Einführung in Matlab
Teil II SPM Batch Skripte

Glad Mihai und Jörg Pfannmöller

Universität Greifswald
Funktionelle Bildgebung
April 2012

Inhaltsverzeichnis

1 Übersicht

SPM ist ein Programm zur statistischen Auswertung funktioneller MRT oder PET Daten. Die Auswertung kann über die grafische Benutzeroberfläche oder mittels Batch-Skript durchgeführt werden. Bei Gruppenstudien sind die Parameter für die SPM-Datenauswertung für alle Individuen identisch. Deshalb kann die Auswertung mit Batch-Skripten automatisch und sehr zeiteffektiv durchgeführt werden. In diesem Seminar werden die Grundlagen zur Erstellung solcher Batch-Skripte präsentiert. Die hier benutzte Version ist SPM8.

In diesem Tutorium werden wir mit einem Beispieldatenset arbeiten. Alle notwendigen Schritte in einer statistischen Analyse eines fMRT-Datensatzes werden durchgeführt. Für die Vorverarbeitung sind es folgende Schritte:

- Dicom Importierung
- Field Map für mehrere Probanden erzeugen
- Slice Timing
- Realign & Unwarp

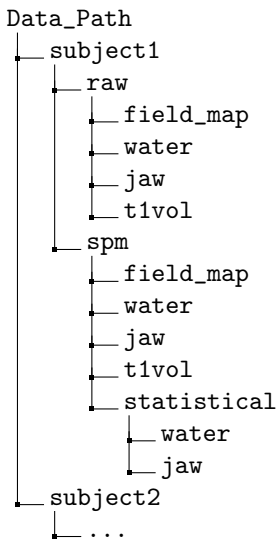
- Coregistration
- Segmentation
- Normalisation of functional data
- Normalisation of structural data
- Smoothing.

Für die 1^{st} Level Analyse:

- fMRI Model Specification
- Model Estimation
- Contrast Manager.

2 Vorverarbeitung

Eine geordnete Pfadstruktur dient zur Erleichterung der Programmierarbeit. Für die hier vorgestellten Skripte ist folgende Pfadstruktur zu verwenden:



`Data_Path` ist das Wurzelverzeichnis, und darunter sind die Zweige für die Probandendaten. Es gibt einen `raw` und einen `spm` Ordner für jeden Probanden. Im `raw` Ordner sind alle DICOM Daten zu finden, geordnet nach Sequenzen. Das Gleiche gilt für den `spm` Ordner, jedoch sind hier alle Dateien im `nifti`-Format angelegt. Im `statistical` Ordner werden alle Daten durch die 1^{st} Level Analyse erzeugt.

Da SPM unter Matlab läuft, muss Matlab gestartet werden. Im Kommandofenster von Matlab den Befehl `spm fmri` eingeben, um SPM zu starten.

2.1 Vorgehensweise

SPM stellt eine Batch-Skript `m`-File Exportierung zur Verfügung. Diese Exportierten `m`-Files sind als Fundament für die eigentliche Programmierung zu gebrauchen. Im Menüfenster von SPM wird durch anklicken des *Batch* Knopfes der Batch Editor geöffnet (Abb. 1).
In dem neu geöffneten Fenster navigiert man zu *SPM* → *Util* und man wählt *Dicom Import* aus, siehe Abb. 2. Die Batch Einträge für DICOM Import werden geladen. Hier könnte man die mit *<-X* gekennzeichneten Einträge füllen. Das wird aber nicht gemacht. Stattdessen werden die unausgefüllten Anweisungen als `m`-file exportiert. Man navigiert zu *File* → *Save Batch and Script* (Abb. 3), und bennent die Datei `dicom_import`. Es werden zwei Dateien erzeugt: `dicom_import.m` und `dicom_import_job.m`.
In der ersten Datei (`dicom_import.m`) werden die Variablen eingegeben, SPM wird initialisiert und die Verarbeitung gestartet. In der zweiten Datei (`dicom_import_job.m`) werden die Einträge die für SPM lesbar sind geschrieben. Die erste Datei greift auf die zweite zu, gibt Informationen weiter und die zweite schickt Informationen an die erste zurück. Wir werden hauptsächlich mit der zweiten Datei arbeiten. Die erste wird umfassend modifiziert um sie an unsere Anforderungen anzupassen.
Diese Bearbeitungssequenz kann genutzt werden um ein Fundament für Batch-Skripte mit beliebigen Funktionen von SPM zu erzeugen.

2.2 Dicom Importierung

Bevor die Vorverarbeitung durchgeführt werden kann, müssen alle Daten ins `nifti`-Format konvertiert werden. In SPM heisst dieser Konvertor *DICOM Import* und ist im Menüfenster zu finden.
Die modifizierten Skripte sind im folgenden angegeben. Im Anschluß an die Skripte werden die einzelnen Skript-Abschnitte diskutiert.

`dicom_import.m`

```
%% DICOM Import script
%  This script converts DICOM files into nifti Files using SPMs
    DICOM
%  Import sequence. It follows a particular directory structure
    which must
%  be adhered to.
%
% Glad Mihai, University of Greifswald, Jan. 2012

%% Clear variables and command window
clear all, clc

%% Specify paths
%  Experiment folder
data_path = 'C:\Users\glad\Documents\Matlab Kurs';

%  Subject folders
subjects = {
            'subject1',...
            'subject2'
            };
%  Sequence folders
```

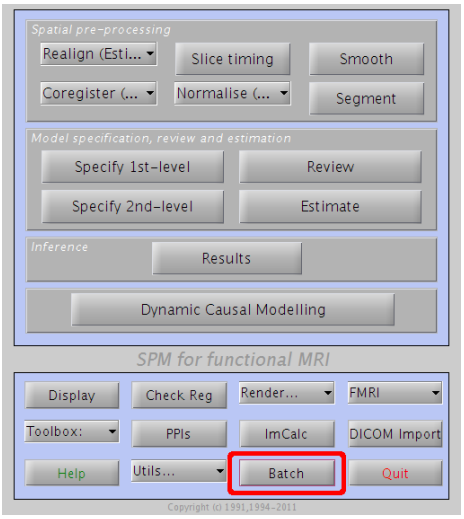


Abbildung 1: Menü-Fenster in SPM.

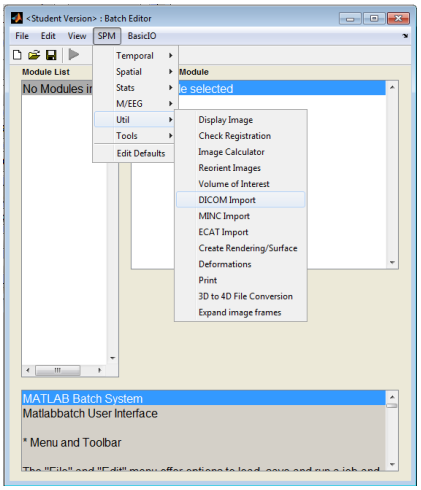


Abbildung 2: Batch Editor mit Menü bei Dicom Import geöffnet.

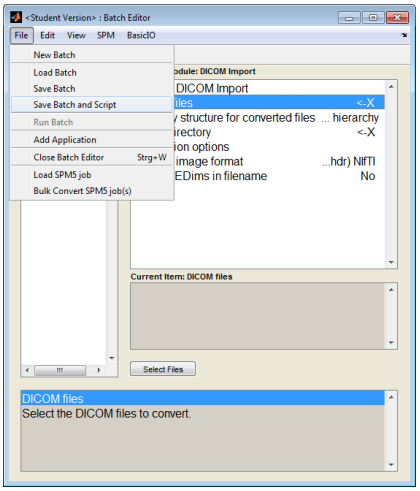


Abbildung 3: Anweisungen als m-File speichern.

```
sequences = {
            '01_gre_field_34_2x_betrag',...
            '02_gre_field_34_2x_phase',...
            'jaw',...
            'wasser',...
            't1vol'
            };

%%  Initialise SPM defaults
spm('defaults', 'FMRI');
spm_jobman('initcfg');

%% Loop to load data from folders and run the job
for i = 1:numel(subjects)

    % load raw data for each sequence without a prompt
    for j = 1:numel(sequences)
        raw = cellstr(spm_select('FPList',fullfile(data_path,...
            subjects{i},'raw',sequences{j}),'^I\w*'));
        outdir = cellstr(fullfile(data_path,subjects{i},...
            'spm',sequences{j}));

        % display which subject and sequence is being processed
        fprintf('Processing subject "%s", "%s" (%s files)\n',...
            subjects{i},sequences{j},sprintf('%d',size(raw,1)));

        % set up SPM-readable structure array
        matlabbatch = dicom_import_job(raw,outdir);
        %% SAVE AND RUN JOB
        %
            ------------------------------------------------------------

        save(fullfile(data_path,subjects{i},'raw',sequences{i},...
          'dicom_import.mat'),'matlabbatch');
        spm_jobman('serial', matlabbatch, '' );
    end
end
```

Algorithmus

Das Wurzelverzeichnis (`data_path`), die Namen der Ordner (`subjects`) und der Sequenzen (`sequences`) werden angegeben. Grundsätzlich muss man die graphische Oberfläche von SPM nicht starten, wenn man ein Batch-Skript ausführen möchte. Notwendig ist eine Initialisierung von SPM. Da wir mit fMRT-Daten arbeiten wird SPM im MRT Modus durch folgende Befehle initialisiert:

```
spm('Defaults','fMRI');
spm_jobman('initcfg');
```

In der Schleife werden alle Rohdateinamen (`raw`) und die Ordner in die die importierten Daten hingeschriebenen werden (`outdir`) eingelesen, bzw. vordefiniert. Mit dem Skript `dicom_import_job` werden dann die ausgesuchten Daten an das Structure Array `matlabbatch` zugewiesen (mehr dazu später). Der Batch-Prozess wird in eine `mat`-Datei gespeichert. Diese kann man im Graphischen Modus von SPM als einen Batch-Job öffnen. Ganz zum Schluss werden mit dem Befehl `spm_jobman` die ausgesuchten Daten konvertiert.
Drei oder mehr Punke '...' bedeuten, dass der Befehl oder die Funktion in der nächsten Zeile weitergeführt wird. Man kann es auch in einer Zeile eingeben,

jedoch verliert diese Schreibweise an Übersicht, vor allem wenn die Anzahl der Probanden groß ist:

```
subjects = {'subject1','subject2','subject3',...,'subject23'};
```

`dicom_import_job.m`

```
% matlabbatch = dicom_import_job(raw,outdir)
% The function takes the list of raw files and output directory and
    passes
% them on to the matlabbatch sturcture array
%
% Glad Mihai, University of Greifswald, Jan. 2012

function matlabbatch = dicom_import_job(raw,outdir)

for i = 1:numel(outdir)
    matlabbatch{i}.spm.util.dicom.data = raw;
    matlabbatch{i}.spm.util.dicom.root = 'flat';
    matlabbatch{i}.spm.util.dicom.outdir = outdir;
    matlabbatch{i}.spm.util.dicom.convopts.format = 'img';
    matlabbatch{i}.spm.util.dicom.convopts.icedims = 0;
end
```

Funktionen in Matlab

Eine Funktion in Matlab ist ein `m`-File, welches als Funktion definiert ist. In diesem Fall ist der Name der Funktion `dicom_import_job`. Dieser Name muss auch die Datei tragen. Eine Funktion kann Variablen annehmen und ausgeben. Hier werden die Variablen `raw` und `outdir` an die Funktion weitergeleitet. Die Funktion gibt die Variable `matlabbatch` zurück.

```
function matlabbatch = dicom_import_job(raw,outdir)
```

Eine Funktion hat nur Zugriff auf die weitergeleiteten oder in der Funktion definierten Variablen. Man kann auch globale Variablen benutzen. Diese sind für alle ausführbaren Funktionen zugänglich und werden mit `global` *Variable* definiert, und müssen in jedem `m`-File definiert werden, welches sie benutzen soll. Um globale Variablen zu löschen benutzt man den Befehl `clear global` oder `clear all`. Das letztere löscht alle im Speicher vorhandenen Variablen.

Structure Array

Die Funktion `dicom_import_job` baut das `matlabbatch` Structure Array auf, was den eigentlichen Kern der SPM-Verarbeitung darstellt. Ein Structure Array ist ein Datentyp von Matlab, welcher hierarchische Informationen zusammen unter einer Dateneinheit speichert. In diesem Array wird die Datenstruktur im SPM-Format aufgebaut. Als Beispiel nehmen wir die erste Zeile (etwas geändert):

```
matlabbatch{1}.spm.util.dicom.data = raw;
```

Der erste Teil ist der Name der Structure Array. Unter Verwendung von geschweiften Klammern deutet man auf den Cell Array. Dies ist, z.B., nützlich

um dessen Inhalte in andere Arrays zu schreiben. Würde man runde Klammern benutzen, so deutet man auf den Inhalt des Cell Arrays. Hier ein Beispiel:

```
EDU>> C = {magic(3) 'Gehirn' 33}
C =
    [3x3 double]    'Gehirn'    [33]
EDU>> a = C{1}
a =
     8     1     6
     3     5     7
     4     9     2
EDU>> b = C(1)
b =
    [3x3 double]
EDU>> whos
  Name        Size              Bytes  Class     Attributes

  C           1x3                 272  cell
  a           3x3                  72  double
  b           1x1                 132  cell
```

Hier ist `C` ein Structure Array, `a` eine Matrix und `b` ein Cell Array. Ein Structure Array kann beliebig viele untergeordnete Felder haben, die Arrays oder Cell Arrays beinhalten. Um mehr über Structure Arrays zu erfahren, kann man im Helpdesk nach 'What is a structure' suchen.

2.3 Field Map Erzeugung

Für das Beispieldatenset wurden ausser den fMRT Daten auch die Inhomogenität des Magnetfeldes durch eine Gradientenechosequenz (GRE) gemessen. Die GRE-Sequenz erzeugt zwei unterschiedliche Betragsbilder (mit zwei unterschiedlichen Echo Zeiten (TE)) und ein Phasenbild. Mit diesen Daten wird eine Field Map erzeugt. Dafür benutzt man die *Field Map Toolbox* die unter dem *Toolbox* Button im SPM Menü zu finden ist. Diese Toolbox kann auch in der Kommandozeile aufgerufen werden.

Verzeichnisstruktur

Das Field Map Batch Skript ist so gedacht, dass es eine bestimmte Verzeichnisstruktur benutzt. Diese sieht folgendermaßen aus:

```
DataPath
└─ subject1
   ├─ raw
   └─ spm
      └─ field_map
         ├─ s...-0002-00001-000001-01.img.....Betrag
         ├─ s...-0002-00001-000001-02.img.....Betrag
         └─ s...-0003-00001-000001-01.img.....Phasenunterschied
```

Field Map Batch Skript

Erstens muss das Hauptverzeichnis der Daten im `m.`-File angegeben sein, gefolgt von den Namen der Subjektverzeichnisse, Field Map Verzeichnis und EPI Verzeichnis. Das letztere wird für diese Berechnung nicht gebraucht, muss aber

trotzdem angegeben werden. Als EPI Verzeichnis gilt jeder Ordner der EPI-Datensätze hat.

```
%% Experiment folder
data_path = '/media/System/Glad/Matlab Kurs';

%% Subjects folders
subjects = {
    'subject1',...    % 1
    'subject2',...    % 2
};

% field map directory
fm_dir = '01_gre_field_34_2x_betrag';

% epi directory of first scan
epi_dir = 'wasser';

%% parameters used to calculate vdm
te1 = 4.92; % short TE
te2 = 7.38; % long TE
epifm = 0;  % no epi-based field map, since GRE
tert = 11.56;   % total epi read out time in ms
kdir = -1;     % blip direction (+1/-1)
mask = 0;   % don't mask brain, done by default with a siemens
    field map
match = 0;  % don't match field map to EPI
write_unw = 0;  % don't write unwarped epi

%% field map calculation #1
for i = 1:numel(subjects)

    % full path of field map
    % needs to be adjusted if data not in 'spm' folder
    FP_fm_dir = fullfile(data_path,subjects{i},'spm',fm_dir);
    FP_epi_dir = fullfile(data_path,subjects{i},'spm',epi_dir);

    VDM = FieldMap_preprocess(FP_fm_dir,FP_epi_dir, [te1, te2,
        epifm, tert, kdir, mask, match, write_unw] );

end
```

Die Field Map Toolbox braucht ein paar vordefinierte Variablen, die zum Teil von der GRE Messung abhängen, wie z.B. die kurze und lange Echozeit in ms. Da in diesem Skript keine EPI-basierte Field Map berechnet wird, werden die anderen Variablen nicht gebraucht, müssen aber trotzdem angegeben werden. Deswegen sind sie entweder auf Null gesetzt, oder haben symbolische Werte. Die Kommentare zu jeder Variable sind selbsterklärend.
Die Field Map Berechnung wird mit dem Befehl `FieldMap_preprocess` durchgeführt. Die `vdm`-Dateien werden im gleichen Ordner erzeugt, wo die GRE Dateien sich befinden.
Nachdem die Field Map berechnet wurde, sollte man diese überprüfen.

2.4 Vorverarbeitung mit SPM

Mit den `nifti`-Dateien und der Field Map kann jetzt die Vorverarbeitung durchgeführt werden. Diese enthält folgende Schritte:

- Slice Timing

- Realign & Unwarp,
- Coregistration,
- Segmentation,
- Normalisation of functional data,
- Normalisation of structural data,
- Smoothing.

Wie oben erwähnt, erstellt man zuerst das Fundament mit Hilfe des Batch Moduses von SPM. Anhand des Beispieldatensets werden wir folgendes eingeben:

- BasicIO → Change Directory;
- SPM → Temporal → Slice Timing
  - Doppelkclick auf Data;
- SPM → Spatial → Realign & Unwarp,
  - Doppelklick auf Data,
  - Für Images wählt man die Dependency (Abhängigkeitsvariable): Slice Timing: Slice Timing Corr. Images (Sess 1)
- SPM → Spatial → Coregister → Coregister: Estimate,
  - Für das Reference Image wählt man die Dependency: Realign & Unwarp: Unwarped Mean Image,
  - Source Image: T1 Bild;
- SPM → Spatial → Segment
  - Data: Dependency: Coregister: Estimate: Coregistered Images;
- SPM → Spatial → Normalise → Normalise: Write für die funktionellen Bilder
  - Doppelklick auf Data,
  - Parameter File Dependency: Segement: Norm Params Subj -> MNI,
  - Images to Write Dependency Realign & Unwarp: Unwarped Images (Sess 1)
  - Hier noch die Voxelgröße $[2, 2, 2]$ und Bounding Box $[-90, -126, -72; 90, 90, 108]$ einstellen;
- SPM → Spatial → Normalise → Normalise: Write für das strukturelle Bild
  - Doppelklick auf Data,
  - Parameter File Dependency: Segement: Norm Params Subj -> MNI,
  - Images to Write Dependency Segment: Bias Corr Images
  - Hier noch die Voxelgröße $[1, 1, 1]$ und Bounding Box $[-90, -126, -72; 90, 90, 108]$ einstellen;

- SPM → Spatial → Smooth
  - Images to Smooth Dependency Normalise: Write: Normalised Images (Subj 1)
  - FWHM: $[6, 6, 6]$.

Zum Schluss Save Batch and Script auswählen und als `preprocessing.m` speichern.
Zwei Dateien werden erstellt: `preprocessing.m` und `preprocessing_job.m`. Interessant ist wieder die letztere. Sie enthält alle für SPM verständliche Anweisungen in der `matlabbatch` Struktur. Jeder Schritt wird als eine untergeordnete Struktur angegeben angefangen mit 1: `matlabbatch{1}, matlabbatch{2}`, etc.

2.5 Algorithmus

`preprocessing.m`

In der Datei `preprocesseing.m` werden, wie bei der DICOM Importierung, die vorher festgelegten Verzeichnisse angegeben. Dazu kommen noch ein paar Variablen die sich am häufigsten mit jedem neuen Experiment ändern, wie z.B. die Voxelgröße, die Phasenkodierungsrichtung oder die Glättungsgröße.
Die Phasenkodierungsrichtung benutzt eine `if-else` Kontrollstruktur um den richtigen Vektor für die gewünschte Richtung auszuwählen.
Alle hier angegebenen Variablen werden dann zur Funktion `preprocessing_job.m` weitergeleitet.

Listing 1: preprocessing.m

```
clear, clc
% voxel size of EPI images
vars.vox_size = [2 2 2];
% smoothing vector
vars.smooth_vec = [6 6 6];
% bounding box
vars.b_box = [-90 -126 -72; 90 90 108];
% assign appropriate vector for the phase encoding direction
vars.wrapping_dir = 'z';

% slice timinig parameters
slice_timing.nslices = 34;      % # of slices
slice_timing.TR = 2;            % TR in secs
slice_timing.sliceOrder = [1:2:slice_timing.nslices 2:2:
    slice_timing.nslices]; % order of slices
slice_timing.refSlice = 17;     % reference slice, should be set to
     middle slice

%% Full path of experiment folder
data_path = '/media/System/Glad/Matlab_Kurs';

%% Field Map Directory name (not full path)
fieldMapDir = '01_gre_field_34_2x_betrag';

%% Subjects and sessions folders (not full path)
subjects = {
            'subject1',...
            'subject2'
            };
%  Sequence folders
```

```
sessions = {
30              'wasser',...
                'jaw'
32              };

34 %% Initialise SPM defaults
spm('Defaults','fMRI');
36 spm_jobman('initcfg');

38 preprocessing_job(data_path,fieldMapDir,subjects,sessions,vars,
    slice_timing);
```

`preprocessing_job.m`

Es werden zwei `for` Schleifen gebildet, eine für die Probanden und eine für die Sitzung. Alle nötigen Variablen werden dann der `matlabbatch` Struktur zugewiesen.
Die funktionellen und strukturellen Bilder und die Field Map werden geladen.

```
        % load functional data for each session
2       funct_data =  spm_select('FPList', fullfile(data_path,
            subjects{i},'spm',sessions{j}), '^f.*\.img$');
        % load structural image and field map. CAREFUL! it might
            change over
4       % sessions
        structural = spm_select('FPList', fullfile(data_path,
            subjects{i},'spm','t1vol'), '^s.*\.img$');
6       fieldmap = spm_select('FPList', fullfile(data_path,subjects
            {i},'spm',fieldMapDir), '^vdm5_scs*.*\.img$'); %$
```

Diese werden dann einer bestimmten Unterstruktur in `matlabbatch` zugeordnet.

```
        matlabbatch{2}.spm.temporal.st.scans = {cellstr(funct_data)
            }';
2   matlabbatch{2}.spm.temporal.st.nslices = slice_timing.nslices;
    matlabbatch{2}.spm.temporal.st.tr = slice_timing.TR;
4   matlabbatch{2}.spm.temporal.st.ta = slice_timing.TR-(
        slice_timing.TR/slice_timing.nslices);
    matlabbatch{2}.spm.temporal.st.so = slice_timing.sliceOrder;
6   matlabbatch{2}.spm.temporal.st.refslice = slice_timing.refSlice
        ;
    matlabbatch{2}.spm.temporal.st.prefix = 'a';
```

Die Dependency erscheint als Variable `cfg_dep`.

```
1       % Realign: Realign and Unwarp
        matlabbatch{3}.spm.spatial.realignunwarp.data.scans =
            cellstr(funct_data);
```

Es ist wichtig die Namen der Abhängigkeitsvariablen richtig anzugeben, ansonsten treten Fehler auf. Diese Einträge kommen jedes Mal vor, wenn eine Abhängigkeit besteht.

```
        matlabbatch{3}.spm.spatial.realignunwarp.data.scans(1).
            sname = 'Slice Timing: Slice Timing Corr. Images (Sess
            1)';
```

Im Segementierungsschritt sind die Vorlagen für die weisse und graue Substanz, sowie für den Liquor angegeben. Diese sind im SPM Ordner zu finden und müssen angepasst werden.

```
matlabbatch{5}.spm.spatial.preproc.opts.tpm = {
            '/home/glad/.spm/spm8/tpm/grey.nii'
            '/home/glad/.spm/spm8/tpm/white.nii'
            '/home/glad/.spm/spm8/tpm/csf.nii'
            };
```

Für den Normalisation Schritt wird die Voxelgröße durch die Variable `vox_size` angegeben:

```
        matlabbatch{6}.spm.spatial.normalise.write.roptions.vox =
            vox_size;
```

Bei der Glättung wird Glättungsgröße durch die Variable `smooth_vec` angegeben:

```
        matlabbatch{8}.spm.spatial.smooth.fwhm = smooth_vec;
```

3 1^{st} Level Analyse

Die vorverarbeiteten Daten werden mit Hilfe einer statistischen Auswertung analysiert. Dafür wird eine Hypothesen über den zeitlichen Verlauf der BOLD Antwort formuliert. Ein Signifikanztest wird verwendet um zu bestimmen mit welcher Wahrscheinlichkeit die Hypothese in den verschiedenen Voxeln der gegebenen Daten erfüllt ist. Die Analyse wird als *1^{st} Level Analyse* oder *Fixed Effects Analyse* interpretiert, d.h. es können ausschließlich Aussagen über die vorliegende Stichprobe ermittelt werden.
Um das Fundament für eine Auswertung der Daten mittels Matlab Skript zu generieren wählt man im SPM Menüfenster den Batch Modus aus und gibt folgendes ein:

- BasicIO → Change Directory;
- SPM → Stats → fMRI model specification,
  - Doppelclick auf Data & Design;
- SPM → Stats → Model estimation,
  - Dependency auswählen für Select SPM.mat: fMRI model specification: SPM.mat File;
- SPM → Stats → Contrast Manager,
  - Dependency auswählen für Select SPM.mat: Model estimation: SPM.mat File;
- SPM → Stats → Results report,
  - Dependency auswählen für Select SPM.mat: Contrast Manager: SPM.mat File,
  - Doppelclick auf Contrasts.

Diese Schritte als Batch & Script speichern, z.B. als `first_level_fundament.m`. In den oben beschriebenen Schritten wird das Modell welches überprüft werden soll spezifiziert. Dieses muss zusammen mit anderen Einträgen noch in die Matlabskripte eingefügt werden, damit es als Batch Skript für mehere Probanden und Sessions durchlaufen kann. Mit Hilfe des Signifikanztests werden Kontraste berechnet, anschließend dargestellt und als `ps`-Datei gespeichert. Die fertigen Matlabskripte werden weiter unten diskutiert.

3.1 `first_level.m`

Listing 2: first_ level.m

```
% Compute first level for multiple subjects and sessions.
%
% Glad Mihai, 02.03.2012

clear, clc
%% Experiment folder
data_path = '/media/System/Glad/Matlab_Kurs';
stat_dir = 'statistical';    % all new folders should be created
    here

% subjects sessions
subjects = {'subject1',...
            'subject2'};
sessions = {
    'wasser',...
    'jaw'};

% use motion parameters as regressors? yes/no
tParams.motion_regress = 'yes';

%%
% timing parameters
tParams.units = 'scans';    % units of design: scans or seconds
tParams.TR = 2;             % repetition time in sec
tParams.mic_res = 34;       % microtime resolution, if # of slices
    > 16 use
                            % # of slices
tParams.mic_onset = 17;     % microtime onset should be set to the
    middle slice

% Swallow
conditions(1).name = 'Swallow TR = 2 s';
conditions(1).onsets = ((5:6:119))'; % since n x 1 vector. Start at
     t = 0!!!!
conditions(1).durations = 0;           % depends on tParams.
    interval, either in scans or seconds
conditions(1).vector = 1;           % contrast vector

% Jaw
conditions(2).name = 'Jaw Movements TR = 2 s';
conditions(2).onsets = ((5:6:119))'; % since n x 1 vector. Start at
     t = 0!!!!
conditions(2).durations = 0;           % depends on tParams.
    interval, either in scans or seconds
conditions(2).vector = 1;           % contrast vector

% print contrast options: unc, FWE
```

```
41 conOpt.whichCon = Inf;      % which contrasts should be printed, Inf
       : all found in SPM.mat
   conOpt.threshType = 'FWE';  % which type of threshold: FWE, none
43 conOpt.thresh = 0.05;       % threshold
   conOpt.extent = 0;          % extent (voxels)
45
   %% initialise spm
47 spm('defaults', 'FMRI');

49 % set up matlabbatch and run job
   first_level_job(data_path,stat_dir,subjects,sessions,conditions,
       tParams,conOpt);
```

Zunächst wird das Wurzelverzeichnis und das Verzeichniss in dem die statistischen Daten gespeichert werden angegeben (siehe Pfadstruktur auf Seite 2). Darauf folgt die Definition der Ordner für die Probanden und der verschiedenen Sessions. Anschließend werden die Timing Parameter definiert, hier werden die Standardparameter verwendet.
Zur Auswertung jeder Session wird ein Sessionname, die Hypothese genannt Onset, die Dauer der Aktivität genannt Duration und der Kontrastvektor festgelegt. Diese Definitionen werden in der Struktur conditions gespeichert. Für jede zusätzliche Session muss eine zusätzliche `conditions` Struktur erstellt werden. Die Ausgabe der Kontraste wird mit Hilfe der `conOpt`-Struktur gesteuern. Hier wird angeben welche Kontraste und mit welcher Schwelle diese dargestellt werden sollen.Ebenso wird hier angegeben ob eine einfache Schwelle oder eine fehler korrigierte Schwelle verwendet werden soll. Abschließend wird SPM initialisiert und die Funktion `first_level_job.m` aufgerufen.

3.2 `first_level_job.m`

```
   function first_level_job(data_path,stat_dir,subjects,sessions,
       conditions,tParams,conOpt)
2  %% loop over subjects and session
   for i = 1:numel(subjects)
4      for j = 1:numel(sessions)
```

Ganz am Anfang wird die Funktion definiert (Zeile 1). Für den Aufbau der `matlabbatch` Struktur werden zwei Schleifen durchgeführt, für die Probanden und für die Sessions (Zeilen 3 und 4).

```
   % create parent directory (statistical) for the new stat folders to
       be created
2  % first check if it exists. if so then skip mkdir
   type = exist(fullfile(data_path,subjects{i},'spm',stat_dir),'dir');
4  if type ~= 7
       mkdir(fullfile(data_path,subjects{i},'spm'),stat_dir);
6  end
   parent_dir = fullfile(data_path,subjects{i},'spm',stat_dir);
8
   % directory where SPM.mat will be written
10 session_dir = fullfile(parent_dir,strcat(sessions{j}));
   % check if session dir exists, if not create it
12 type = exist(session_dir,'dir');
   if type ~= 7
14     mkdir(parent_dir,strcat(sessions{j}));
   end
```

Es wird überprüft ob es den Ordner `statistical` gibt (Zeile 3) und wenn nicht (d.h., wenn Zeile 4 Wahr ist, also wenn `type` nicht gleich 7), dann soll dieser erstellt werden (Zeile 5). Das Gleiche passiert für die Session Ordner im `statistical` Ordner—sind diese nicht vorhanden, so werden sie hier kreiert.

```
% load functional data
files = spm_select('FPList',fullfile(data_path,subjects{i},'spm',
    sessions{j}), '^sw.*\.img$');
matlabbatch{2}.spm.stats.fmri_spec.sess.scans = cellstr(files);
```

Die funktionellen Dateien (geglätteten Bilder `sw*.img`) werden mit Hilfe von `spm_select` ausgewählt und and die Struktur `matlabbatch` übergeben.

```
% assign condition name, onset, duration
matlabbatch{2}.spm.stats.fmri_spec.sess.cond.name = conditions(j).
    name;
matlabbatch{2}.spm.stats.fmri_spec.sess.cond.onset = conditions(j).
    onsets;
matlabbatch{2}.spm.stats.fmri_spec.sess.cond.duration = conditions(
    j).durations;
```

Die in der ersten Datei als Conditions vordefinierte Hypothesen werden hier angegeben. Der Index `j` gibt die Sessions an.

```
% load multiple regressors
reg_file = spm_select('FPList', fullfile(data_path,subjects{i},'spm
    ',sessions{j}), '^rp.*\.txt$');
matlabbatch{2}.spm.stats.fmri_spec.sess.multi_reg = {reg_file};
```

An diesem Punkt werden die multiplen Regressoren zur Bewegungskorrektur aus der Text Datei geladen und an die `matlabbatch` Struktur weitergegeben.

```
matlabbatch{4}.spm.stats.con.consess{1}.tcon.name = conditions(j).
    name;
matlabbatch{4}.spm.stats.con.consess{1}.tcon.convec = conditions(1)
    .vector;
```

Für die Berechnung der Kontraste müssen Name und Vektor angegeben werden.

```
% print all contrasts found in SPM.mat file by setting it to Inf
matlabbatch{5}.spm.stats.results.conspec.contrasts = conOpt.which;
matlabbatch{5}.spm.stats.results.conspec.threshdesc = conOpt.
    threshType;
matlabbatch{5}.spm.stats.results.conspec.thresh = conOpt.thresh;
matlabbatch{5}.spm.stats.results.conspec.extent = conOpt.extent;
```

Hier werden die Optionen für die Kontrastenausgabe angegeben: welche Kontraste berechnet werden sollen, die Art der Schwelle, die Schwelle selbst und wieviele Voxel für ein Cluster vorhanden sein müssen.

```
batch_file_name = strcat('firstLev_batch_',sessions{j},'.mat');
save(fullfile(data_path,subjects{i},'spm','statistical',
    batch_file_name),'matlabbatch');
spm_jobman('serial', matlabbatch , '');
clear matlabbatch
```

Der Name der `mat`-Datei wird erzeugt, und die `matlabbatch` Struktur wird in dieser Datei gespeichert. Die Verarbeitung wird durch den `spm_jobman` für jede einzelne Session ausgeführt. Danach wird die `matlabbatch`-Struktur gelöscht um den Arbeitsspeicher frei zu geben, und um sicher zu stellen, dass keine alten Einträge vorkommen.