In: (B+H)CI: The Human in Brain-Computer Interfaces and the Brain in Human-Computer Interaction. Desney S Tan, Anton Nijholt (eds.)

MATLAB-Based Tools for BCI Research

Arnaud Delorme[1,2,3], Christian Kothe[4], Andrey Vankov[1], Nima Bigdely-Shamlo[1], Robert Oostenveld[5], Thorsten Zander[4], Scott Makeig[1]

arno@ucsd.edu, christiankothe@googlemail.com, avankov@ucsd.edu, nima@sccn.ucsd.edu, r.oostenveld@gmail.com, thorsten.zander@mms.tu-berlin.de, smakeig@ucsd.edu

1. Swartz Center for Computational Neuroscience, Institute for Neural Computation, University of California San Diego, La Jolla CA, USA

2. Université de Toulouse, UPS, Centre de Recherche Cerveau et Cognition, Toulouse, France

3. CNRS, CerCo, Toulouse, France

4. Department of Psychology and Ergonomics, Chair Human-Machine Systems, Berlin Institute of Technology, Berlin, Germany

5. Donders Institute for Brain, Cognition and Behaviour, Radboud University Nijmegen, Nijmegen, The Netherlands

Abstract

We first review the range of standalone and MATLAB-based software currently freely available to BCI researchers. We then discuss two MATLAB-centered solutions for realtime data streaming, the environments FieldTrip (Donders Institute, Nijmegen) and DataSuite (DataRiver, Producer, MatRiver) (Swartz Center, La Jolla). We illustrate the relative simplicity of coding BCI feature extraction and classification under MATLAB (The Mathworks, Inc.) using a minimalist BCI example, and then describe BCILAB (Team PhyPa, Berlin), a new BCI package that uses the data structures and extends the capabilities of the widely used EEGLAB signal processing environment.

Introduction

Brain Computer Interface (BCI) systems and algorithms allow the use of brain signals as volitional communication devices or more generally create some sort of useful interconnection between the operation of a machine system and the brain activity of a human or animal subject using, engaged with, or monitored by the system. Because of its portability, most BCI systems use electroencephalographic (EEG) signals recorded from one or more scalp channels. Although most of the approaches we review here are also applicable to single-channel recordings, we will focus on sfotware for processing multi-channel EEG data in the MATLAB computing environment (The Mathworks, Inc., Natick MA), a widely available commercial platform-independent numerical computing and visualization software environment. Although, MATLAB applications are rarely used outside of research environments, they offer a valuable tool for developing, prototyping, and testing BCI approaches.

While freely available MATLAB-compatible software alternatives exist (e.g., Octave, see www.gnu.org/software/octave) and alternative open-source software is slowly emerging (www.sagemath.org), MATLAB is presently used in most research centers and is widely considered the tool of choice for developing and, often, applying computational methods in cognitive neuroscience and beyond. While early versions of MATLAB were much slower than compiled versions of the same code, the most recent version of MATLAB has more than doubled in speed, a fact that increasingly makes MATLAB a suitable environment for realtime processing. As of its 2009 release, MATLAB processes may also make use of multiple cores. MATLAB also sells a dedicated package, the Realtime target, designed to facilitate real-time operations.

Several requirements for a research BCI software development environment arise from the demands of the BCI research field:

- *Flexibility.* BCI is an active and rapidly advancing field. Thus, any BCI environment not supporting development and testing of more advanced uses than those initially anticipated will necessarily be of limited value. BCI software environments should therefore allow, invite, and facilitate flexibility in methods extension and re-definition.
- *Ease of Use.* BCI software users include psychologists, human factors experts, human interface designers, signal processing engineers, computer scientists, and mathematicians. All these users cannot be expected to have comprehensive knowledge of the mathematical and neurophysiological bases of BCI operation. While lack of relevant scientific background among a BCI project team might impact their productivity, BCI software environments may minimize or at least mitigate serious errors and misunderstandings by establishing and documenting best practices, providing reasonable default values, flagging and detailing mistakes in usage, and by making common tasks simple to perform.
- *Effciency.* The choice of algorithms that can be applied under given conditions is often determined by the computation time required and available. Therefore, the computational efficiency of the BCI environment is a critical element. Moreover, prototyping and testing of new methods and applications itself should be efficient,

because successful designs will typically require many iterations to perfect. Thus, the BCI environment should allow users to quickly update and test new designs.

• *Performance.* Since current BCI system performance levels are often at best close to the lower boundary of what is considered practical, inference performance, or the accuracy of the predictions, is a most critical aspect of BCI system design. Higher levels of performance are reached by newer state-of-the-art methods whose inventors may not have the resources to perform extensive testing. Ideally, therefore, BCI software environments should include measures and methods for fairly evaluating BCI system performance.

• *Robustness.* BCI research often involves making empirical estimates about the performance of a given BCI design from limited amounts of training and test data, making the problem of overfitting acute and ever present. Because of this, current designs may not adequately model and compensate for the massive diversity and non-stationarity of brain EEG signals. Lack of adequate training and testing data means that BCI systems should have a tendency to work best in the precise subject situations and contexts in which they were developed, and may fail to prove robust as the recording situation or context changes. Thus, performance estimates based on limited testing data are almost always optimistic. Yet dependability of BCI solutions must be an important goal if BCI systems are to find uses outside the laboratory. Ideal BCI software environments should therefore facilitate routine collection of relatively large amounts of training and test data.

The three main components of a BCI system are data streaming and online data processing, plus delivery of user feedback. Data streaming includes channel selection, data filtering and buffering, and extracting epochs in real time based on event presentation. Online data processing involves data preprocessing followed by feature extraction and classification. User feedback involves selection and promotion of desired user interactions based on classification results. Feedback methods of choice depend on the specific application and will not be dealt with here. Below, we discuss first data streaming and then online data processing.

Data streaming

Processing of EEG data in (near) real time in BCI software applications requires, first and foremost, access to the data. Data acquired by an acquisition system must therefore first be streamed into the BCI processing pipeline. Currently, there is no fundamental problem in reading data acquired by a digital recording system in near real time using any general purpose programming language including MATLAB. Many EEG acquisition systems provide some way to interface custom software to their output EEG data stream. For example, Biosemi and TMSI both offer a dynamically-linked library (dll) for interfacing with the hardware, while BrainProducts provides the specification of a TCP protocol under which data can be streamed over a network. Under MATLAB, it is possible to directly interface this EEG data stream by direct calls to DLL routines, by interfacing acquisition cards using the RealTime Workshop (The Mathworks), or by using TCP/IP and the Instrument Control Toolbox (The Mathworks). MATLAB may collect either

one sample or one block of data at a time, and then populate data blocks into a larger data matrix of dimensions channels-by-samples. This data matrix may then be processed under MATLAB using feature extraction and translation algorithms.

However, since MATLAB is a single-threaded application, collecting the continuously streaming data *and* processing it in near-real time may be challenging. Imagine acquiring samples of the EEG data stream and then performing a CPU-intensive computation on those samples, e.g. overlapping fast Fourier transforms (FFTs). During the time that MATLAB requires to compute each FFT, new data arriving in the EEG stream from the acquisition system may be ignored. To allow for full control of the timing of a processing pipeline in MATLAB, incoming data must be buffered to avoid gaps whenever repeated computation is taking place. While the low-level TCP/IP network stack of the operating system will buffer the data for some time, the duration that the data remains in the network stack buffer cannot be guaranteed.

Therefore, instead of having MATLAB itself read one sample at a time from the data stream, another standalone application or thread should read the incoming samples and copy them into a fixed-length or data-adaptive circular buffer. MATLAB can then read new data from this buffer at any time appropriate, e.g., after completion of each computation. Successively separating the buffering of the data stream from computation on segments of that data stream depends on a having fast interface between MATLAB and the buffering software, so that little time is lost in copying data from the buffer into MATLAB memory. In the next sections we present two MATLAB-centered solutions that use this approach: FieldTrip and DataSuite. FieldTrip aims only to provide a usable interface to a single online data stream, while DataSuite in addition allows synchronization of dissimilar data streams, including streams output by online computational clients running on the same or different machines on the network, plus integrated, flexible, and if desired distributed stimulus control.

FieldTrip. The FieldTrip toolbox (R. Oostenveld, www.ru.nl/neuroimaging/fieldtrip) for EEG/MEG analysis under MATLAB provides an open-source implementation of a realtime data buffering scheme. The FieldTrip buffer is implemented as a network transparent TCP server, which allows the acquisition client to stream EEG data to it sample by sample or in small blocks, while at the same time any data that is present in the buffer can be retrieved and processed by another application. The buffer is implemented as a multi-threaded application in C/C++, allowing multiple clients to connect simultaneously to read/write data and event codes.

The FieldTrip buffer may be used more generally to communicate between separate applications. One application program is responsible for data acquisition, writing the data (and optionally also event codes) to the buffer. Another application can connect to the server to read some of the data and event codes (typically, the most recent), and may optionally also write new event codes (e.g., as the output of a classification algorithm) into the same buffer. Source code for the buffering can be integrated into any EEG/MEG acquisition or analysis system, first writing the header information and describing the number of channels and sampling frequency, then delivering the stream of data and/or event codes.

The TCP protocol controls reading and writing to the buffer and can issue a flush/empty command when data collection is restarted.

The buffer code is compiled into a MATLAB 'mex' file. This allows processing of small segments of streaming EEG data under MATLAB while incoming new data is buffered in a separate thread. Since the buffer allows multiple concurrent read connections, multiple MATLAB clients can connect to it, each analyzing a specific aspect of the data concurrently. The MATLAB mex file can also be used to access a remote buffer linked to the acquisition software running as a separate program, possibly even on a separate computer, to instantiate a local buffer linked to the MATLAB process as a separate thread.

DataSuite: DataRiver and MatRiver. The DataSuite environments (www.sccn.ucsd.edu/wiki/DataSuite) form a distributed data acquisition, synchronization, online processing, and stimulus delivery system based around DataRiver (A. Vankov), a unique data management and synchronization real-time engine. DataRiver is based on a real-time data management core, previously developed for the ADAPT data acquisition and analysis system and language [17]. Producer (A. Vankov) is a DataRiver client application for flexibly controlling stimulus presentaton using an original scripting language. MatRiver (N. Bigdely-Shamlo), described below, is a MATLAB client toolbox for DataRiver. Data acquired by independent devices are by definition asynchronous, even when they are acquired at the same nominal sampling rate, because of the independent clocks typically used to pace data acquisition. Moreover, sampling rates for different data sources can differ significantly: while EEG is typically sampled between 250 Hz and 2000 Hz, concurrent body motion capture or button press data, for example, may be sampled at much lower rates. Another major source of time delays is data acquisition hardware buffering to ensure regularity of the data samples. For data acquired through an IP socket connection, network delays can also be significant. Finally, Windows (or any other multi-user) operating system itself introduces variable delays in processing of asynchronous streams through its pre-emptive multitasking – in a multitasking scheme, typically data are processed only when the corresponding thread is activated, not when data become available.

DataRiver was developed to solve these synchronization issues. DataRiver is a flexible and universal system for high precision synchronization of data streams, providing a dynamic, near real-time mechanism for synchronizing concurrent data streams with designed and tested precision better than 2 ms on current workstations. The flexibility of DataRiver derives from its modular design – data output by a variety of hardware devices are handled by specialized device drivers that convert each of them into a device-independent data stream. Those data streams are then continuously merged together in real time into a data "river." DataRiver device drivers, currently available for several types of data input devices and systems, allow ready development of a wide range of interactive experimental paradigms in a wide variety of application environments. Data in incoming data streams can be used in real time by "stream recipient" modules for recording, online data processing, and/or stimulus control. DataRiver has built-in support for a real-time data exchange with one or more remote computers in a local area network (LAN), allowing a distributed, cooperative experimental

environment (Figure 1). New DataRiver routines can easily be added at any time, ensuring expandability to meet evolving research goals.

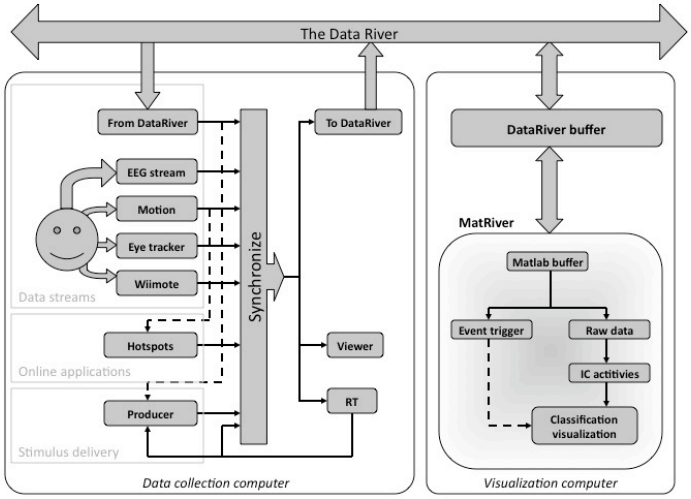


Figure 1. DataSuite data flow. Two computers each running an instance of DataRiver are represented. One acquires data (left); the other (right) uses MatRiver to perform data classification and feedback visualization. Dashed lines indicate control signals.

EEGLAB. A general offline analysis environment for EEG and other electrophysiological data is EEGLAB [14] (www.sccn.ucsd.edu/eeglab), an interactive menu-based and scripting environment for processing electrophysiological data based under MATLAB. EEGLAB provides command line and interactive graphic user interface (GUI) allowing users to flexibly and interactively process their high-density electrophysiological data (up to several hundred channels) or other dynamic brain data time series. Its functions implement several methods of electroencephalographic data analysis including independent component analysis (ICA) [18, 22] and time/frequency analysis [23]. EEGLAB has become a widely used platform for processing biophysical time series and sharing new techniques. At least 28 plug-in functions have been implemented by a variety of user groups. Both MatRiver (described below) and BCILAB (described later) use the EEG dataset structure of EEGLAB. Thus BCI applications written in either environment may make direct use of the many EEGLAB data processing and visualization functions.

MatRiver. The MatRiver MATLAB toolbox includes a MATLAB DataRiver client optimized for real-time data processing, buffering and visualization with emphasis on EEG analysis. It calls DLL functions under Windows OS to communicate with DataRiver and provides a pipeline for EEG pre-processing and classification. In addition to performing common EEG processing steps such as channel selection, re-referencing, frequency filtering and

linear spatial filtering (ICA [18] or other linear models), MatRiver includes simple-to-use routines for dynamic noisy 'bad' channel detection and compensation (taking into account the ICA source model if any). The preprocessed activities of channels or independent components (ICs) are accumulated in MATLAB and may be used for event-based classification or continuous visualization of derived EEG features such as alpha band power.

Event-based EEG classification is facilitated in MatRiver using MATLAB callback functions that are executed at predefined latencies after selected events (triggers). This architecture allows for use of any classifier function accessible in MATLAB, for example from other toolboxes such as BCILAB (described below). Since MatRiver uses MATLAB timers running in the background for real-time processing, it operates in a non-blocking manner – the MATLAB command line stays available throughout the the session, allowing for interactive exploration of incoming data. Online sessions can also be simulated in MatRiver using previously recorded data.

MatRiver is optimized for speed of computation and display; EEG preprocessing and most event-related data classifications can be performed in less than 10 ms on contemporary (2009) hardware. Also, continuous visualizations of derived EEG features (for example, alpha band power) may be rendered at more than 19 frames per second using the Open-GL based Simulink 3-D (The MathWorks, Inc.). The computer gaming industry generally considers screen response latencies of less than 80 ms to be imperceptible for human subjects. MatRiver can thus achieve comparable or better response latency in a wide range of applications. Visualization in MatRiver thus complements the C++-based DataSuite stimulus delivery environment ('Producer') optimized for real-time use with DataRiver. Producer clients may also be used to visualize results of MatRiver computations that are merged via MatRiver routines to the ongoing data river.

Other solutions: BCI2000, OpenViBE, and other packages also allow performing limited processing under MATLAB. rtsBCI in BIOSIG uses MATLAB Simulink and the RealTime Workshop to interface ADC cards. Similarly, the g.tec company uses MATLAB Simulink for high-speed online processing via specially-developed hardware interrupt-controlled drivers. These approaches are not further discussed here.

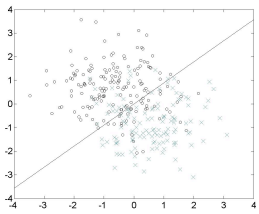
Online data processing

Online BCI processing often consists of first a BCI-specific portion involving custom signal processing and/or feature extraction for which there is already quite a large palette of published algorithms, followed by a generic machine learning / inference portion,

A
```
n        = 300;                                  % Number of samples
d        = 10;                                   % Number of features
labels   = sign(randn(1,n));                     % Labels -1 and 1
data     = [randn(d,n) + 0.5*randn(d,1)*labels]; % Data (1 distributions' distance)

for i = 1:10                                     % 10-fold cross-validation
    tst = [round(n/10)*(i-1)+1:round(n/10)*i];   % Test indices
    trn = logical(ones(1,n)); trn(tst) = 0;      % Train indices

    w = inv(cov(data'))*data(:,trn)*labels(trn)'; % Train LDA
    c = w'*data(:,tst);                          % Test LDA
    p(i) = sum(sign(c)==labels(tst))/length(tst); % Compute percentage correct
end

fprintf('Perf.: %2.1f%%(+-%1.1f)\n',100*mean(p),100*std(p)); % Result (~95% accuracy)
```

Figure 2. A. Minimal MATLAB code for training and testing a simple LDA classifier and performing ten-fold cross-validation. B. The test data versus LDA solution hyperplane in the first two dimensions.

B



for which many toolboxes and simple yet powerful algorithms like Linear Discriminant Analysis (LDA) [19] are available. In view of the wide range of available tools, flexibly prototyping custom data processing and classification methods is a main reason to use MATLAB for BCI research applications.

A minimalistic BCI script using native MATLAB code. MATLAB itself allows easy prototyping of complex algorithms. For instance, the implementation of LDA projection requires 286 lines of C++ code in the OpenViBE toolbox, whereas in MATLAB it can essentially be implemented as the single line >> *result = sign(w'*x-b), x* being the data*, w* the weights and *b* the bias factor. This is one reason why many new computational methods are tested under MATLAB before implementing them in a more structured application-oriented language. For example, MATLAB functions can be used directly to perform learning with and rigorous testing of Linear Discriminant Analysis (LDA), using only simple matrix manipulation [19]. The sample code in Figure 2 above creates two classes of Gaussian-distributed data and then performs training and testing. The script performs 10 fold cross-validation (10 training repetitions on 90% of the data; testing on the remainder) using LDA, and returns mean and std. dev. detection classification accuracy.

MATLAB scripts may also perform more advanced classification. For instance, the Common Spatial Pattern (CSP) algorithm for oscillatory data processing [6] is used in many BCI systems. Implementations of CSP often involve manually tuning its frequency filter and time window. Methods to automatically adapt these parameters exist (e.g., Spec-CSP [20]) but are significantly more difficult to implement. Figure 3 below shows a minimalistic BCI script performing CSP classification that would require thousands of lines of C or C++ code. Despite its simplicity, it can perform online BCI control whose performance may rival that of much more complex BCI software.

Training BCI function

```
% [S,T,w,b] = train_bci(Raw-Signal, Sample-Rate, Markers,
%     Epoch-Wnd, Spectral-Flt, Flt-Number, Flt-Length)
function [S,T,w,b] = train_bci(EEG,Fs,mrk,wnd,f,nof,n)

% frequency filtering and temporal filter estimation
[t,c] = size(EEG); idx = reshape(1:t*c-mod(t*c,n),n,[]);
FLT = real(ifft(fft(EEG).*repmat(f(Fs*(0:t-1)/t)',1,c)));
T = FLT(idx)/EEG(idx);

% data epoching, class-grouping and CSP
wnd = round(Fs*wnd(1)):round(Fs*wnd(2));
for k = 1:2
    EPO{k} = FLT(repmat(find(mrk==k),length(wnd),1) + repmat(wnd',1,nnz(mrk==k)),:);
end
[V,D] = eig(cov(EPO{1}),cov(EPO{1})+cov(EPO{2}));
S = V(:,[1:nof end-nof+1:end]);

% log-variance feature extraction and LDA
for k = 1:2
    X{k} = squeeze(log(var(reshape(EPO{k}*S, length(wnd),[],2*nof))));
end
w = ((mean(X{2})-mean(X{1}))/(cov(X{1})+cov(X{2})))';
b = (mean(X{1})+mean(X{2}))*w/2;
```

Test BCI

```
% Prediction = test_bci(Raw-Block, Spatial-Flt, Temporal-Flt, Weights, Bias)
function y = test_bci(X,S,T,w,b)

global B; % B is the buffer
if any(size(B) ~= [length(T),length(S)])
    B = zeros(length(T),length(S));
end
B = [B;X]; B = B(end-length(T)+1:end,:);
y = log(var(T*(B*S)))*w - b;
```

Offline test

```
load data_set_IVb_al_train
flt = @(f)(f>7&f<30).*(1-cos((f-(7+30)/2)/(7-30)*pi*4));
[S,T,w,b] = train_bci(single(cnt), nfo.fs, ...
    sparse(1,mrk.pos,(mrk.y+3)/2),[0.5 3.5],flt,3,200);

load data_set_IVb_al_test
for x=1:length(cnt)
    y(x) = test_bci(single(cnt(x,:)),S,T,w,b);
end

load true_labels
plot((1:length(cnt))/nfo.fs,[y/sqrt(mean(y.*y)); true_y']);
xlabel('time (seconds)'); ylabel('class');
```

R-T

```
y = [];
start(timer('R = get_rawdata; y(end+1) = test_bci(R,S,T,w,b); plot(y(max(1,length(y)-
20):end);', 'InstantPeriod', 0.1));
```

Figure 3. A minimalistic BCI script (C. Kothe). The top function (*train_bci*) performs temporal filtering and training of a CSP filter. The second function (*test_bci*) applies the model to incoming blocks of raw data, and can be used for online processing. The scripts load data (here from the BCI Competition III), perform training and testing, and display results as in Figure 2C. Assuming a function "*get_rawdata*" allows asynchronous data collection (for example to DataRiver or Fieldtrip), the fourth script performs real-time (R-T) classification and displays an evolving time course of classification over the past 2 seconds with a refresh rate of 100 ms. Figure 4 shows the spatial and temporal filters learned and used. For more details see www.sccn.ucsd.edu/minimalistBCI.

Here, the data involved imagined left and right hand movements plus rest periods.The learned test function can also be applied in real time to a new incoming data stream using a MATLAB timer. The only usage guideline for *test_bci* is that whenever the user wants a prediction of the current mental state, he

must feed it all raw EEG samples that have been collected since the previous call to the function.

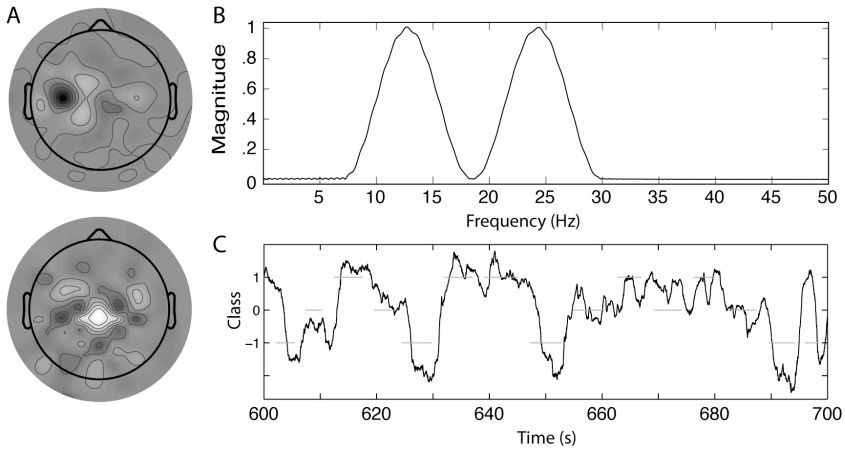


Figure 4. Spatial and temporal filters associated with the code in Figure 3. A. CSP 118-channel spatial filter weights for the best two of six CSP filters used. B. The temporal filter tailored to the BCI data. This filter was learned by the Test BCI script in Figure 3. C. BCI performance over a 100 second window. The black curve indicates the output of the *test_bci* function. Gray plateaus indicate the detected class (1 is imagined left hand movement; 0 is rest; -1 is imagined right hand movement).

The spatiotemporal filters associated with the example from Figure 3 are shown in Figure 4 above. Figure 4C shows the output of the model and the data labels.

BCILAB. BCILAB is a system for easily constructing new BCI systems with a strong focus on advancing the state of the art. BCI systems constructed using BCILAB can be applied online and/or evaluated offline. Many BCI designs are permitted, from the simplest signal processing chain to advanced machine learning systems. BCILAB is designed to become (likely in 2010) a freely available toolbox for the creation, evaluation, and application of BCI systems. BCILAB also provides tools to explore and visualize datasets offline for basic research purposes. As there is no clear boundary between data analysis for BCI and for neuroscience, here BCILAB blends into EEGLAB on which it is built. BCILAB system design is based on three concepts:

• *BCI Detectors*. These are the fundamental component of any BCI system, the actual methods mapping continuous EEG data measures to a control signal. Whereas in environments such as EEGLAB, the primary object of study is the data itself, in BCILAB the primary object of study are BCI Detectors, with one or more Detectors forming a BCI system.

• *Detector components*. BCILAB provides a large collection of components that can be used to construct BCI Detectors. Three categories exist: Signal processing, feature extracting, and machine learning. Custom signal processors and machine learning algorithms can also be implemented by the user, subject to framework

contracts guiding the implementation which are sufficiently general to encompasses most approaches.

• *Detection paradigms*. Detection paradigms are prototypes of commonly re-used Detector designs consisting of multiple BCI components, usually from all three categories.

BCILAB data processing capabilities are reviewed in Table 2.

Signal processing	Feature extraction	Machine Learning Algorithms
• Channel selection • Resampling • Deblinking • Envelope extraction • Epoch extraction • Baseline filtering • Re-referencing • Surface Laplacian filtering [24] • ICA methods (Infomax, FastICA, AMICA) [18, 25] • Spectral filters (FIR, IIR) • Spherical spline interpolation [26]	• Multi-window averaging for detection based on slow cortical potentials [11, 12] • Common Spatial Patterns (CSP) [6] • Spectrally-weighted Common Spatial Patterns [20] • Adaptive Autoregressive Modeling, from BioSig [4]	• Linear Discriminant Analysis (LDA) [19] • Quadratic Discriminant Analysis (QDA) [21] • Regularized LDA and QDA [21] • Linear SVM [7] (implemented using LIBLINEAR) • Kernel SVM [7] (implemented using SVMPerf, with LibSVM fallback) • Gaussian Mixture Models (GMM three methods [[27], [28], [29]], implemented using GMMBAYES) • Variational Bayesian Logistic Regression [30] (contributed by T.Klister), • Deep Restricted Boltzmann Machines [31] (contributed by F.Bachl) • Relevance Vector Machines (RVM) [32] (implemented using SparseBayes).

Table 2. Signal processing, feature extraction, and Machine learning algorithms in the BCILAB/EEGLAB framework.

BCILAB natively implements some default BCI paradigms. These allow the researcher to simply provide data and designate a paradigm name: CSP for imagined movements with LDA [19], Spec-CSP for imagined movements with LDA [20], logarithmic band-power estimates with Hjorth surface Laplacian filter [33], multi-segment averages with LDA (for using the Lateralized Readiness Potential) [11], adaptive autoregressive models on band-pass filtered channels with LDA [4], common spatial patterns for slow cortical potentials [34], multi-band CSP, ICA-decomposed logarithmic band-power estimates, and, as meta-algorithms, feature combinations [35] as well as multi-class classification by panels of experts. These default detection paradigms are massively adaptible. For example, the CSP paradigm for imagined movements can easily be parameterized to measure aspects of mental workload or relaxation. Much BCILAB design work amounts to re-parameterization of existing paradigms for new goals (epoch length, filtering, etc.). In principle, the entire preprocessing chain of the paradigm can be replaced element by element as desired. The ease of adding new components to the toolbox has allowed ready implementation of a variety of methods from EEG and BCI research.

Automated Parameter Search is a particularly convenient feature of the design interface. Instead of a parameter, the special expression *search(…)* can be given, to specify a parameter range. This can be used, for example, to auto-determine the best model parameters, or to regularize a classifier.

The BCILAB Detector Design Interface (DDI) is the primary interface for configuring, training, and evaluating Detectors offline. There are three interface functions covering this area of BCI research. *bci_preproc*, *bci_train*, and *bci_predict* are command line interfaces, though a GUI wrapper for each of them is planned. The data are first preprocessed by *bci_preproc* with the help of EEGLAB functions. *bci_train* then finds the optimal Detector given the paradigm; this Detector function can then be used for online data processing. The *bci_preproc* function can apply customization to the whole flow, from raw data to a final online-ready Detector, on the fly. Real-time use of the toolbox is similar to the minimalistBCI.

When attempting new BCI applications, often not much is known about the nature of the data at hand, and therefore not much about how Detector parameters should be chosen. This is where strong visualization functions can help. It is relatively easy to get a BCI running under MATLAB (cf. *train_bci* in Figure 3), but it involves much more work to visualize the data in time and frequency and to plot scalp maps, tasks which are practical when tuning the parameters of these functions and checking the neurophysiological plausibility of the learned models. BCILAB contains a function (*vis_hyperspectrum*) to display accuracy-coded time/frequency images that encode, for every time/frequency voxel, the cross-validated performance estimates of a CSP Detector, as shown in Figure 5. A variant of it allows inspection of the similarity of optimal filters over time and frequency: similar colors imply similarly successful filters. Another function displays class-colored distributions of slow cortical potentials over time, thus showing at which times the slow potentials for the contrasted conditions become discriminative. These functions allow quick identification of good parameters for Detectors, using spectral power and/or (near DC) SCP classifiers.

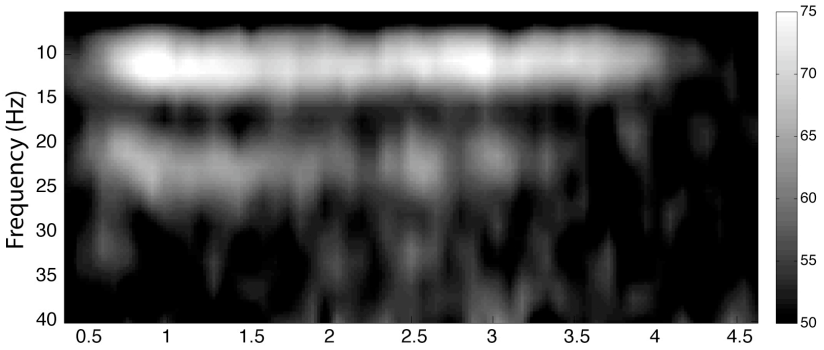


Figure 5: Illustration of estimated accuracy of a CSP-based time/frequency Detector in a two-class imagined movement task [1] using Logistic Regression to classify every time/frequency voxel during the 5 seconds after stimulus presentation. Here the spectral time windows were Hann windows with 90% overlap. Time/frequency estimates were obtained by computing 150-sample FFTs. The cross-validated performance of a CSP+LDA classifier was then estimated and mapped to an oversamples and interpolated color (or here greyscale) image.

For instance, Figure 5 (above) was used to select an optimal frequency filter for the model implemented in the minimalistic BCI code (Figure 3). A collection of additional visualization functions display internal properties of trained Detectors, for example showing linear classifier weights as scalp map plots using EEGLAB plotting functions,

To summarize, the BCILAB toolbox can fit the online processing slot in most research BCI environments. For example, it can be linked as processing node into DataSuite, FieldTrip, BCI2000, or OpenViBE systems, or be connected to a proprietary acquisition and stimulus presentation system. As it is fully scriptable, when MATLAB is available it can in principle also be embedded into research prototype systems for use outside the laboratory. In addition to functioning as processing block, BCILAB has a user interface for developing, customizing, training, evaluating, and tuning Detectors using a array of methods likely to grow wider before release. Finally, BCILAB can also serve as a tool to explore discriminative questions about data, and can be viewed as a plug-in extension to EEGLAB for this purpose.

Other MATLAB BCI classification tools. Various classification methods may be implemented using functions in the commercial MATLAB add-on toolboxes. The *classify* function of the MATLAB Statistics toolbox performs LDA, its variants Quadratic Discriminant Analysis (QDA) [21], and classification using Mahalonobis distance (MDA). The *hmmtrain* function allows training of Hidden Markov Models (HMM). The Neural Network toolbox adds Multilayer Perceptrons. The Bioinformatics toolbox adds suport vector machine (SVM) and K-means classification, and also contains a user-friendly and versatile function *crossvalind* to produce cross-validation indices, plus a function *classperf* to store and accumulate classifier performances and statistics. It is beyond the scope of this chapter to review all the MATLAB commercial and free tools available to perform classification and data processing, especially since these tools are in constant evolution. Instead, in Table 1 we list some commonly used tools (as of early 2010) for classifying data.

Package	License	Content
BCILAB	GPL	See table 2.
g.BSanalyse	Commercial	LDA, Minimum Distance Classifier (MDC), QDA, MutiLayer Perceptron (MLP), Radial Basis Function (RBF), Kmean
BIOSIG	GPL	Various LDA, QDA/MDA, Regularized Disciminant Analysis (RDA), MDC, Partial Least Square (PLS), RBF, various SVM and bayesian classifiers.
NMLT	GPL	This toolbox is associated with FieldTrip. Currently in development.
MATLAB	Commercial	LDA; Minimum Distance Classifier, QDA, HMM (Statistic Toolbox); MLP (Neural Network Toolbox), SVM, Kmean (Bioinformatics Toolbox).
CVX	GPL	Logistic regression, SVM, Gaussian process regression.
GPML toolbox	GPL	Gaussian process classification.

LibSVM	GPL	SVM, supports multi-class.
MLOSS	Mostly GPL	Various Machine Learning Open Source Software.

Table 1. Free and commercial classifiers running under MATLAB. The double horizontal line separates the EEG-tailored tools from more general all-purpose classification tools. GPL refers to software freely available under the Gnu Public Licence.

Other BCI software also offers comprehensive solutions on different platforms and operating systems. We review them briefly below.

Other existing MATLAB and non-MATLAB BCI tools

• *BioSig* [3] (www.biosig.sourceforge.net) emerged from the original Graz BCI system as a MATLAB/Octave toolbox. This software (as of version 2.31), supports a wide range of functionality in statistics and time-series analysis, with a focus on online biosignal processing. It includes the most complete adaptive auto-regression implementation [4] as well as blind source separation [5], Common Spatial Patterns classification [6], and code to perform classification according to a variety of methods including kernel Support Vector Machines [7], as well as basic cross-validation methods for estimating classifier performance (see Table 1 for the list of classification algorithms suported). Most implemented features are linked to full paper references. Online and realtime operation is implemented in "rtsBCI", a module based on Simulink and the RealTime Workshop from The Mathworks. Using the BioSig software tends to require strong programming abilities and in-depth knowledge of code internals.
• *OpenViBE* (www.irisa.fr/bunraku/OpenViBE) is a relatively new project developed in France quite different from BioSig. The current implementation (version 0.4.0) is a clean-slate approach to BCI written in C++ with a focus on online processing and virtual reality integration. Most of OpenViBE is a visual programming toolkit for low-level signal processing and higher-level classification, implemented via building blocks that can be graphically combined, making it an ambitious programming project. OpenViBE has been used together with a relatively versatile machine learning library (BLiFF++). OpenViBE also contains a module to run MATLAB code in real-time, although currently this only deals with real-time data processing (or offline streaming of data). When eventually completed, OpenViBE could become one of the easier-to-use tools for BCI signal analysis.
• *BCI2000* [8] (www.bci2000.org) is also a native C++ implementation headed by Gerwin Schalk at the Wadsworth Center (Albany NY) that focuses on online data collection, processing, and feedback. BCI2000 is a complete research BCI toolkit including a data recording module with integrated processing, a simple stimulus presentation module, an operator control module, and a feedback module. It is a robust and sufficient tool for testing simple and established BCI approaches such as 'P300'-based spellers [9] and mu rhythm demodulation [10]. In theory, it can also handle more complex workloads such as adaptive spatiotemporal filtering, non-linear classification, and BCI performance evaluation. The BCI2000 software is reliable and has a large user base and several extensions are available, including

an implementation of the Common Spatial Patterns algorithm. BCI2000 has support for executing functions in MATLAB in real time, and includes some basic functions for offline processing of data from disk in MATLAB..

• *The BBCI Toolbox* (www.bbci.de) is an in-house, closed-source, BCI toolkit developed by Berlin Institute of Technology, the Fraunhofer FIRST institute and the Charité University Medicine (Berlin, Germany). Though not much is known about its internal structure, there is reason to believe that it is flexibly designed. Its functionality, judging by the authors' publications (i.e. [11, 12]), may make it the most complete BCI toolbox written to date. Its feedback system is currently being rewritten in Python and has been released as open source software [13].

• *g.BSanalyze* (www.gtec.at) is a commercial biosignal analysis toolbox developed by the Austrian company g.tec and written in MATLAB. A feedback application based on MATLAB's Simulink can be obtained separately. Its documentation implies that its level of BCI functionality can be compared to that of BioSig plus a large subset of EEGLAB [14]. Clearly a massive amount of work went into optimizing its GUI design and usability. However, the most advanced classifiers and feature extractors are currently not yet implemented.

• *Other projects*. The EU funded Tobi project (www.tobi-project.org) is a multi-million euro project that includes both the Graz BCI team and the BBCI teams. It is currently developing a common software platform for BCI operation and calibration. The Dutch government funded Braingain project (www.braingain.nl) is supporting the development of real-time FieldTrip (described above) and BrainStream (www.brainstream.nu), a simplified MATLAB-based BCI environment for non-programmers.

Conclusion

BCI research now underway has at least three objectives. First, much BCI research attempts to identify efficient and low-latency means of obtaining volitional control over changes in EEG features, thus forming 'mental signals' usable for BCI communication (see Chapter 9 by Zander et al.). A second class of BCI systems attempt to use modulation of brain activity in response to external stimulation, often by volitional control of user attention. The modulated brain activity, for instance the P300 target stimulus complex [36], is mapped to an artificial control signal such as a speller that detects a characteristic brain dynamic response elicited when an on-screen letter attended by the user is highlighted. A third objective performs passive cognitive monitoring of user state including actions or intention, so as to enhance overall human-system productivity, safety, enjoyment, or equilibrium. Applications in this area are as diverse as alertness monitoring [37], systems to detect user confusion [38], neurofeedback [39], and systems proposed to automatically detect and quench epileptic seizures.

Current BCI technology is quite young, much in flux, and is likely moving toward eventual convergence on robust and flexible mental state inference methods. Real-world BCI applications for healthy or disabled users can be efficiently designed and prototyped using currently available MATLAB tools and

toolboxes, but breakthrough into widely-applicable methods will probably not occur until dry, wireless EEG systems are readily available [40], and more advanced signal processing methods are developed based on more complete understanding of distributed brain processes. The introduction of machine learning techniques from BCI research into cognitive neuroscience may facilitate development of more comprehensive models of brain function. Despite their efficiency and simplicity, many current BCI algorithms such as Common Spatial Patterns (CSP) are not directly based on or interpretable in terms of EEG source neurophysiology. Incorporating advances in understanding EEG and brain function will likely help BCI systems mature and improve in performance.

Finally, although early work in BCI-based communication systems designed for use with 'locked-in' patients took appropriate care to exclude use of potentials arising from muscle activity in normal control subjects, there is no reason that BCI systems need rely on EEG signals alone. Rather, the prospect of using mobile brain/body imaging (MoBI) [41] to model concurrent motor behavior and psychophysiology (including body movements and muscle activities) as well as EEG *and* electromyographic (EMG) data should open up a much wider range of BCI (or perhaps brain/body computer interface) concepts [41]. Components of such systems are already being developed commercially for computer gaming, and will likely be soon applied to much broader classes of human-system interaction research.

References:

1. Kothe, C., *Design and Implementation of a Research Brain-Computer Interface*. 2009, Berlin Institute of Technology: Berlin. p. section 8.2.1.
2. A, S., et al., *Characterization of Four-Class Motor Imagery EEG Data for the BCI-Competition 2005.* 2005.
3. Schlögl, A. and C. Brunner, *Biosig: A free and open source software library for BCI research.* Computer 2008. **41**(10): p. 44-50.
4. Schlögl, A., *The electroencephalogram and the adaptive autoregressive model: theory and applications*. 2000, ISBN3-8265-7640-3: Shaker Verlag, Aachen, Germany.
5. Bell, A.J. and T.J. Sejnowski, *An information-maximization approach to blind separation and blind deconvolution.* Neural Comput, 1995. **7**(6): p. 1129-59.
6. Ramoser, H., J. Müller-Gerking, and G. Pfurtscheller, *Optimal spatial filtering of single trial EEG during imagined hand movement.* IEEE Trans. Rehab. Eng, 1998(8): p. 441-446.
7. Schölkopf, B. and A. Smola, *Learning with Kernels*. 2002, Cambridge, MA: MIT Press.
8. Schalk, G., et al., *BCI2000: a general-purpose brain-computer interface (BCI) system.* IEEE Transactions on Biomedical Engineering, 2004. **51**(6): p. 1034-1043.

9. Sellers, E., et al., *A p300 event-related potential brain–computer interface (BCI): the effects of matrix size and inter stimulus interval on performance.* Biological Psychology, 2006. **73**(3): p. 242-252.
10. Miner, L.A., D.J. McFarland, and J.R. Wolpaw, *Answering questions with an electroencephalogram-based brain-computer interface.* Arch Phys Med Rehabil, 1998. **79**(9): p. 1029-33.
11. Blankertz, B., G. Curio, and K. Müller, *Classifying single trial EEG: Towards brain computer interfacing*, in *Advances in Neural Inf. Proc. Systems (NIPS 01)*, T. Diettrich, S. Becker, and Z. Ghahramani, Editors. 2002. p. 157-164.
12. Blankertz, B., et al. *Single trial detection of EEG error potentials: A tool for increasing BCI transmission rates*. in *Artificial Neural Networks -- ICANN 2002*. 2002.
13. Venthur, B. and B. Blankertz. *A platform-independent open-source feedback framework for BCI systems*. in *4th International Brain-Computer Interface Workshop and Training Course*. 2008.
14. Delorme, A. and S. Makeig, *EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis.* J Neurosci Methods, 2004. **134**(1): p. 9-21.
15. Kothe, C. and T. Zander. *Benchmarking common BCI algorithms for fast-paced applications*. in *Proc. of the 4th Int. BCI Workshop & Training Course*. 2008. Graz, Austria: Graz University of Technology Publishing House.
16. Zander, T., et al. *Team PhyPA: Developing applications for BrainComputer Interaction. BrainComputer Interfaces for HCI and Games*. in *CHI 2008 Workshop*. 2008. Florence, Italy.
17. *Adapt © 1987-2003 and Varieté , © 2000,2001 are property of EEG Solutions LLC, and are used under free license for scientific non-profit research.*
18. Makeig, S., et al., *Independent component analysis of electroencephalographic data*, in *Advances in Neural Information Processing Systems*, D. Touretzky, M. Mozer, and M. Hasselmo, Editors. 1996. p. 145-151.
19. Fisher, R., *The Use of Multiple Measurements in Taxonomic Problems.* Annals of Eugenics, 1936. **7**: p. 179-188.
20. Tomioka, R., et al. *An Iterative Algorithm for Spatio-Temporal Filter Optimization*. in *3rd International BCI Workshop and Training Course*. 2006. Verlag der Technischen Universität Graz, Graz, Austria.
21. Friedman, J., *Regularized Discriminant Analysis.* Journal of the American Statistical Association, 1989. **84**(405): p. 165-175.
22. Makeig, S., et al., *Dynamic brain sources of visual evoked responses.* Science, 2002. **295**(5555): p. 690-4.
23. Makeig, S. and M. Inlow, *Lapses in alertness: coherence of fluctuations in performance and EEG spectrum.* Electroencephalogr Clin Neurophysiol, 1993. **86**(1): p. 23-35.

24. Babiloni, F., et al., *Performances of surface Laplacian estimators: a study of simulated and real scalp potential distributions.* Brain Topogr, 1995. **8**(1): p. 35-45.
25. Palmer, J.A., et al. *Modeling and Estimation of Dependent Subspaces with Non-radially Symmetric and Skewed Densities*. in *7th International Conference on Independent Component Analysis and Signal Separation* 2007. London, UK.
26. Perrin, F., et al., *Mapping of scalp potentials by surface spline interpolation.* Electroencephalogr Clin Neurophysiol, 1987. **66**(1): p. 75-81.
27. Bilmes, J., *Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models.* International Computer Science Institute, 1998.
28. Vlassis, N. and A. Likas, *A Greedy EM Algorithm for Gaussian Mixture Learning*. Neural Processing Letters 15. Vol. 15. 2002: Kluwer Academic Publishers.
29. Figueiredo, M. and A. Jain, *Unsupervised Learning on Finite Mixture Models.* IEEE transactions of pattern analysis and machine intelligence, 2002. **24**(3).
30. Jaakkola, T. and M. Jordan. *A variational approach to bayesian logistic regression models and their extensions*. in *Sixth International Workshop on Artificial Intelligence and Statistics*. 1997.
31. Hinton, G., S. Osindero, and Y. Teh, *A fast learning algorithm for deep belief nets.* Neural Computation, 2006. **18**: p. 1527-1554.
32. Tipping, M., *Sparse Bayesian learning and the relevance vector machine.* Journal of Machine Learning Research, 2001. **1**: p. 211-244.
33. Vidaurre, C. and A. Schlögl. *Comparison of adaptive features with linear discriminant classifier for brain computer interfaces*. in *Engineering in Medicine and Biology Society. EMBS 2008. 30th Annual International Conference of the IEEE*. 2008.
34. Dornhege, G., B. Blankertz, and G. Curio. *Speeding up classification of multi-channel brain-computer interfaces: common spatial patterns for slow cortical potentials*. in *First International IEEE EMBS Conference on In Neural Engineering*. 2003.
35. Dornhege, G., et al., *Combining features for BCI*, in *Proc. Systems (NIPS 02)*, S. Becker, S. Thrun, and K. Obermayer, Editors. 2003. p. 1115-1122.
36. Farwell, L. and E. Donchin, *Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials.* Electroencephalography and Clinical Neurophysiology, 1988. **70**(6): p. 510–523.
37. Jung, T.-P., et al., *Estimating alertness from the EEG power spectrum.* IEEE Trans Biomed Eng, 1997. **44**(1): p. 60-69.
38. Zander, T. and S. Jatzev. *Detecting affective covert user states with passive Brain-Computer Interfaces*. in *ACII 2009*. 2009. Los Alamitos, CA: IEEE Computer Society Press.

39. Birbaumer, N., et al., *Neurofeedback and brain-computer interface clinical applications.* Int Rev Neurobiol, 2009. **86**: p. 107-17.
40. Lin, C.-T., et al., *A noninvasive prosthetic platform using mobile & wireless EEG.* Proceedings of the IEEE, 2008. **96**(7): p. 1167-83.
41. Makeig, S., et al., *Linking brain, mind and behavior.* Int J Psychophysiol, 2009. **73**(2): p. 95-100.