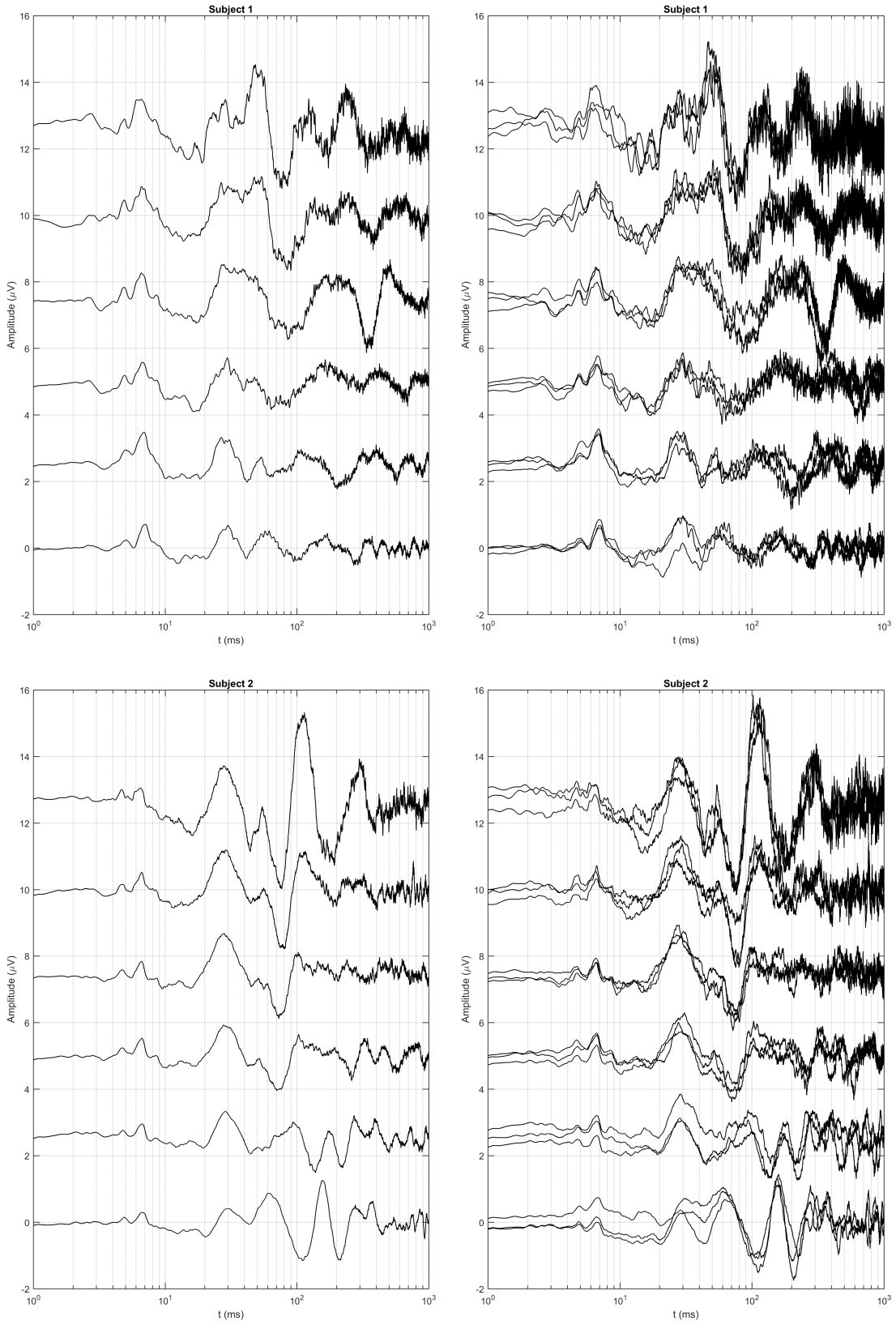

Figure 7: Responses estimated with the IRSA-matrix-fast algorithm, 120 dB as convergence criterion, subjects 1 and 2. Left panels: responses estimated from the complete EEG (684 s); right panels: responses estimated with three EEG portions of 228 s. From top to bottom, the responses correspond to the ISI configurations: 480-960 ms, 240-480 ms, 120-240 ms, 60-120 ms, 30-60 ms, and 15-30 ms.

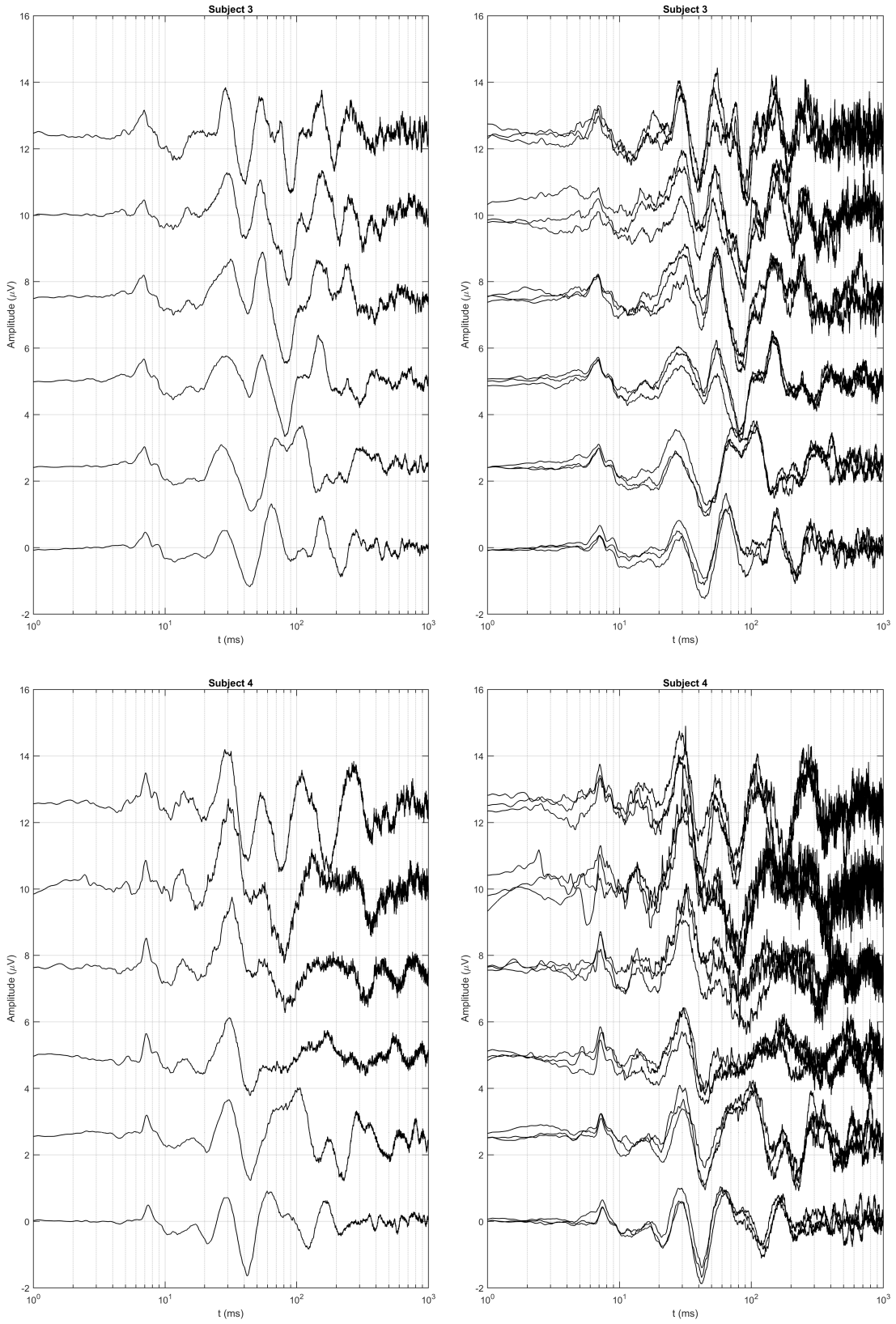

Figure 8: Responses estimated with the IRSA-matrix-fast algorithm, 120 dB as convergence criterion, subjects 3 and 4. Left panels: responses estimated from the complete EEG (684 s); right panels: responses estimated with three EEG portions of 228 s. From top to bottom, the responses correspond to the ISI configurations: 480-960 ms, 240-480 ms, 120-240 ms, 60-120 ms, 30-60 ms, and 15-30 ms.

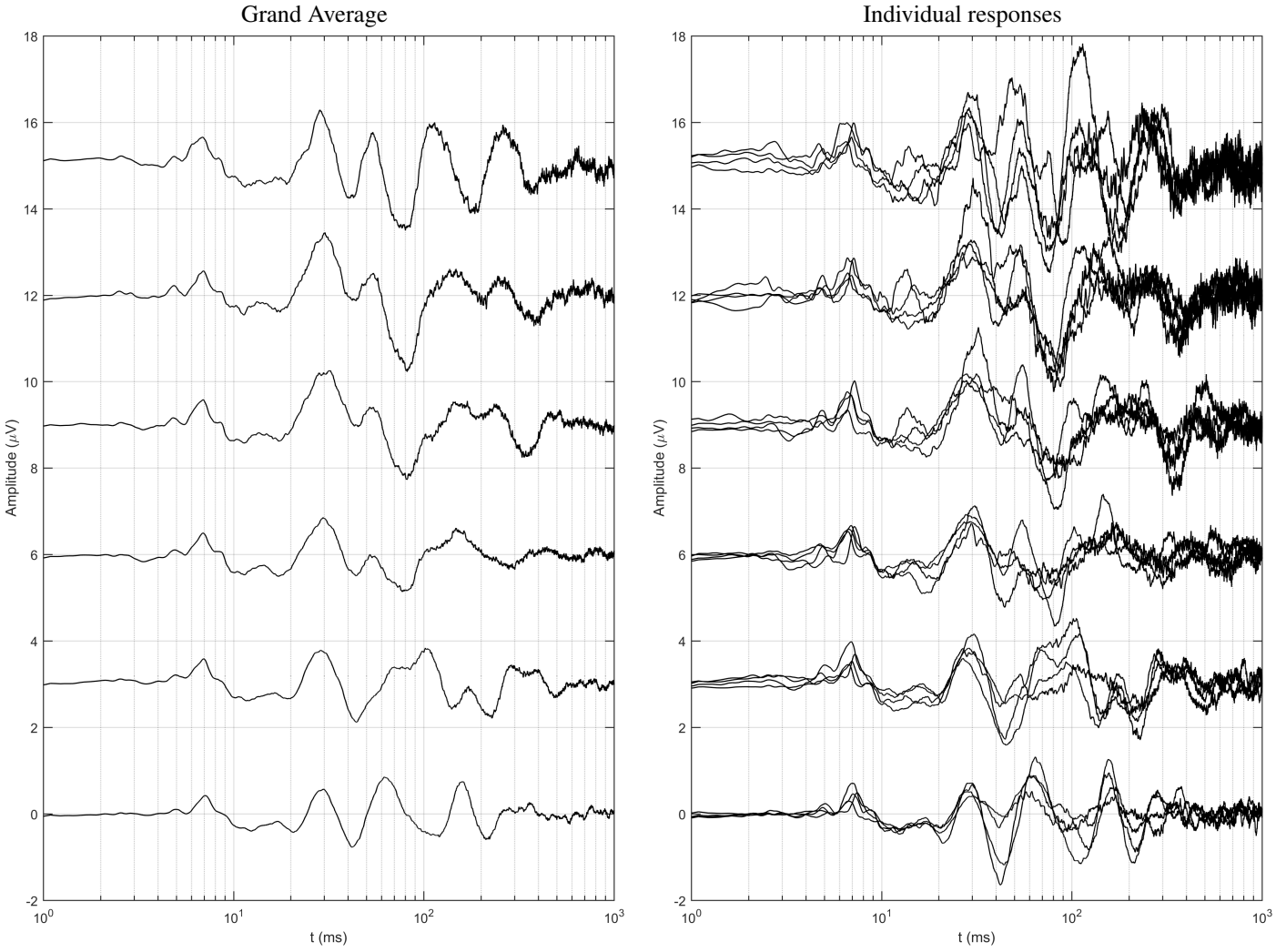

Figure 9: Responses estimated with the IRSA-matrix-fast algorithm, 120 dB as convergence criterion. Left panel: grand average (from all the subjects). Right panel: comparison of the individual responses for each subject. From top to bottom, the responses correspond to the ISI configurations: 480-960 ms, 240-480 ms, 120-240 ms, 60-120 ms, 30-60 ms, and 15-30 ms.

16 Code for a script running an AEP simulation and testing the different IRSA and RSLSD implementations

The following MatLab / Octave code is a script running a simulation for testing the different IRSA and RSLSD implementations. A reference response is loaded in “x”; the EEG is simulated using the response, and a predefined ISI configuration. Some noise is added. The responses are estimated using the different IRSA and RSLSD algorithm implementations and the results are compared. Different experiments can easily be simulated with the configuration parameters.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% script_simuation_IRSA.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This script compares different IRSA implementations with a simulated EAP
% experiment
% The execution of this script requires the following functions:
%      IRSA_convent.m       IRSA conventional implementation
%      IRSA_matrix.m        IRSA matrix-based implementation
%      IRSA_matrix_opt.m    similar to previous, optimized initialization
%      IRSA_matrix_fft.m    similar to previous, matrix products with FFT
%      IRSA_matrix_fft_fast.m similar, optimum alpha, convergence check
%      RSLSD_iter.m         iterative version RSLSD, equivalent to IRSA
%      RSLSD_inf.m          RSLSD at convergence (infinite number of iter.)
% And additionally, a response used for the simulation
%      response_30_60_subj1.mat
% It requires signal processing toolbox
% For Octave users, run:
%          pkg load signal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear,clc;

%% Configuration parameters for simulation
fs=14700;          % sampling rate 14700 Hz
NOISE_GAIN=2;      % noise level
%J=round(1000e-3*fs);   % length of the response 14700, for 1 second
J=round(200e-3*fs);    % length of the response: 2940 for 200 ms
isi_min=30e-3; % 30 ms
isi_max=60e-3; % 60 ms
%n_stim=15200;  % number of stimuli 15200
n_stim=500;    % number of stimuli 200
N_iter=20;    % number of iterations
alpha=0.05;    % convergence parameter
OUTPUT_IRSA=1; % flag for reporting algorithm evolution
CONVERG_CRITERION=120; % convergence criterion for IRSA_matrix_fft_fast, 120 dB

%% Stimulation sequence
isi=rand(n_stim,1)*(isi_max-isi_min)+isi_min; % random distribution of isi, uniform
stim_ind = round(cumsum(isi)*fs); % indexes of stimulus start
N=max(stim_ind)+fs; % number of samples
s=zeros(N,1);
s(stim_ind)=1;

%% Response to be used in simulation (response to be estimated)
load('response_30_60_subj1.mat');
x(J+1)=0; x=x(1:J); % this guarantees a response length equal to J
t_plot=(0:(J-1))/fs;
figure(1)
semilogx(t_plot*1000,x); xlabel('time (ms)'); ylabel('amplitude');
title('Response to be estimated (time log-scaled)');
grid on; xlim([0.1 1000*J/fs]);
figure(2)
plot(t_plot*1000,x); xlabel('time (ms)'); ylabel('amplitude');
title('Response to be estimated (time in linear scale)');
grid on; xlim([0 1000*J/fs]);

%% Simulation of EEG: y=s*x+noise
y0=filter(x,1,s);
y=y0+randn(size(y0))*std(y0)*NOISE_GAIN;
SNR_EEG=10*log10(var(y0)/var(y-y0));
t_plotN=(0:(N-1))/fs;
```

```
figure(3)
plot(t_plotN,y); xlabel('time (s)'); ylabel('amplitude');
title(sprintf('EEG     (SNR-EEG: %.2f dB)',SNR_EEG))
figure(4)
plot(t_plotN,y); xlim([0 1])
xlabel('time (s)'); ylabel('amplitude');
title(sprintf('EEG (first second)     (SNR-EEG: %.2f dB)',SNR_EEG))

%% Deconvolution (with predefined alpha and number of iterations)
fprintf('Running conventional IRSA....\n')
[x1,t_run1]    = IRSA_convent(y,stim_ind,N_iter,J,alpha,OUTPUT_IRSA);
fprintf('Running matrix-based IRSA....\n')
[x2,t_run2]     = IRSA_matrix(y,stim_ind,N_iter,J,alpha,OUTPUT_IRSA);
fprintf('Running matrix-based IRSA, optimized initialization....\n')
[x3,t_run3] = IRSA_matrix_opt(y,stim_ind,N_iter,J,alpha,OUTPUT_IRSA);
fprintf('Running matrix-based IRSA, FFT-based matrix product....\n')
[x4,t_run4] = IRSA_matrix_fft(y,stim_ind,N_iter,J,alpha,OUTPUT_IRSA);
fprintf('Running RSLSD iterative (geometric series of matrices)....\n')
[x5,t_run5]      = RSLSD_iter(y,stim_ind,N_iter,J,alpha,OUTPUT_IRSA);
% Deconvolution at convergence
fprintf('Running matrix-based IRSA, fast implementation....\n')
[x6,t_run6] = IRSA_matrix_fft_fast(y,stim_ind,10000,CONVERG_CRITERION,J,OUTPUT_IRSA);
fprintf('Running RSLSD at convergence (direct RSLSD for infinite iterations)....\n')
[x7,t_run7] = RSLSD_inf(y,stim_ind,J);
fprintf('\n--------END OF IRSA/RSLSD ALGORITHMS---------------\n\n')

%% Results: figure
figure(5)
semilogx(t_plot*1000,[x+4.5 x7+3.5 x6+3 x5+2 x4+1.5 x3+1 x2+0.5 x1 ]);
xlabel('time (ms)'); ylabel('amplitude');
title('Estimated responses');
legend('Response','RSLSD-convergence','IRSA-fast','RSLSD-iter','IRSA-matrix-fft',...
    'IRSA-matrix-opt','IRSA-matrix','IRSA-conventional');
legend('Location','eastoutside');
grid on; xlim([1 1000*J/fs]);

%% Results: difference among responses
% SNR using real response as reference
SNR1_a=10*log10(var(x)/var(x1-x)); SNR2_a=10*log10(var(x)/var(x2-x));
SNR3_a=10*log10(var(x)/var(x3-x)); SNR4_a=10*log10(var(x)/var(x4-x));
SNR5_a=10*log10(var(x)/var(x5-x)); SNR6_a=10*log10(var(x)/var(x6-x));
SNR7_a=10*log10(var(x)/var(x7-x));
% SNR using RSLSD-converg as reference
SNR1_b=10*log10(var(x)/var(x1-x7)); SNR2_b=10*log10(var(x)/var(x2-x7));
SNR3_b=10*log10(var(x)/var(x3-x7)); SNR4_b=10*log10(var(x)/var(x4-x7));
SNR5_b=10*log10(var(x)/var(x5-x7)); SNR6_b=10*log10(var(x)/var(x6-x7));
% SNR using IRSA-conventional as reference
SNR2_c=10*log10(var(x1)/var(x2-x1)); SNR3_c=10*log10(var(x)/var(x3-x1));
SNR4_c=10*log10(var(x1)/var(x4-x1)); SNR5_c=10*log10(var(x)/var(x5-x1));
SNR6_c=10*log10(var(x)/var(x6-x1)); SNR7_c=10*log10(var(x)/var(x7-x1));
fprintf('SNR of estimated responses, using real response as reference:\n')
fprintf('  IRSA-conv:      %.2f dB\n  IRSA-matr:      %.2f dB\n  IRSA-matr-opt:  %.2f dB\n',...
    SNR1_a,SNR2_a,SNR2_a);
fprintf('  IRSA-matr-fft:  %.2f dB\n  RSLSD-iter:     %.2f dB\n',SNR4_a,SNR5_a);
fprintf('  IRSA-matr-fast: %.2f dB\n  RSLSD-converg:  %.2f dB\n\n',SNR6_a,SNR7_a);
fprintf('SNR of estimated responses, using RSLSD-converg response as reference:\n')
fprintf('  IRSA-conv:      %.2f dB\n  IRSA-matr:      %.2f dB\n  IRSA-matr-opt:  %.2f dB\n',...
    SNR1_b,SNR2_b,SNR2_b);
fprintf('  IRSA-matr-fft:  %.2f dB\n  RSLSD-iter:     %.2f dB\n',SNR4_b,SNR5_b);
fprintf('  IRSA-matr-fast: %.2f dB\n\n',SNR6_b);
fprintf('SNR of estimated responses, using IRSA-convent response as reference:\n')
fprintf('  IRSA-matr:      %.2f dB\n  IRSA-matr-opt:  %.2f dB\n',SNR2_c,SNR2_c);
fprintf('  IRSA-matr-fft:  %.2f dB\n  RSLSD-iter:     %.2f dB\n',SNR4_c,SNR5_c);
fprintf('  IRSA-matr-fast: %.2f dB\n  RSLSD-converg:  %.2f dB\n\n',SNR6_c,SNR7_c);
fprintf('Total execution time:\n')
fprintf('  IRSA-conv:      %.5f s\n  IRSA-matr:      %.5f s\n',t_run1,t_run2);
fprintf('  IRSA-matr-opt:  %.5f s\n  IRSA-matr-fft:  %.5f s\n',t_run3,t_run4);
fprintf('  RSLSD-iter:     %.5f s\n  IRSA-matr-fast: %.5f s\n',t_run5,t_run6);
fprintf('  RSLSD-converg:  %.5f s\n\n',t_run7);

return
```