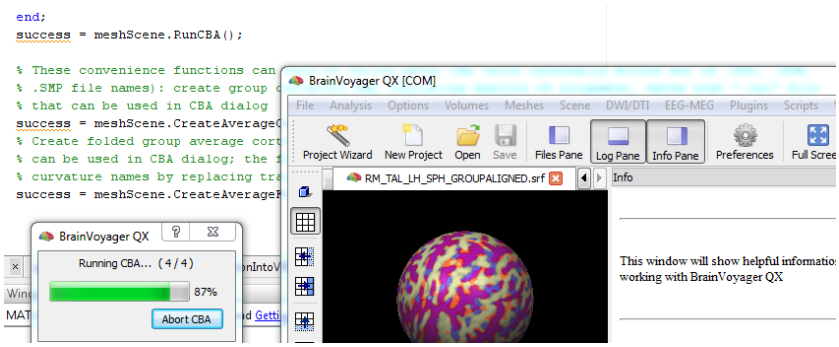
Scripting BrainVoyager 23.0



from Matlab

Rainer Goebel

Contents

Chapter 1

Introduction

The Microsoft component object model (COM) technology on Windows is in BrainVoyager QX 1.9-2.0 and 2.2-3.4 (BrainVoyager 21.4) implemented in its scripting module. This makes it possible to communicate between other COM-enabled programs and BrainVoyager [2] via the scripting commands (1.1). Please note that it is always possible to check the BrainVoyager Scripting Reference in case details are missing, since this uses the same API.

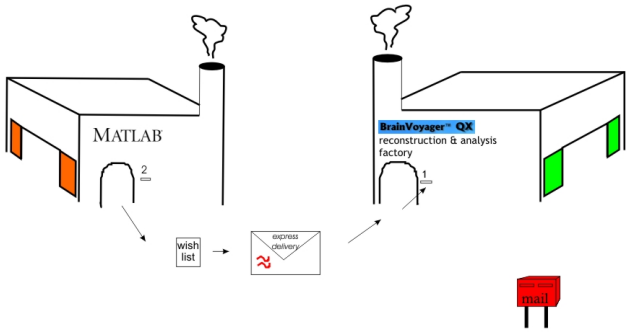


Figure 1.1: Remote scripting in BrainVoyager from matlab

These programs are other COM-enabled programs, like Matlab (The MathWorks, Inc.) or Microsoft Excel. Via the OLE Viewer it is possible to check which programs are COM-enabled. First, activate OLE View (see figure 1.2).

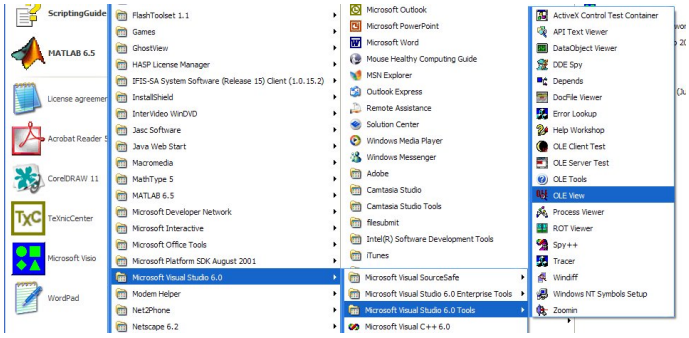


Figure 1.2: Activating OLE View

Then, look in the category 'Automation objects' (see figure 1.3). (Thanks to Dirk Heslenfeld for this

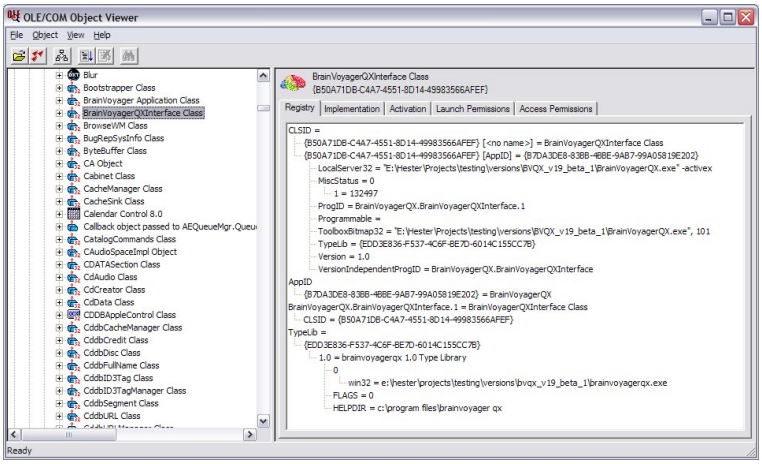


Figure 1.3: BrainVoyager in the OLE Viewer

contribution).

When BrainVoyager is not installed via the installer, use the command `-regserver` to activate the COM interface in BrainVoyager (figure 1.4):

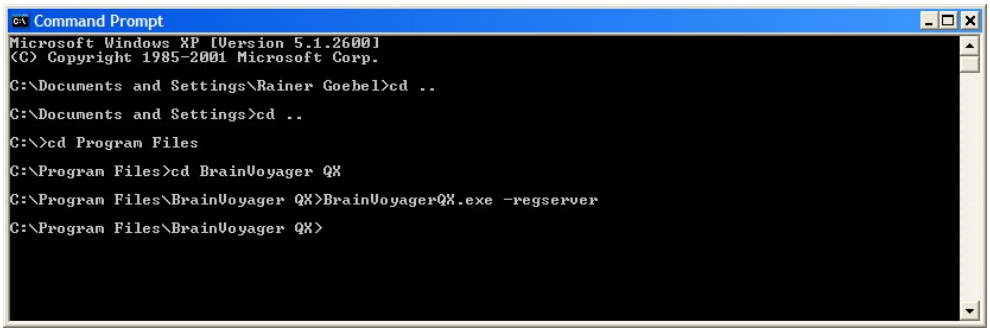


Figure 1.4: Activating the COM interface in BrainVoyager

In this case, ensure that there are no spaces in the filename and pathname, or that the name is provided between double quotes (thanks to Jens Schwarzbach).

For Windows 10 and higher, use the PowerShell and add a dot slash (Figure 1.5). Please note that on Windows 10 some security settings might prevent for COM to run; in that case, use administrator rights. Also reinstallation of BrainVoyager can help, if there are issues with the Windows Registry. (Check the Registry by searching for the program 'regedit' in the Windows Start menu.)

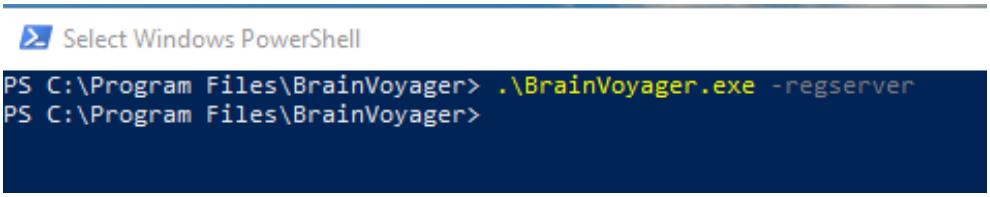


Figure 1.5: Activating the COM interface in BrainVoyager on Windows 10

1.1 History

1.1.1 Changes in scripting

Note: the information below about changes in scripting is applicable to scripting in BrainVoyager in general.

1.1.2 BrainVoyager 23

The following functions have been added: IsVMR(), IsFMR(), IsDMR(), SaveVolumeMaps(), GetNameOfVolumeMap(), DeselectVOI(), GetBetaValueOfROIGLM(), GetBetaNameOfROIGLM(), GetVTCsOfMDM(), FileExists(), CopyFile(), MoveFile(), GetFileFromFilePath(), GetPathFromFilePath(), ShowVolumeMapsDlg(), HideVolumeMapsDlg(), UpdateView(), SaveFMRAndSTCFromMem(), SaveScreenshotOfSurfaceWindow(), CreateVMRFromDICOM(), CreateFMRFromDICOM(), CreateDMRFromDICOM(), CreateVMRFromDICOMAsNIFTIBIDS(), CreateFMRFromDICOMAsNIFTIBIDS(), CreateDMRFromDICOMAsNIFTIBIDS(), SaveFMRAndSTCFromMem(), HideLogPane(). Also SetGUIScriptWindowTitle(), FinishPlugin()FinishPlugin(), CloseDialog(), HideScriptsWindow(), ShowScriptsWindow(), CloseAll() and Exit().New properties are: FileName, PathFileName, Path, NrOfVolumes, NrOfSkippedVolumes, NrOfSlices, ProtocolParametric, VOIFileName, Visible, WindowX, WindowY, WindowWidth, WindowHeight, Documents, VersionPatch, Is64Bits, DarkMode, BVTheme, NrOfMTCVolumes, NrOfMTCTimePoints, NrOfSurfaceMaps.

And the following function names have been changed: MapSphereMeshFromStandardSphere() → MapSphereFromStandardSphere(), MergeMeshesInScene() → MergeMeshes(), CreateStandardSphereMesh() → CreateSphereMesh(), ShowLogTab() → ShowLogPane().

Property names that changed, are: VMRVoxelResolutionX/Y/Z → VMRVoxelSizeX/Y/Z, PixelSizeOfSliceDimX/Y → PixelSizeX, VoxelResolutionVerified → VoxelSizeVerified, *StimulationProtocol properties → Protocol properties, DimX/Y/Z.

BrainVoyager 20-22.2

Several surface functions have been introduced (such as functions for MTC statistics) and some have been renamed from 'mesh' to 'geometry' (new function names are SmoothGeometrySimple(), SmoothGeometry(), InflateGeometry(), InflateGeometryToSphere(), CorrectInflatedSphereDistortions() and SimplifyGeometry()). The function CorrectSliceTiming() and CorrectSliceTimingWithSliceOrder() received a third argument.

BrainVoyager 21.4

It is now possible to load and save NIfTI files via Python (and JavaScript) scripts. There is no new command - the respective functionality is invoked in case that the file name parameter of the `OpenDocument()` command of the global `BrainVoyager` object contains the file extension ".nii" or ".nii.gz". The NIfTI file extension also triggers exporting VMR anatsomical and FMR functional documents as NIfTI files when using the `SaveAs()` command of a `Document` object. Note that the earlier deprecated opening and creation document command names using `Project` (e.g. `CreateProjectVMR()`) have now been removed, i.e. scripts need to use the corresponding `Document` commands (e.g. `CreateDocumentVMR()`).

BrainVoyager 21.0

Replaced `GetMeshScene()` by CreateMeshScene().

BrainVoyager 20.6

Some commands have been added to BrainVoyager related to surface maps: SmoothMap(), SmoothMapLags(), CreateSurfaceMapFromVolumeMap() and CreateSurfaceMapFromVolumeMapDepth().
Some documentation has been added for map functions:
LoadVolumeMaps(), ShowVolumeMap(), ShowSurfaceMap(), LoadSurfaceMaps().

BrainVoyager 20.4

Invoking `actxserver()` has changed slightly. Also, a function to obtain the same resolution in all dimensions has been added: TransformToIsoVoxel(), as well as some major transformation functions: CoregisterFMRToVMR(), CoregisterFMRToVMRUsingBBR(), NormalizeToMNISpace(), CreateVTCInMNISpace(), NormalizeToMNISpace().

BrainVoyager QX 3.0/BrainVoyager 20

BrainVoyager can now also be used with Python scripting. Also, a Data Analysis Manager is an alternative to scripting for document/project creation, preprocessing and statistics in volume space. The function `GetMeshScene()` has replaced the property `CurrentMeshScene` (see section 7).

BrainVoyager QX 2.8.0

The available scripting commands have been substantially extended in this release. The most substantial change is the addition of a mesh object that allows to script mesh morphing, mesh simplification, rigid and non-rigid CBA, MTC preprocessing and statistical analyses. A number of new example scripts are provided (file names start with the "Mesh" substring) demonstrating how to use the new scripting possibilities. Also, scripting of inhomogeneity correction and transformation to AC-PC and Talairach space are now available for anatomical files (*.vmr). For further details, consult the updated Scripting Reference Manual and the AppleScript Guide.

BrainVoyager QX 2.8.2 Two new functions and a property, concerning correction of slice timing and mean intensity adjustment, have been introduced.

BrainVoyager QX 2.8.4 Quite a number of VOI-GLM functions have been added (see section 6.4).

BrainVoyager QX 2.4.1

It is now possible to perform *temporal high-pass filtering (drift removal) using the GLM approach* using Fourier or discrete cosine transform (DCT) basis functions. In previous versions, only the FFT-based high-pass filtering was available. The new commands are "TemporalHighPassFilterGLMFourier()" and "TemporalHighPassFilterGLMDCT()" with one parameter that specifies the number of cycles (pairs of two basis functions) used to build an appropriate design matrix. The installed script "HighPassFilterUsingGLM.js" shows how to use the new commands. Other new scripting commands allow to interrogate information about the running BrainVoyager version including the build number and whether the program is running in 32 or 64 bit mode. The installed script "VersionScript.js" shows how these commands can be used.
Scripts are installed in a standard location within the user's "Documents" folder ("BVQXExtensions/Scripts"). For some scenarios, it would be beneficial if scripts could be accessed from a custom folder as default, i.e. when written scripts are made available to members of a research group in a shared network folder. For such scenarios it is now possible to change the default scripts folder in the "Scripts" tab of the "Global Preferences" dialog.
There are two new possibilities for manipulating VTCs. The first is that the manual specification of bounding boxes is enabled. This works for any target reference space. Use the property 'UseBoundingBoxForVTCCreation' in combination with TargetVTCBoundingBoxXStart (or Y, or Z) and TargetVTCBoundingBoxXEnd (or Y, or Z).
It is now possible to save the VTC after a protocol has been linked, using the command SaveVTC().

BrainVoyager QX 2.3

BrainVoyager QX can now be used in combination with AppleScript on Mac OS X. For more information, see the 'BVQXAppleScripting.pdf' guide.

For Qt Script there are the following additions. When creating VMR projects, the internally created V16 data set is now stored to disk. When saving the VMR data with a new name ("save as" command used usually after VMR creation), both files will be renamed as long as they have the default "untitled.vmr/.v16" file name.
The new command "CorrectSliceTimingWithSliceOrder" allows to run slice scan correction with a custom slice order (see "Preprocessing.js" script). The "getCurrentDirectory()" function is now a property, i.e. you can use "BrainVoyagerQX.CurrentDirectory" to read and set its value.
There are also new properties pointing to common locations:
The "PathToData" property points as default to the "BVQXData" folder in the user's "(My)Documents" folder and the "PathToSampleData" points as default to the "BVQXSampleData" folder.
Please note that the `FileNameOfPreprocessedFMR` now provides the filename and the path, not just the filename.

BrainVoyager QX 2.2

There are now reading and writing (File I/O) possibilities. Also, the COM (component object model) functionality has been implemented, which means that communication between COM-enabled applications on Windows is possible, for example scripting BrainVoyager from Matlab.
New BrainVoyager script functions are available for preprocessing VTC files, creation of MTC from VTC and create function handles.

BrainVoyager QX 2.1

The language is now fully ECMA-script compliant, which means it is basically JavaScript. Most of the language features are the same; the graphical user interface (GUI) widgets can be made using external user interface files (*.ui).
The parameter `dataType` has been added for creating VTC and VDW files. Two properties for changing the confound in SDM files have been added.
COM not available.

BrainVoyager QX 2.0

No changes.

BrainVoyager QX 1.10

In BrainVoyager QX 1.10.4, it is also possible to create VDW files via scripting.
In BrainVoyager QX 1.10.3, the types of interpolation that can be selected have been extended. For more information, please consult the Interpolation in motion correction page.

BrainVoyager QX 1.9

This version is updated for BrainVoyager QX 1.9. Two important new changes in BrainVoyager QX 1.9 are the DTI analysis functionality and scripting via the component object model (COM)(this works on Windows platforms).

Also, there are 5 new scripting functions:
RenameDicomFilesInDirectory(), BrowseFile(), BrowseDirectory(), CreateProjectDMR() and CreateProject-MosaicDMR().

For details on creating diffusion weighted projects (DMR), see the topic Creating DMR projects.

For the function to rename DICOM files and the use of BrowseDirectory(), please see the new rename DICOM files page.

The new BrainVoyager QX Getting Scripted Guide can be consulted for a step-by-step approach into scripting.

In 1.9.10, the number of interpolation options has increased for slice scan time correction* and VTC creation. For details, see the BrainVoyager QX 1.9.10 Release Notes.
Changes in the programming language itself concern, for example, the `undefined` which is in BrainVoyager QX 1.9 an object (so no double quotes are needed). Also, the arguments for the getOpenFileName(s) functions have changed. The language specification for Qt Script 1.2.2 by Trolltech can be found also in this guide.

1.1.3 Changes in documentation

15-09-2023: Added new and changed functions and properties in BrainVoyager 23.
18-11-2021: Added third argument to CorrectSliceTiming() and CorrectSliceTimingWithSliceOrder(). With thanks to Dr. Bick.
Several surface functions have been added and some renamed from 'mesh' to 'geometry'; the renamed functions are: SmoothGeometry(), SmoothGeometrySimple(), InflateGeometryToSphere(), InflateGeometryToSphereExt(), InflateGeometry() and SimplifyGeometry(). Also, added the mesh functions SaveSingleStudyGLMDesignMatrix(), LoadMultiStudyGLMDefinitionFile(), AddStudyAndDesignMatrixAndCortexMapping(), SaveMultiStudyGLMDefinitionFile(), ComputeMultiStudyGLM(), ComputeRFXGLM(), LoadGLM(), SaveSurfaceMaps(), GetNameOfSurfaceMap(), Update3DViewer() and mesh properties `NrOfMTCVolumes`, `NrOfMTCTimePoints`, `NrOfSurfaceMaps`.
14-09-2021: Scripts updated to Getting Started Guide 4.0. Removed Presentation-to-protocol script and data-driven analysis section.
28-04-2021: Replaced CreateProject*() functions with CreateDocument*()
27-04-2021: Added function information, removed some references to QX, updated some code
15/16-08-2017: Added small comment on inhomogeneity correction.
22-06-2017: Added some commands concerning volume and surface maps.
06-03-2017: Added `SaveAs()`, corrected `CreateVTCInMNISpace()`. Corrected some hyperlinks. Added registration diagram.
17-01-2017: Added new scripting functions of BrainVoyager 20.4.
26-09-2015: Adaptation to new naming of BrainVoyager.
26-11-2014: Added new scripting functions of BrainVoyager QX 2.8.4.
04-07-2014: Added new scripting functions of BrainVoyager QX 2.8.2.
15-06-2013: Added new scripting functions, scripts of BrainVoyager QX 2.8.
29-08-2012: Added information about new functions and properties

1.2 Getting started

1.2.1 Starting BrainVoyager from Matlab

BrainVoyager can be started from Matlab via the simple command

`bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')` (BrainVoyager 20.4 and higher)
`bvqx = actxserver('BrainVoyagerQX.BrainVoyagerQXScriptAccess.1')` (QX 2.2 and higher)
`bvqx = actxserver('BrainVoyagerQX.BrainVoyagerQXInterface.1')` (QX 1.9 and 2.0)

BrainVoyager will fullfil the role of COM-server, where the COM-client Matlab will request its methods via the COM-interface (see figure 1.6).

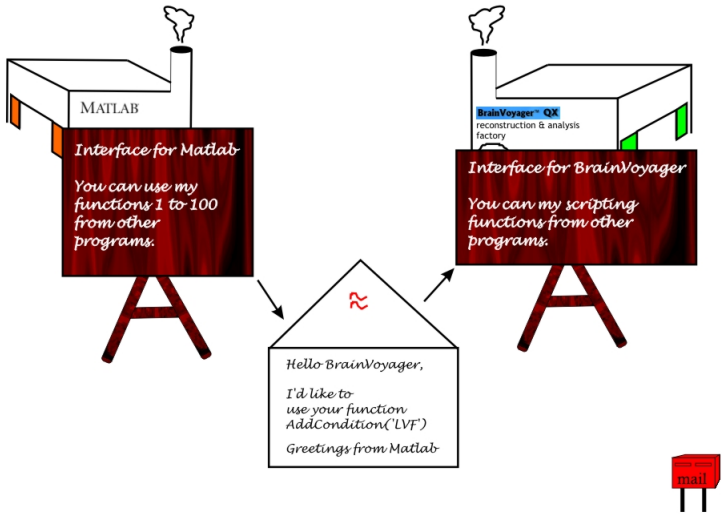


Figure 1.6: Matlab requesting the COM-interface to the BrainVoyager scripting methods

The items that are accessible via COM are shown as yellow cubes (objects) in the Matlab Workspace (see figure 1.7).

We can literally send messages from Matlab to BrainVoyager via the `PrintToLog` function of BrainVoyager. The figure 1.8 below shows how easy this is. First, the BrainVoyager COM server object is invoked via
`bvqx = actxserver('BrainVoyagerQX.BrainVoyagerQXInterface.1');` (QX 1.9 and 2.0) or
`bvqx = actxserver('BrainVoyagerQX.BrainVoyagerQXScriptAccess.1');` (QX 2.2 and higher)
`bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')` (BrainVoyager 20.4 and higher)
Then, the log tab is sent to the front via `bv.ShowLogTab;`. Now the message can be sent via the argument for the function `PrintToLog()`:
`bv.PrintToLog('Message from Matlab...')`.

The methods and properties of an object can be found via the Property Inspector in Matlab (see figures 1.9 and 1.10). To open the Property Inspector, right-click on the `bvqx` object.

They can also be obtained in a Workspace object list via the method `bvqxfuncs = get(bvqx)`.

If necessary, the BrainVoyager window can be resized so that Matlab and BrainVoyager fit on one screen via
`bv.ResizeWindow(600,700);`.

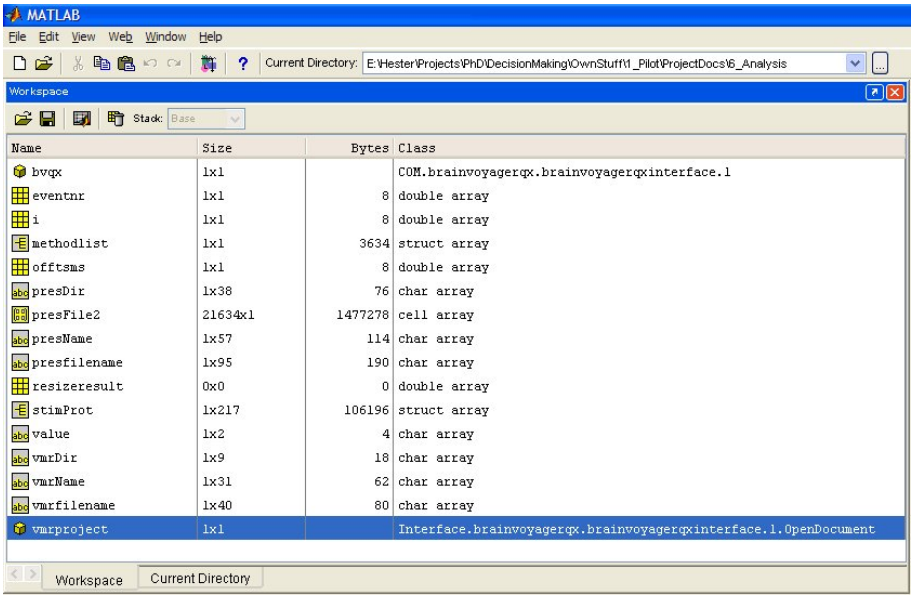


Figure 1.7: COM and other objects in the Matlab Workspace

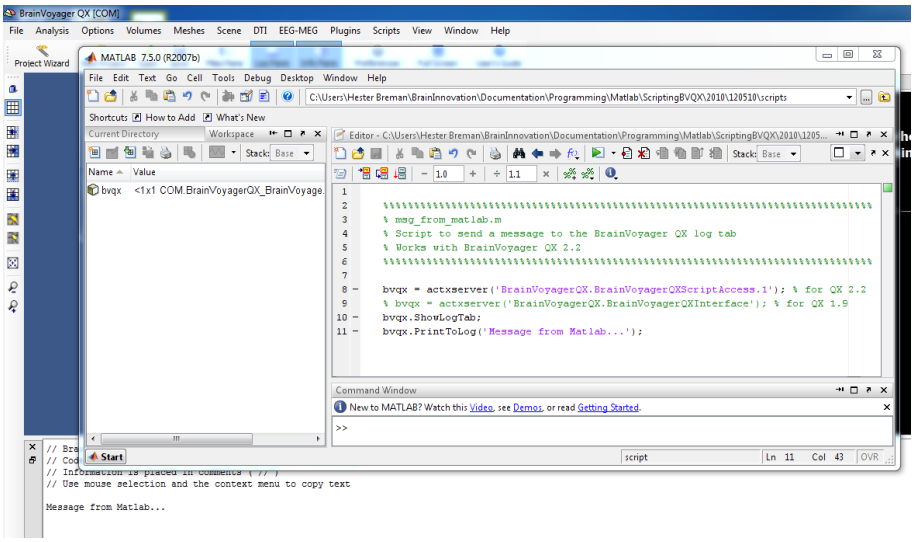


Figure 1.8: Sending a message from Matlab to the BrainVoyager log tab

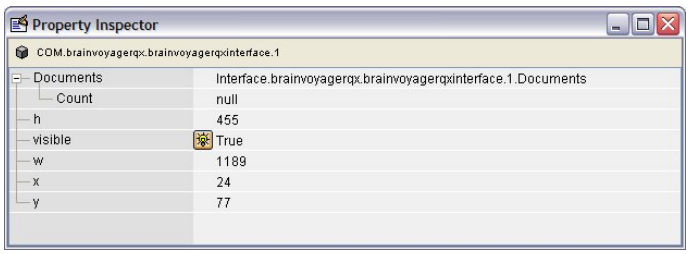


Figure 1.9: Methods of the BrainVoyager object in Matlab in the Property Inspector

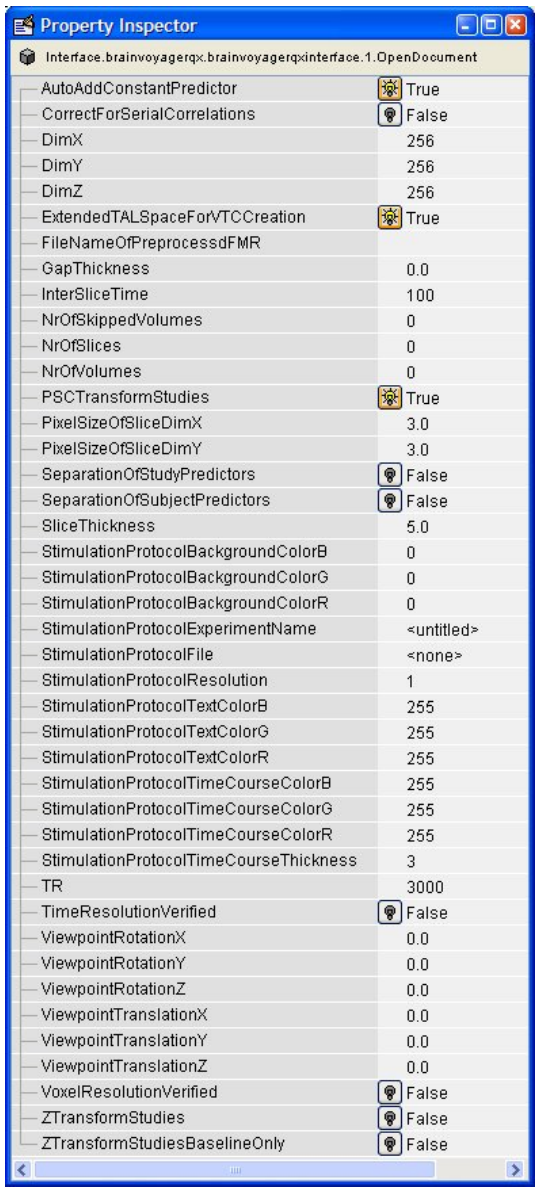


Figure 1.10: Methods of the VMR document in Matlab in the Property Inspector

1.2.2 Syntax differences

The same principles as in Qt Script should be used when scripting BrainVoyager from Matlab. When using a scripting function, first the name of the object is mentioned, then a '.' and then the function of that object: `nameObject.functionObject();`. So when the BrainVoyager server just is started, only the methods of the BrainVoyager application object (here 'bvqx') can be used, for example `bv.OpenDocument('C:/Data/CG_QX_DCM.vmr');`.
Once a BrainVoyager FMR, VMR, DMR or AMR document is opened, the functions of documents can be used as well, for example `vmrdocument.AddCondition('LVF');`. However, there is one difference between scripting via the BrainVoyager Scripting Editor and BrainVoyager scripting via Matlab, which is that the parentheses `()` are not used when there are no arguments for a function, for example `vmrdocument.ClearStimulationProtocol;`.

In BrainVoyager, several scripting functions return a boolean value `true` or `false` to indicate whether the operation succeeded, for example when creating documents or preprocessing FMR files. In Matlab, these values are simply represented by 0 for `false` and 1 for `true`.

A script does not need to be executed at once in Matlab, also fragments can be used. In this way one can 'walk through' a script to see the effect of each of the steps. Code fragments can be executed by selecting the code, right-clicking with the mouse and choosing 'Evaluate selection' (see figure 1.11).

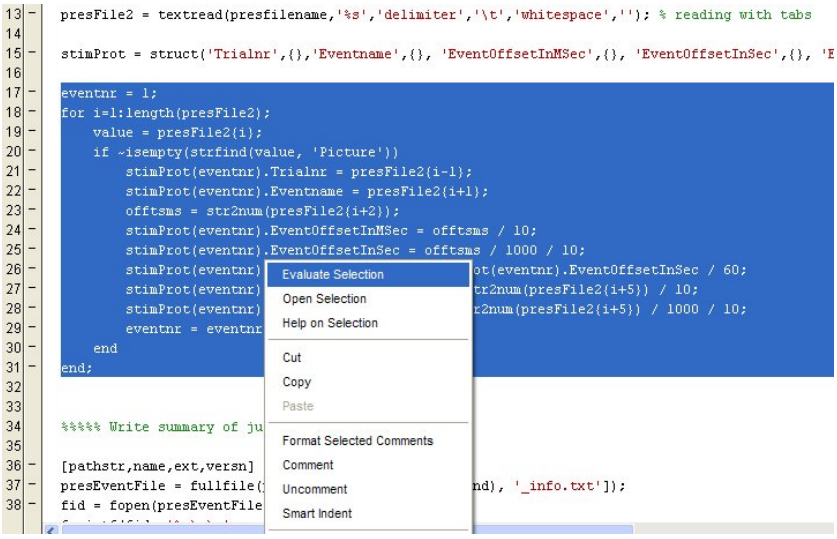


Figure 1.11: Executing a selection of code in Matlab choosing 'Evaluate selection' after right-clicking

1.2.3 List of functions

Table 1.1: Relevant BrainVoyager functions

Description of function	*Function in BrainVoyager API*	*Since version*
Update surface window	UpdateSurfaceWindow;	
Make snapshot of surface window	SaveSnapshotOfSurfaceWindow;	
Show BrainVoyager log tab	ShowLogTab	1.4
Show BrainVoyager log tab	ShowLogPane	23.0
Hide BrainVoyager log tab	HideLogPane	23.0
Send text to log tab	PrintToLog('message')	1.4
Move BrainVoyager window	MoveWindow(new x, new y)	1.0
Open document (FMR, VMR, DMR, AMR)	OpenDocument('name')	1.0
Close document (FMR, VMR, DMR, AMR)	Close;	1.0
Resize the BrainVoyager window	ResizeWindow(new width, new height);	1.0
Rename DICOM files in directory	RenameDICOMFilesInDirectory('dirname')	1.9
Get file name	BrowseFile()	1.9
Get directory name	BrowseDirectory()	1.9
Check whether file exists	FileExists()	23.0
Copy a file	CopyFile()	23.0
Move file to other directory	MoveFile()	23.0
Extract filename from full name	GetFileFromFilePath()	23.0
Extract path name from full name	GetPathFromFilePath()	23.0
Show BV script editor	ShowScriptsWindow()	23.0
Hide BV script editor	HideScriptsWindow()	23.0
Close all documents	CloseAll()	23.0
Close BrainVoyager application	Exit()	23.0
(n/a) Set GUI-QML window title	SetGUIScriptWindowTitle()	23.0
(n/a) Close GUI-QML plugin	FinishPlugin()	23.0
(n/a) Close GUI-QML dialog	CloseDialog()	23.0

Table 1.2: BrainVoyager object properties

Description	*Property*	*Since*
Get or set current directory	CurrentDirectory	QX 2.3 (change from function to property)
Points to "BVQXData" folder in the user's "(My)Documents" folder	PathToData	QX 2.3
Points to "BVQXSampleData" folder	PathToSampleData	QX 2.3
Get version number of BrainVoyager	VersionMajor	QX 2.3
Get subversion number of BrainVoyager	VersionMinor	QX 2.3
Get build number of BrainVoyager	BuildNumber	QX 2.3
Get patch number of BrainVoyager	VersionPatch	BV 23.0
Check whether current BrainVoyager version is 64-bits (otherwise 32-bits)	Is64Bits	QX 2.3
Get array of currently opened documents	Documents (property)	2011
This readonly string property provides the path of any loaded document.	Path	QX 3.4 (BV20.4)
Show or hide the BrainVoyager main window	Visible	BV 23.0
Position on screen of left corner of BrainVoyager window on x-axis	x (property)	QX 1.0
Position on screen of left corner of BrainVoyager window on x-axis	WindowX (property)	BV 23.0
Position on screen of upper corner of BrainVoyager window on y-axis	y (property)	QX 1.0
Position on screen of upper corner of BrainVoyager window on y-axis	WindowY (property)	BV 23.0
Get/set width of BrainVoyager window	WindowWidth	BV 23.0
Get/set height of BrainVoyager window	WindowHeight	BV 23.0
Get all open documents	Documents	BV 23.0
Check whether system is 64-bits	Is64Bits	BV 23.0
Get/set BrainVoyager dark mode	DarkMode	BV 23.0
Get/set the BrainVoyager theme	BVTheme	BV 23.0

More information on the `Path` property: The returned string contains the full path to the directory where the document is stored on disk. The path component delimiter is the "/" character, which is also the trailing element of the returned path. The returned value can also be derived from the PathFileName property by removing the last (i.e. file name) component Also, since 2011, there is a new object, the `Documents` property. This is an array of BrainVoyager documents. By requesting the array and querying each document at a time, one can infer the properties of all documents that are currently opened in BrainVoyager (see table below).

Table 1.3: Documents object properties

Description of property	*Function in BrainVoyager API*	*Since version*
Get number of currently open documents	Count	2011
Get document i of array	Item(i)	2011

Chapter 2

Creating documents

In this chapter is shown via examples how to create BrainVoyager functional (FMR), anatomical 3D (VMR), anatomical 2D (AMR) and diffusion weighted image (DMR) documents.

2.1 Renaming and anonymizing DICOM files

But first it might be necessary to rename the files, in case they are DICOM. This can be performed with the command
`RenameDICOMFilesInDirectory( 'dirname' )`. Example use is first to start BrainVoyager via:
`bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1');`
then to rename the files in a certain directory:
`bv.RenameDICOMFilesInDirectory('C:\Data\Experiment\');` For anonymizing DICOM files, the function `AnonymizeDicomFilesInDirectory()` can be used from BrainVoyager 20.4. This requires two arguments, the path (string): String specifying the path (folder) containing the files to process. If a relative path is provided, make sure that the current directory is set appropriately, and newPatientsName (string): String specifying the new patient name replacing the original name found in the DICOM file (0010,0010). The name is only replaced if the existing size of the text entry is large enough to hold the new patient's name, i.e. the length field of the tag is not adjusted at present and in that case a message is written to the Log pane; it is thus advised to use short names, such as "P24".
This function renames DICOM files to a standard format that is easier to process by functions reading raw data files operating in the same way as the "RenameDicomFilesInDirectory()" command (see above). The patient's name is replaced by the provided new patient's name to anonymize the file resulting file names. The provided new name is also used to replace the value of the patient's name entry in the DICOM file (0010,0010) providing basic anonymization (other patient data such as date of birth is not changed). Note that all DICOM files found in the specified folder receive the same new patient's name, i.e. all files in the directory should be from the same participant.

2.2 Creating a functional document (*.fmr)

For a mosaic functional data document (*.fmr), the following parameters need to be provided:

1. Filetype: Define the file type. `fileType = 'DICOM';` or use one of `'SIEMENS', 'GE_I', 'GE_MR', 'PHILIPS_REC', 'ANALYZE'.`

2. Name of first file: Create a variable with the first file name including the file path. `firstFile = 'C:/Data/0001.dcm';`

3. Number of volumes: Specify the number of volumes `nrOfVols = 291;`

4. Number of volumes to skip: Specify the number of volumes that should be skipped
   `var skipVols = 0;`
5. Create pseudo AMR: Indicate whether a pseudo AMR document should be created from the first functional volume via true or false.
   `createAMR = true;`
6. Number of slices: Specify the number of slices.
   `nrSlices = 64;`
7. STC prefix: Provide an prefix name that will be used for the *.stc slice(s).
   `stcprefix = 'run1-';`
8. Swap: In case the raw data are Big Endian, set the 'swapBytes' parameter to true.
   `byteswap = false;`
9. Width of mosaic image: Indicate the x-resolution of the mosaic image, this is the size of the concatenated slices (for example 25 slices will be in a square grid of 5 x 5. When the x-resolution of a single file is 64, the x-resolution for the mosaic image is 5 x 64 = 320. The size is also visible in the 'Rows' and 'Columns' of the Info tab when creating a document via the user interface.
10. Height of mosaic image: Indicate the y-resolution of the mosaic image.
11. Number of bytes per pixel: Insert the number of bytes per pixel of the data. For functional data, the number of bytes is usually 2 (short integer, 16 bits). `bytesperpixel = 2;`
12. Directory to save the data: Provide the name of the path where the files should be saved.
    `savingDir = "C:/Data/";`
13. Number of volumes in mosaic image: Enter the number of volumes in a mosaic image.
14. Image width: The x-resolution parameter should describe the width of the image. `sizeX = 100;`
15. Image height: The y-resolution parameter should describe the height of the image. `sizeY = 100;`

Please note that the order of the parameters is different when the data are non-mosaic (see the examples below).

Non-mosaic

```
  fmr = BrainVoyager.CreateDocumentFMR(fileType, fmrname, nrOfVols,...
skipVols, createAMR, nrSlices,...
stcprefix,byteswap, sizeX, sizeY, bytesperpixel);
```

Mosaic

```
  fmr = BrainVoyager.CreateDocumentMosaicFMR(fileType, fmrname,...
 nrOfVols, skipVols, createAMR, nrSlices,...
stcprefix, byteswap, mosaicSizeX, mosaicSizeY, bytesperpixel,...
targetfolder, nrVolsInImg, sizeX, sizeY);,...
```

For the BrainVoyager sample data used for the Getting Started Guide 2.5, the command would be:

```
   fmr = bv.CreateDocumentMosaicFMR( 'DICOM',...
'C:/Data/JudEck_20181128_BI_Exercises_sess4-0002-0001-00001.dcm',...
 291, 0, true, 64, 'FacesHousesExperiment-', false, 800, 800, 2,...
'C:/Data/Experiment', 1, 100, 100 );
```

Save the document via `fmr.SaveAs('sub-01_ses-04_task- blocked_run-1_bold.fmr');`.

CreateFMRFromDICOM()

CreateFMRFromDICOM()
Function: CreateFMRFromDICOM(FirstFileName, prefixSTCs, savingDir, strProtocolFile="").
Description: FMR documents consist of a set of functional data in the original "slice space". This convenient scripting function is only for modern DICOM files that contain 1 volume-per-file: multi-frame enhanced Dicoms and Siemens Mosaic Dicoms. For other DICOM files, please use `CreateDocument(Mosaic)FMR()`.
Member of class: BrainVoyager.
Parameter 1: FirstFileName: name of the first "raw" data file.
Parameter 2: prefixSTCs: name for the *.stc file.
Parameter 3: savingDir - directory for saving the FMR document.
Parameter 4 (optional): strProtocolFile: name of stimulation protocol file (*.prt)
Returns: Document

CreateFMRFromDICOMAsNIFTIBIDS()

CreateFMRFromDICOMAsNIFTIBIDS()
Function: CreateFMRFromDICOMAsNIFTIBIDS(strFileOfSeries, subj_id, ses_id, run_id, strProjectFolder, strProtocolFile="").
Description: FMR documents consist of a set of functional data in the original "slice space". This function creates NIFTI files in BIDS folder structure below the provided project folder (if without path, project will be created/used as sub-folder of "Documents/BrainVoyager/Projects". Since BrainVoyager 23.0. This convenient scripting functions is only for modern DICOM files that contain 1 volume-per-file: multi-frame enhanced Dicoms and Siemens Mosaic Dicoms.
Member of class: BrainVoyager.
Parameter 1: FirstFileName: name of the first "raw" data file.
Parameter 2: subj_id: subject ID for the document.
Parameter 3: ses_id: session ID for the FMR document.
Parameter 4: run_id: run ID for the FMR document.
Parameter 5: strTaskName: name of task
Parameter 6: project folder for the FMR document.
Parameter 7 (optional): strProtocolFile: name of protocol file (*.prt)
Returns: Document

2.2.1 List of functions

Table 2.1: FMR files

Description of function	*Function in BrainVoyager API*	*Since version*
Create document from raw data (use BV object)	CreateDocumentFMR(arguments)	BV QX 1.0
Create mosaic document from raw data (use BV object)	CreateDocumentMosaicFMR(arguments)	BV QX 1.0
Create document from raw DICOM data (use BV object)	CreateFMRFromDICOM(4 arguments)	BV 23.0
Create document from raw DICOM data, save as NIfTI-BIDS (use BV object)	CreateFMRFromDICOMAsNIFTIBIDS()	BV 23.0
Link AMR	LinkAMR(argument)	BV QX 1.6
Link PRT	LinkStimulationProtocol(argument)	BV QX 1.3
Link PRT	LinkProtocol(argument)	BV 23.0
Get/set whether protocol is parametric	ProtocolParametric (property)	BV 23.0
Get/set repetition time	TR (property)	
Get/set inter slice time	InterSliceTime (property)	
Get/set number of volumes	NrOfVolumes (property)	
Get/set number of skipped volumes	NrOfSkippedVolumes	BV 23.0
Get/set number of slices	NrOfSlices	BV 23.0
Get/set pixel size of x-dimension of image (mm)	PixelSizeOfSliceDimX (property)	
Get/set pixel size of y-dimension of image (mm)	PixelSizeOfSliceDimY (property)	
Get/set slice thickness (mm)	SliceThickness (property)	
Get/set slice gap (mm)	GapThickness (property)	
Get/set whether voxel resolution is verified	VoxelResolutionVerified (property)	
Get/set whether temporal resolution is verified	TimeResolutionVerified (property)	
Get filename of preprocessed file	FileNameOfPreprocessdFMR (property)	BV QX 1.0
Get filename of preprocessed file (incl. path)	FileNameOfPreprocessedFMR (property)	BV QX 2.3
Get filename of document	FileName (property)	BV 23.0
Get pathname of document	PathFileName (property)	BV 23.0
Remove preprocessed FMR document from disk	Remove()	x.x
Save FMR/DMR/VMR/AMR document to disk	SaveAs()	x.x
Save FMR	SaveFMRAndSTCFromMem()	BV 23.0
Check whether document is an FMR file	IsFMR()	BV 23.0

2.2.2 Scripts

Script to create a mosaic FMR document in Matlab

Below a script to use the `CreateDocumentMosaicFMR()` function.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create_fmr_document.m
% Script to create a functional mosaic document (*.fmr) in Matlab
% Works with BrainVoyager 22, updated May 2021
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% declare and initialize variables (change for own use)
[filename, pathname] = uigetfile('*.dcm');
firstfile = [pathname filename];
filetype = 'DICOM';
nrofvols = 291;
nrofvolsskip = 0;
createamr = 1;
nrslices = 64;
stcprefix = 'untitled';
swapbytes = 0;
mosaicx = 800;
mosaicy = 800;
bytesperpixel = 2;
nrofvolsinimg = 1;
imgx = 100;
imgy = 100;
% start (does not require change)
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')
fmr = bv.CreateDocumentMosaicFMR(filetype, firstfile, nrofvols, nrofvolsskip, createamr, nrslices, ...
stcprefix, swapbytes, mosaicx, mosaicy, bytesperpixel, pathname, nrofvolsinimg, imgx, imgy);
[pathstr,name,ext] = fileparts(firstfile);
fmr.SaveAs(fullfile(pathstr, [name '.fmr']));
fmr.Close;
```

2.3 Creating an anatomical document (*.vmr)

Scripting a VMR document is relatively straightforward, if the parameters are known.
The function `CreateDocumentVMR()` only requires 7 parameters:

1. The `filetype`, which is one of 'DICOM', 'SIEMENS', `'GE_I'`, `'GE_MR'`, `'PHILIPS_REC'` or 'ANALYZE'.
2. The first file name, including path.
3. A boolean value `true` or `false` indicating whether the data need to be swapped.
4. The number of slices.
5. The size of the image on the x-axis.
6. The size of the image on the y-axis.
7. The number of bytes per pixel. For anatomical data, this likely to be 1 or 2 bytes.

The function returns a VMR document object (see figure 2.1). If one would like to perform inhomogeneity (bias) correction, this needs to be performed directly after creation of the VMR document, using the function `CorrectIntensityInhomogeneities()`. The function does not take arguments.

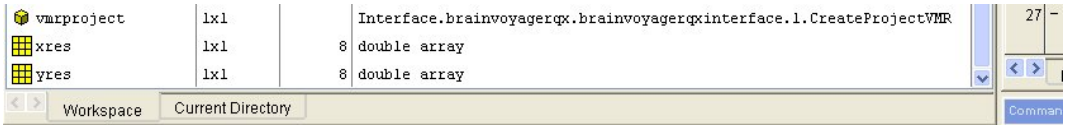


Figure 2.1: The VMR document object

CreateVMRFromDICOM()

Function: CreateVMRFromDICOM()
Creates an anatomical document in 8-bit (.vmr) and 16-bit (*.v16) from DICOM files.*
Parameter 1: FirstFileName: name of the first *.dcm file.
Returns: Document

CreateVMRFromDICOMAsNIFTIBIDS()

CreateVMRFromDICOMAsNIFTIBIDS()
Function: CreateVMRFromDICOMAsNIFTIBIDS(strFileOfSeries, subj_id, ses_id, run_id, strProjectFolder, deface_anat=false).
Description: Creates an anatomical document from DICOM data. This function creates NIFTI files in BIDS folder structure below the provided project folder (if without path, project will be created/used as sub-folder of "Documents/BrainVoyager/Projects". Since BrainVoyager 23.0.
Member of class: BrainVoyager.
Parameter 1: FirstFileName: name of the first "raw" data file.
Parameter 2: subj_id: subject ID for the document.
Parameter 3: ses_id: session ID for the VMR document.
Parameter 4: project folder for the VMR document.
Parameter 5 (optional): deface: true or false (default)
Returns: Document

2.3.1 List of functions

Table 2.2: VMR files

Description of function	*BrainVoyager API*	*Since version*
Create anatomical document from raw data (use BV object)	CreateDocumentVMR()	QX 1.0
Create anatomical document from raw DICOM data (use BV object)	CreateVMRFromDICOM()	BV 23.0
Create anatomical NIfTI-BIDS document from raw DICOM data (use BV object)	CreateVMRFromDICOMAsNIFTIBIDS()	BV 23.0
Make voxels isotropic	AutoTransformToIsovoxel()	BV QX 1.6
Make voxels isotropic	TransformToIsoVoxel()	QX 3.4 (BV20.4)
Transform to sagittal orientation	AutoTransformToSag()	BV QX 1.6
Transform VMR to MNI space	NormalizeToMNISpace	QX 3.4 (BV20.4)
Size of a voxel along x-dimension	VMRVoxelResolutionX	BV QX 2.0
Size of a voxel along y-dimension	VMRVoxelResolutionY	BV QX 2.0
Size of a voxel along z-dimension	VMRVoxelResolutionZ	BV QX 2.0
Size of a voxel along x-dimension	VMRVoxelSizeX	BV 23.0
Size of a voxel along y-dimension	VMRVoxelSizeY	BV 23.0
Size of a voxel along z-dimension	VMRVoxelSizeZ	BV 23.0
Set normal or extended Talairach box	ExtendedTALSpaceForVTCCreation	
Retrieve file name of attached functional (VTC) file	FileNameOfCurrentVTC	QX 2.2
Get/set repetition time (TR)	TR (property)	BV QX 1.0
Get voxel intensity for arguments x,y en z	GetVoxelIntensity()	
Set voxel intensity	SetVoxelIntensity()	1.0
Get/set whether protocol is parametric	ProtocolParametric (property)	BV 23.0
Get filename of document	FileName (property)	BV 23.0
Get pathname of document	PathFileName (property)	BV 23.0
Save FMR/DMR/VMR/AMR document to disk	SaveAs()	
Check whether document is an VMR file	IsVMR()	BV 23.0
Save screenshot of surface window	SaveScreenshotOfSurfaceWindow()	BV 23.0

2.3.2 Scripts

Script to create a VMR document via Matlab dialogs

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create_vmrdocument_via_interface.m
% Script to create a BrainVoyager VMR document in Matlab
% Should work with BrainVoyager 22
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')
h = msgbox('Please select the first file of the raw data set', 'Create VMR document');% See figure 2.2
uiwait(h);
cancel = 0;
% get first file % See figure 2.3
[filename, pathname] = uigetfile( ...
    {'*.dcm; *.IMA; *.I; *.MR; *.REC;*.hdr','Raw data files (*.dcm,*.IMA,*.I, *.MR, *.REC,*.hdr)';
    '*.dcm', 'DICOM files (*.dcm)'; ...
    '*.IMA','Siemens DICOM files (*.IMA)'; ...
    '*.I','GE I files (*.I)'; ...
    '*.MR','GE MR (*.MR)'; ...
    '*.REC','Philips files (*.REC)'; ...
    '*.hdr','Analyze (*.hdr)'}, ...
    'Please select the first file');
firstfilename = strcat(pathname, filename);
[pathstr,name,ext] = fileparts(firstfilename);
switch ext
    case '.dcm', filetype = 'DICOM'
    case '.IMA', filetype = 'SIEMENS'
    case '.I', filetype = 'GE_I'
    case '.MR', filetype = 'GE_MR'
    case '.REC', filetype = 'PHILIPS_REC'
    case '.hdr', filetype = 'ANALYZE'
    otherwise cancel = 1
end
if ~cancel
    % get parameters
    if (strcmp(ext, '.dcm') == 1) && (exist('dicominfo.m') > 0)
        dcminfo = dicominfo(firstfilename);
        bytesperpixel = (dcminfo.BitsAllocated/8);
        xres = dcminfo.Columns;
        yres = dcminfo.Rows;
    else
        prompt = {'Matrix size x:','Matrix size y:', 'Number of bytes per pixel:'}; % See figure 2.4
        dlgtitle = 'Parameters for VMR document';
        nrlines = 1;
        def = {'256','256', '2'};
        answer = inputdlg(prompt,dlgtitle,nrlines,def);
        xres = answer{1};
        yres = answer{2};
        bytesperpixel = answer{3};
    end
    button = questdlg('Are there only raw data files for a VMR document in this directory?',...% See figure 2.5
        'Create VMR script','Yes','No','Yes');
    if strcmp(button,'Yes')
        rawfileselection = [pathname ['*' ext]];
        files = dir(rawfileselection);
        sizeAr = size(files);
        nrslices = sizeAr(1);
    else
        answer = inputdlg('Please enter the number of slices'); % See figure 2.6
        nrslices = answer{1};
    end
    button = questdlg('Swap data?', 'Create VMR script','Yes','No','No'); % See figure 2.7
    if strcmp(button,'Yes'), swap = 1; else swap = 0; end;
    vmrdoc = bv.CreateDocumentVMR(filetype, firstfilename, nrslices, swap, xres, yres, bytesperpixel);
    success = vmrdoc.SaveAs(fullfile(pathname, [name '.vmr']));
end
```

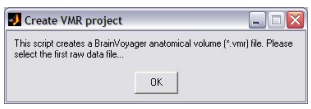


Figure 2.2: Announcement at beginning of script

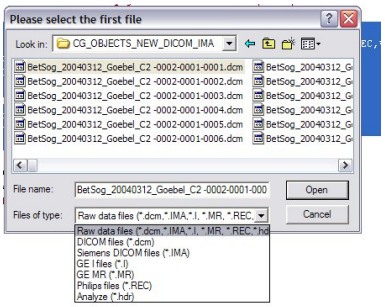


Figure 2.3: Select the first file

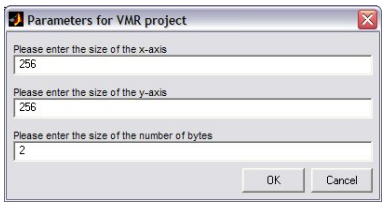


Figure 2.4: Enter parameters

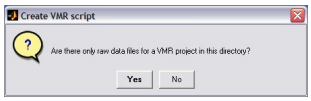


Figure 2.5: Enter the number of slices

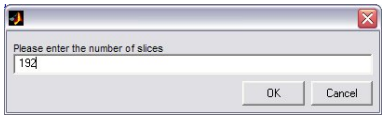


Figure 2.6: Enter the number of slices

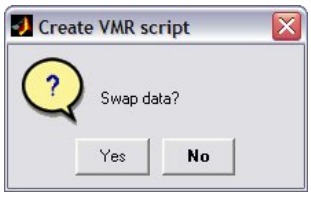


Figure 2.7: Question whether swapping is necessary

Script to display a VMR slice in Matlab

Below a script to use the `GetVoxelIntensity(x,y,z)` function to display a slice of an anatomical file (see figure 2.8). For an alternative that reads the whole file, use the BVQXtools/NeuroElf to read binary BrainVoyager files in Matlab (please see chapter 8).

```
% bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')
[FileName,PathName] = uigetfile('*.vmr','Please select a VMR file');
vmrfilename = [PathName FileName];
vmr = bvqx.OpenDocument(vmrfilename);
bvqx.PrintToLog('Get voxel intensities in Matlab...');
midslice = round(vmr.DimZ/2)
for j = 1:vmr.DimY
    for i = 1:vmr.DimX
        vmrdata(i,j) = vmr.GetVoxelIntensity(i,j,midslice);
    end
end
imagesc(vmrdata)
colormap gray
title(['VMR slice ' num2str(midslice)])
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% display_vmr_slice_in_matlab.m
% Script to display a slice of a BrainVoyager VMR project in Matlab
% Works with BrainVoyager QX 2.2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%bvqx = actxserver('BrainVoyagerQX.BrainVoyagerQXScriptAccess.1') % BVQX 2.2
[FileName,PathName] = uigetfile('*.vmr','Please select a VMR file');
vmrfilename = [PathName FileName];
vmr = bvqx.OpenDocument(vmrfilename);
bvqx.PrintToLog('Get voxel intensities in Matlab...');
midslice = round(vmr.DimZ/2)
for j = 1:vmr.DimY
    for i = 1:vmr.DimX
        vmrdata(i,j) = vmr.GetVoxelIntensity(i,j,midslice);
    end
end
imagesc(vmrdata)
colormap gray
title(['VMR slice ' num2str(midslice)])
```

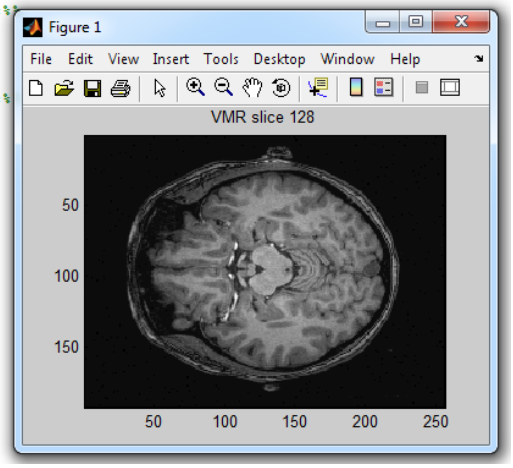


Figure 2.8: Script and resulting VMR slice in Matlab

Script to filter a VMR in Matlab

This script applies a Laplacian of a Gaussian, a Sobel (approximating a gradient) or a Canny filter on a VMR document and displays it in BrainVoyager.

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% filter_vmr_in_matlab.m
% Script to filter a BrainVoyager VMR document in Matlab
%
% Works with BrainVoyager QX 1.9 and VMR versions higher than 1.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    [FileName,PathName] = uigetfile('*.vmr','Please select a VMR file');
vmrfilename = [PathName FileName];
fid = fopen(vmrfilename, 'rb');
version = fread(fid, [1 1], 'uint16');
dimx = fread(fid, [1 1], 'uint16');
dimy = fread(fid, [1 1], 'uint16');
dimz = fread(fid, [1 1], 'uint16');
data = fread(fid, [1 dimx*dimy*dimz], 'uchar');
rest = fread(fid);
fclose(fid);
vmrdata = reshape(data, dimx, dimy, dimz);

    %  H = FSPECIAL('log',HSIZE,SIGMA) returns a rotationally symmetric
%     Laplacian of Gaussian filter of size HSIZE with standard deviation
%     SIGMA (positive). HSIZE can be a vector specifying the number of rows
%     and columns in H or a scalar, in which case H is a square matrix.
%     The default HSIZE is [5 5], the default SIGMA is 0.5.

    % show filters %(see figure 2.9)
figure;
img = vmrdata(:,:,floor(dimx/2));
img = rot90(img, 3);
subplot(2,2,1);imagesc(img);title('Original Image');
H = fspecial('log',[5 5],.5);
Laplacian = imfilter(img,H);
subplot(2,2,2);imagesc(Laplacian);title('Laplacian of Gaussian filter');
cannyimg = edge(img,'canny',.15);
subplot(2,2,3);imagesc(cannyimg);title('Canny filter (0.15)');
H = fspecial('sobel');
sobelimg = imfilter(img,H);
subplot(2,2,4);imagesc(sobelimg);title('Sobel filter');
colormap gray;

    filterlist = {'Laplacian', 'Canny', 'Sobel'}; %(see figure 2.10)
[filter,ok] = listdlg('PromptString','Select a filter:',...
'SelectionMode','single',...
'Name', 'Filter VMR',...
'ListString',filterlist);

    H_log = fspecial('log',[5 5],.5);
H_sobel = fspecial('sobel');

    if filter == 2 %(see figure 2.11)
 prompt = {'Enter canny filter value:'};
    dlg_title = 'Filter VMR';
    num_lines= 1;
    def     = {'0.15'};
    answer  = inputdlg(prompt,dlg_title,num_lines,def);
    filterval = answer{1};
end

    % apply filter and write image
[pathstr,name,ext,versn] = fileparts(vmrfilename);
filteredimgname = fullfile(pathstr, [name '_filtered' ext]);
if ok == 1
    % write VMR header
    fid = fopen(filteredimgname,'w');
    fwrite(fid,version,'ushort');
    fwrite(fid,dimx,'ushort');
    fwrite(fid,dimy,'ushort');
    fwrite(fid,dimz,'ushort');
    % write VMR image volume
    for slicenr=1:len(3)
        img = vmrdata(:,:,slicenr);
        if filter==1
            filteredimg = imfilter(img,H_log);
        elseif filter ==3
            filteredimg = imfilter(img, H_sobel);
        elseif filter == 2
            filteredimg = edge(img,'canny',filterval);
```

```
        else
            break;
        end
        fwrite(fid,filteredimg,'uchar');
    end
    % write rest of VMR header
    fwrite(fid,rest);
    fclose(fid);
end

    % show in BrainVoyager
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1');
filteredvmr = bv.OpenDocument(filteredimgname);
bvqx.ShowLogTab;
bvqx.PrintToLog(['Filter: ' filterlist{filter}]);
```

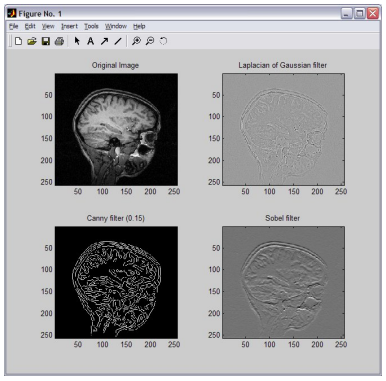


Figure 2.9: Show filters

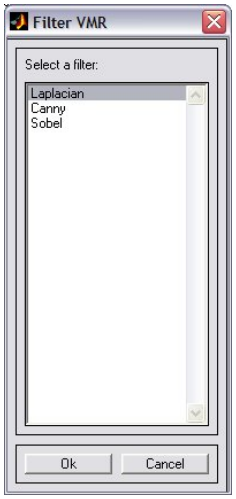


Figure 2.10: Select filter dialog

Figure 2.11: Enter canny filter value

2.4 Creating AMR documents

The parameters for `CreateDocumentAMR()` are:
Parameter: Filetype (string): file type of original data. One of `"DICOM"`, `"SIEMENS"`, `"GE_I"` (no parameters for +20 logic like in GUI),`"GE_MR"`, `"PHILIPS_REC"` or `"ANALYZE"`.
Parameter: firstFile (string): the filename and path of the first file of the data.
Parameter: nrOfSlices (integer): the number of slices in a volume.
Parameter: isLittleEndian (boolean): Is 'true' when the byte order is little endian; otherwise 'false'.
Parameter: xSize (integer): Size of image along x-axis. Example value: 256.
Parameter: ySize (integer): Size of image along y-axis. Example value: 256.
Parameter: nrOfBytes (integer): Number of bytes for each pixel. 1 byte is 8 bits. Example value: 2.
Returns: AMR document object

2.4.1 List of functions

Table 2.3: AMR files

Description of function	*BrainVoyager API*	*Since version*
Create document from raw data	CreateDocumentAMR()	BV QX 1.0

2.5 Creating a diffusion weighted image document (*.dmr)

Diffusion weighted data can be treated in the same fashion as anatomical and functional data, via the creation of a BrainVoyager document.

2.5.1 List of functions

Table 2.4: DMR files

Description of function	*Function in BrainVoyager API*	*Since version*
Create document from DWI data (use BV object)	CreateDocumentDMR()	BV QX 1.9
Create mosaic document from DWI data (use BV object)	CreateDocumentMosaicDMR()	BV QX 1.9
Create document from DWI data (use BV object)	CreateDMRFromDICOM()	BV 23.0
Create document from DWI data, save as NIfTI-BIDS (use BV object)	CreateDMRFromDICOMAsNIFTIBIDS()	BV 23.0
Link AMR	LinkAMR()	BV QX 1.6
Get data	TR (property)	
	InterSliceTime (property)	
	NrOfVolumes (property)	
	PixelSizeOfSliceDimX (property)	
	PixelSizeX (property)	BV 23.0
	PixelSizeOfSliceDimY (property)	
	PixelSizeY (property)	BV 23.0
	SliceThickness (property)	
	GapThickness (property)	
	VoxelResolutionVerified (property)	
	TimeResolutionVerified (property)	
Get filename of document	FileName (property)	BV 23.0
Get pathname of document	PathFileName (property)	BV 23.0
Save FMR/DMR/VMR/AMR document to disk	SaveAs()	x.x
Check whether document is an DMR file	IsDMR()	BV 23.0

2.5.2 Create DMR documents

CreateDocumentDMR()

Function: CreateDocumentDMR()
Parameter 1: Filetype (string): file type of original data. One of `"DICOM"`, `"PHILIPS_REC"` or `"ANALYZE"`.
Parameter 2: firstFile (string): the filename and path of the first file of the data.
Parameter 3: Number of directions (integer): the number of directions (=volumes).
Parameter 4: Number of directions to skip (integer): the number of directions (=volumes) to skip.
Parameter 5: Create AMR (boolean): should an AMR document be created: true or false. This is for visualization purposes.
Parameter 6: nrOfSlices (integer): the number of slices in a volume.
Parameter 7: Prefix (string): Name for DWI file.
Parameter 8: BigEndianByteOrder (boolean): set big or little endian flag.
Parameter 9: nrOfBytes (integer): Number of bytes for each pixel. 1 byte is 8 bits. Example value: 2.
Parameter 10: xSize (integer): Size of image along x-axis. Example value: 128.

Parameter 11: ySize (integer): Size of image along y-axis. Example value: 128.
Parameter 12: Number of bytes per pixel (integer): Precision of intensity value. Usually 2 byte (16 bits).
Parameter 13: Saving directory (string): Name of path where DMR document should be saved.
Returns: Document

CreateDocumentMosaicDMR()

Function: CreateDocumentMosaicDMR()
Member of class: BrainVoyager.
Parameter 1: FileType (string): file type of original data. One of `"DICOM"`, `"SIEMENS"`, `"GE_I"` (no parameters for +20 logic like in GUI),`"GE_MR"`, `"PHILIPS_REC"` or `"ANALYZE"`.
Parameter 2: FirstFileName: provide first name of run including path
Parameter 3: NrOfDirections (number of volumes) per run
Parameter 4: nrOfVolumesToSkip: enter the number of volumes at the beginning of the run that need to be discarded; otherwise enter 0.
Parameter 5: CreatePseudoAMR: create *.amr file (bitmaps of size 256 x 256) of the first volume, which maybe be T1-saturated, if it was not discarded at the scanner; can be useful for intensity-based fine alignment or visualization purposes.
Parameter 6: nrOfSlices: provide the number of slices per volume
Parameter 7: prefixSTCs: provide a prefix for the *.stc files or provide empty string; can be overwritten when using `SaveAs()`
Parameter 8: BigEndianByteOrder: set big endian or little endian flag (most systems are little endian these days)
Parameter 9: mosaicResX: mosaic size: dimension of images in volume along x-axis
Parameter 10: mosaicResY: mosaic size: dimension of images in volume along y-axis
Parameter 11: nrBytes: number of bytes per pixel (usally 2)
Parameter 12: savingDir: target directory for DMR project
Parameter 13: volsInImg: number of volumes per file (for mosaic images)
Parameter 14: resX: dimension of image along x-axis
Parameter 15: resY: dimension of image along y-axis
Returns: Document

CreateDMRFromDICOM()

CreateDMRFromDICOM() *Function:* CreateDMRFromDICOM(FirstFileName, prefixDWIs, savingDir, strDirectionsFile="").
Description: DMR documents consist of a set of functional data in the original "slice space". This convenient scripting function is only for modern DICOM files that contain 1 volume-per-file: multi-frame enhanced Dicoms and Siemens Mosaic Dicoms.
Member of class: BrainVoyager.
Parameter 1: FirstFileName: name of the first "raw" data file.
Parameter 2: prefixDWIs: name for the *.dwi file.
Parameter 3: savingDir - directory for saving the DMR document.
Parameter 4 (optional): strDirectionsFile: name of the directions file (*.grb)
Returns: Document

CreateDMRFromDICOMAsNIFTIBIDS()

CreateDMRFromDICOMAsNIFTIBIDS() *Function:* CreateDMRFromDICOMAsNIFTIBIDS(strFileOfSeries, subj_id, ses_id, run_id, strProjectFolder, strDirectionsFile="").
Description: DMR documents consist of a set of functional data in the original "slice space". This function creates NIFTI files in BIDS folder structure below the provided project folder (if without path, project will be created/used as sub-folder of "Documents/BrainVoyager/Projects". Since BrainVoyager 23.0. This convenient scripting function is only for modern DICOM files that contain 1 volume-per-file: multi-frame enhanced

Dicoms and Siemens Mosaic Dicoms.
Member of class: BrainVoyager.
Parameter 1: FirstFileName: name of the first “raw” data file.
Parameter 2: subj_id: subject ID for the document.
Parameter 3: ses_id: session ID for the DMR document.
Parameter 4: run_id: run ID for the DMR document.
Parameter 5: project folder for the DMR document.
Parameter 6 (optional): strDirectionsFile: name of directions file (*.grb)
Returns: Document

2.5.3 Scripts

Script to create diffusion weighted documents (*.dmr) and normalized files (*.vdw)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% diffusion_weighted_data.m
% Assumes: a) all data are in same folder b) for VDW creation: data are
% aligned and anatomical volume is transformed to all spaces (please
% comment lines out if not the case)
%
% Usage: load in BrainVoyager, click 'Run'. Pop-up boxes will appear to ask
% whether to create a diffusion weighted document, and to create VDW files.
%
% Works with BrainVoyager 22
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')
bv.PrintToLog('Start processing diffusion weighted data...');
% òòò this part will require modifications, if used for own purposes òòò
dmrname = 'D:/BVData/diffusion/human31dir_from_script.dmr'; % this variable is also used in 2nd part
answer = questdlg('Create diffusion weighted document?', 'Create DMR');
if strcmp(answer,'Yes')
    dmr = bv.CreateDocumentDMR( 'DICOM', ...
        'D:/BVData/diffusion/pimpul_070907_dti -0007-0001-00001.dcm', ...
        31, 0, true, 75, 'human31dir', false, 128, 128, 2, 'D:/BVData/diffusion');
    dmr.TR = 8900;
    dmr.InterSliceTime = 118;
    dmr.TimeResolutionVerified = 1;
    dmr.SaveAs(dmrname);
end
% òòò this part does not require modifications, is all user-interface based òòò
% if alignment files happen to be available, one could as well create
% normalized diffusion weighted data files:
answer = questdlg('Create normalized diffusion weighted data? (requires *.trf files)', 'Create VDW');
if strcmp(answer, 'Yes')
    [pathstr, name, ext] = fileparts(dmrname)
    [FileName, PathName] = uigetfile('*.vmr','Please select the anatomical file...');
    vmr = bv.OpenDocument([PathName FileName]);
    % assumes a single initial alignment file in the directory of the VMR
    iafilename = dir([PathName '*_IA.trf'])
    % assumes a single fine alignment file in the directory of the VMR
    fafilename = dir([PathName '*_FA.trf'])
    % assumes a single AC-PC transformation file in the directory of the VMR
    acpcfilename = dir([PathName '*ACPC.trf'])
    % assumes a single Talairach landmark file in the directory of the VMR
    talfilename = dir([PathName '*.tal'])
    answer = questdlg('Would you like 32-bit float datatype? (Otherwise 16-bit integer)', 'Datatype');
    if strcmp(answer, 'Yes')
        datatype = 2;
    else
        datatype = 1;
    end
    resolutionlist = {'1mm3', '2mm3', '3mm3'};
    [resolution, ok] = listdlg('PromptString','Select the target VTC resolution:',...
        'SelectionMode','single',...
        'Name', 'VDW resolution',...
        'ListString', resolutionlist);
    interpolationlist = {'nearest neighbor', 'trilinear', 'sinc'};
    [interpolation, ok] = listdlg('PromptString','Select the interpolation type:',...
        'SelectionMode','single',...
        'Name', 'Interpolation',...
        'ListString', interpolationlist)
    interpolation = interpolation - 1; % 0: nearest neighbour, 1: trilinear interpolation, 2: interpolation.
    prompt = {'Enter intensity threshold:'};%,'Enter colormap name:'};
    dlg_title = 'Intensity threshold';
    num_lines = 1;
    def = {'100'};%,'hsv'};
    threshold = inputdlg(prompt,dlg_title,num_lines,def);
    ia = fullfile(PathName, iafilename(1).name) % takes first initial alignment file it finds in folder
    fa = fullfile(PathName, fafilename(1).name) % takes first fine alignment file it finds in folder
    acpc = fullfile(PathName, acpcfilename(1).name) % takes first AC-PC alignment file it finds in folder
    tal = fullfile(PathName, talfilename(1).name) % takes first Talairach landmarks file it finds in folder
    thresh = str2num(threshold{1});
    vmr.ExtendedTALSpaceForVTCCreation = 0; % no extended bounding box for cerebellum
    vdw = fullfile(pathstr, [name '_native.vdw'])
    success = vmr.CreateVDWInVMRSpace(dmrname,ia,fa,vdw,datatype,resolution, interpolation, thresh);
    vdw = fullfile(pathstr, [name '_acpc.vdw'])
    success = vmr.CreateVDWInACPCSpace(dmrname,ia,fa,acpc,vdw,datatype,resolution, interpolation, thresh);
    vdw = fullfile(pathstr, [name '_tal.vdw'])
    success = vmr.CreateVDWInTALSpace(dmrname,ia,fa,acpc,tal,vdw,datatype,resolution, interpolation, thresh);
    end
```

```
bv.PrintToLog('Finished processing diffusion weighted data.');
```

Chapter 3

Preprocessing functional data

In this section will be described how to preprocess functional data in BrainVoyager from Matlab.

3.1 Preprocessing functional data (*.fmr)

3.1.1 Mean intensity adjustment

Volume-based adjustment of mean intensity (MIA) can be performed with the function AdjustMeanIntensity(). During operation, the program plots two curves, one showing the measured mean intensity of volumes over time and the other showing the mean level of each volume after correction (a straight line). Furthermore, a zero-mean predictor of the global fluctuations is automatically stored to disk allowing to add it as a confound predictor in the design matrix. If the MIA preprocessed FMR is used for further processing (e.g., VTC creation), adding of the MIA confound predictor is not necessary. If one wants to perform the correction as part of the GLM, however, one should use the non-MIA FMR for further processing adding the MIA confound predictor to the design matrix.

3.1.2 Slice scan time correction

There are three slice scan time correction functions. The first slice scan time correction method, `CorrectSliceTiming()`, accepts three parameters, which are the slice order, the type of interpolation and the multiband factor (new since ± BrainVoyager 22).

Slice scan time correction with slice order and interpolation arguments

The slice order parameter can have the following values:

Table 3.1: Slice order options

Slice order	*Value*
Ascending	0
Ascending interleaved	1
Ascending interleaved 2	2
Descending	10
Descending interleaved	11
Descending interleaved 2	12

The interpolation can take the following values:

Table 3.2: Interpolation options

Interpolation type	*Value*
Trilinear	0
Cubic spline	1
Sinc	2

The multi-band factor specifies how the proportion of slices acquired simultaneously (MB-EPI). If no multi-band has been applied, use '1'.

Perform the slice scan time correction from Matlab via

```
success = fmr.CorrectSliceTiming(1, 0, 1);
```

Obtain the name of the slice scan time corrected project via

```
newfmrname = fmr.FileNameOfPreprocessedFMR;.
```

Close the non-preprocessed FMR via

```
fmr.Close;.
```

Open the slice scan time corrected project via

```
newfmr = fmr.OpenDocument(newfmrname);
```

Slice scan time correction with specified slice order

In the first argument of the function `CorrectSliceTimingWithSliceOrder` the order of the slices is provided via a text string:

```
fmr = BrainVoyager.ActiveDocument;
fmr.CorrectSliceTimingWithSliceOrder("1 14 2 15 3 16 4 17 5 18 6 19 7 20 8 21 9 22 10 23 11 24 12 25 13", 1, 1);
```

The second argument is interpolation (0 = trilinear, 1 = cubic spline, 2 = sinc) and the third the multiband factor (new since ± BrainVoyager 22)

Slice scan time correction for multi-band EPI

The slice-scan time correction for multi-band sequences that acquire 2 or more slices in a single shot can be automated using the function CorrectSliceTimingUsingTimeTable(). For Siemens Mosaic data files, slice timing information is now directly extracted from the DICOM header; since this information (time of acquistion for each slice with respect to the begin of a volume) can be used to correct slice timing differences for all 2D EPI sequences (e.g. with ascending, descending, interleaved slice order with or without multi-band with or without extra (silent) gaps between TRs), a new "slice time table" option is now used as default if the respective data is available. In order to be available for preprocessing, the extracted slice timing data (one value per slice) is permanently stored in created .FMR files. The availability of time table can be interrogated using the `HasSliceTimeTable` property)(function and property new in BVQX 2.8.2).

```
if (docFMR.HasSliceTimeTable)
docFMR.CorrectSliceTimingUsingTimeTable(1); % 1: cubic spline interpolation
else
docFMR.CorrectSliceTiming(2, 1); % 2: ascending interleaved 2, 1: cubic spline interpolation
end
```

3.1.3 Motion correction

In BrainVoyager, there are two methods to perform motion correction within one run. These are `CorrectMotion()` and `CorrectMotionEx()`. In the latter method, the number of parameters that can be used for the motion correction is extended.

If multiple runs are performed in one session, the run closest to the 3D scan would be corrected by using the single-run version: `CorrectMotion(int TargetVolume)`, i.e. with a value of `1` for the parameter

TargetVolume. All subsequent runs would then be corrected by specifying the name of the first run as target as well as the same volume as specified for the first run. This ensures that all volumes of all runs are aligned to the same target volume.

If CorrectMotion(<target volume nr>) is used, the applied method is trilinear detection and sinc interpolation. This can be inspected in the *_3DMC.log file. This can also be seen from the resulting filename, containing '3DMCTS', where the 'T' stands for trilinear detection and the 'S' for sinc interpolation. The full data set will be used, and the default number of iterations is set (100). No extended log file is created.

In the version with extended parameters, the parameters are the following:

```
CorrectMotionEx(Target volume (number), ...
Interpolation method (0 or 1),...
 Full data set (true or false), ...
Max nr of iterations (number),...
Movies (true or false),...
Extended log (true or false));
```

(Replace the names by real parameters and remove spaces in the names).

3.1.4 Motion correction and intrasession alignment

To combine motion correction with intra-session alignment, the methods CorrectMotionTargetVolumeInOtherRun() and CorrectMotionTargetVolumeInOtherRunEx() are available.

The CorrectMotionTargetVolumeInOtherRun(filename, volume) accepts the following 2 parameters:

1. target fmr file name
2. target volume number

When this function is applied, a reduced data set will be used, and the default number of iterations is set (100). No extended log file is created.

Its use can be illustrated in the following way:

```
success = fmr2.CorrectMotionTargetVolumeInOtherRun('C:/Data/ObjectsExperiment.fmr', 1);
```

The CorrectMotionTargetVolumeInOtherRunEx() accepts the following 7 parameters:

1. target fmr file name
2. target volume number usually the first volume, so 1.
3. interpolation method use 1: trilinear
4. use full data set if 1: yes, if 0: no
5. maximum number of iterations this determines how often the estimates should be changed in order to find the values for the rotation and translation parameters; in the BrainVoyager user interface, this value is default set to 100.
6. create movies these *.avi files cannot be generated yet, so the parameter can be set to 0.
7. create extended log file this is very useful. if 1: yes, if 0: no.

These parameters are the same as for CorrectMotionEx() method except for the extra TargetFMRFileName parameter used here.

3.1.5 Interpolation differences

When invoking `CorrectMotion()`, the trilinear detection and windowed sinc function for correction is used. When using `CorrectMotionEx()`, the trilinear detection and correction is applied. This is also the case for the motion correction functions including intra session alignment, i.e.
`CorrectMotionTargetVolumeInOtherRun()` and
`CorrectMotionTargetVolumeInOtherRunEx()`.

The applied interpolation methods are reflected in the resulting FMR filename. The 'T' means just trilinear for both detection and correction. The characters 'TS' are used in the FMR filename for trilinear detection with sinc (windowed) correction.

3.1.6 Temporal filtering

Linear trend removal

Linear trend removal is automatically included when applying high pass filtering to the data. In case it should be applied independently,
`success = newfmr.LinearTrendRemoval;`
can be used.

Temporal high-pass filtering

There are two approaches for temporal high pass filtering: the classical "frequency-domain" approach and the "GLM" approach using Fourier or discrete cosine transform basis functions.

High-pass filtering: "Frequency-domain" approach

The high pass filter is applied in the frequency domain. The temporal high-pass filter can be specified in cycles or in Hz.
When it is specified in cycles, is this relative to the number of time points specified in the functional data, i.e. length of the time series. When the units are specified in `Hz`, it is in signal intensity changes per Hz, for example:

```
success = newfmr.TemporalHighPassFilter(0.115385, 'Hz');
```

The units in Hertz can be computed from the cycles per time course via: number of cycles / number of data points.

High-pass filtering: GLM with Fourier or DCT basis functions

With Fourier basis functions: `TemporalHighPassFilterGLMFourier()`.
With discrete cosine transform basis functions: `TemporalHighPassFilterGLMDCT()`.
The functions take one parameter that specifies the number of cycles (pairs of two basis functions) used to build an appropriate design matrix.

Temporal Gaussian smoothing

Temporal Gaussian smoothing in FMR projects can be applied by specifying the width of the smoothing kernel in seconds ('s') or in TR ('TR'), for example:

```
newfmr.TemporalGaussianSmoothing(20, 's');
```

Spatial smoothing

Specification of the smoothing kernel for the FMR project is possible up to two decimals. When more values are specified, they will be rounded (1.06667 pixels round to 1.07). The units can be specified in pixels, 'px' or millimeters, 'mm':

```
newfmr.SpatialGaussianSmoothing(4, 'mm');
```

and

```
newfmr.SpatialGaussianSmoothing(1.06667, 'px');
```

The resulting files will be placed in the same folder as the source files. The filenames will contain the `_SD3DSS` appendix.

3.1.7 List of functions

Table 3.3: Preprocessing of FMR files

Description of function	*BrainVoyager API*	*Since version*
Get new file name	FileNameOfPreprocessdFMR (property)	BV QX 1.0
Mean intensity adjustment	AdjustMeanIntensity()	BV QX 2.8.2
Slice time correction	CorrectSliceTiming()	BV QX 1.5
Slice scan time correction with specified slice order	CorrectSliceTimingWithSliceOrder()	BV QX 2.3
Slice scan time correction for MB-EPI	docFMR.CorrectSliceTimingUsingTimeTable()	BV QX 2.8.2
Motion correction	CorrectMotion(targetvolume)	
Motion correction (extra parameters)	CorrectMotionEx(params)	BV QX 1.0
Motion correction and intra-session alignment	CorrectMotionTargetVolumeInOtherRun(target FMR)	
Motion correction and intra-session alignment (extra parameters)	CorrectMotionTargetVolumeInOtherRunEx()	BV QX 1.2
Low pass filter/smooth	TemporalGaussianSmoothing()	BV QX 1.0
Spatial Gaussian smoothing	SpatialGaussianSmoothing()	
Linear high pass filter	LinearTrendRemoval()	BV QX 1.0
Non-linear (and linear) high pass filter (frequency domain)	TemporalHighPassFilter()	
High pass filter with Fourier basis functions	TemporalHighPassFilterGLMFourier()	BV QX 2.4.1
High pass filter with discrete cosine transform basis functions	TemporalHighPassFilterGLMDCT()	BV QX 2.4.1

3.2 Preprocessing of functional normalized data (*.vtc)

Since BrainVoyager QX 2.2, it is possible to apply temporal and spatial filtering to functional normalized data (*.vtc).

3.2.1 List of functions

Table 3.4: Preprocessing of VTC files

Description of function	*Function in BrainVoyager API*	*Since version*
Spatial low-pass filter	SpatialGaussianSmoothing	BV QX 2.2
Temporal low pass filter/smooth	TemporalGaussianSmoothing()	BV QX 2.2
Non-linear (and linear) temporal high-pass filter	TemporalHighPassFilter()	BV QX 2.2
Linear temporal high-pass filter	LinearTrendRemoval()	BV QX 2.2

3.2.2 Detailed description of methods

SpatialGaussianSmoothing()

Function: SpatialGaussianSmoothing() (VTC)
Description: Spatial low-pass filter for VTC file; removes spatial high-frequency elements in VTC file, like sharp edges.
Parameter 1: FWHM value (number)
Parameter 2: FWHM unit: “mm” or “vx”

LinearTrendRemoval()

Function: LinearTrendRemoval() (VTC)
Description: Removes temporal linear trends in VTC file. A linear trend is estimated by fitting a line through the data estimating its slope and intercept.

TemporalHighPassFilter()

Function: TemporalHighPassFilter() (VTC)
Description: Removes low-frequency noise in VTC file, for example physiological noise.
Parameter 1: High-pass filter value
Parameter 2: High-pass filter unit: “cycles” or “Hz”

TemporalGaussianSmoothing()

Function: TemporalGaussianSmoothing() (VTC)
Description: Removes high-frequency noise in VTC file, like sharp peaks in the timecourse.
Parameter 1: FWHM value (number)
Parameter 2: FWHM unit: “d” or “dps” (data points) or “s” or “secs” (seconds)

3.2.3 Scripts

Preprocessing a functional data file (*.fmr)

This script applies preprocessing to a functional data (*.fmr) file.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% preprocess_fmr.m
% Preprocess a previously created functional data file (*.fmr). Here we open the "CG_OBJECTS_SCRIPT.fmr" file.
% Data: BrainVoyager Getting Started Guide 4
% Works with BrainVoyager 22
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')
[FileName PathName] = uigetfile('*.fmr') % invoke file dialog
docFMR = bv.OpenDocument([PathName FileName]);
% Set spatial and temporal parameters relevant for preprocessing
% This can be skipped if these values are set when reading the data
% To check whether these values have been set already (i.e. from header),
% use the "VoxelResolutionVerified" and "TimeResolutionVerified" properties
if ~docFMR.TimeResolutionVerified
    docFMR.TR = 2000;
    docFMR.InterSliceTime = 31;
    docFMR.TimeResolutionVerified = 1;
end
if ~docFMR.VoxelResolutionVerified
    docFMR.PixelSizeOfSliceDimX = 2.0;
    docFMR.PixelSizeOfSliceDimY = 2.0;
    docFMR.SliceThickness = 2.0;
    docFMR.GapThickness = 0.0;
    docFMR.VoxelResolutionVerified = 1;
end
% We save the new settings into the FMR file
docFMR.Save;
% Preprocessing step 1: mean intensity adjustment (optional)
docFMR.AdjustMeanIntensity();
ResultFileName = docFMR.FileNameOfPreprocessdFMR;
docFMR.Close;
docFMR = bv.OpenDocument( ResultFileName );
% Preprocessing step 2: Slice time correction
if (docFMR.HasSliceTimeTable)
    docFMR.CorrectSliceTimingUsingTimeTable(1); % 1: cubic spline interpolation
else
    docFMR.CorrectSliceTiming( 2, 0 ); % GSG4: ascending interleaved 2, MB 2
% First param: Scan order 0 -> Ascending, 1 -> Asc-Interleaved, 2 -> Asc-Int2,
% 10 -> Descending, 11 -> Desc-Int, 12 -> Desc-Int2
% Second param: Interpolation method: 0 -> trilinear, 1 -> sinc
end
ResultFileName = docFMR.FileNameOfPreprocessdFMR;
docFMR.Close;
docFMR = bv.OpenDocument( ResultFileName );
% Preprocessing step 3: 3D motion correction
docFMR.CorrectMotion(1);
ResultFileName = docFMR.FileNameOfPreprocessdFMR;
% the current doc (input FMR) knows the name of the automatically saved output FMR
docFMR.Remove; % close or remove input FMR
bv.PrintToLog('Removed slice scan time corrected files instead of just closing...')
% docFMR.Close(); // close input FMR
docFMR = bv.OpenDocument( ResultFileName );
% Open motion corrected file (output FMR) and assign to our doc var
% Preprocessing step 4: Spatial Gaussian Smoothing (not recommended
% for individual analysis with a 64x64 matrix)
docFMR.SpatialGaussianSmoothing( 4, 'mm'); % FWHM value and unit
ResultFileName = docFMR.FileNameOfPreprocessdFMR;
docFMR.Close; % docFMR.Remove(); % close or remove input FMR
docFMR = bv.OpenDocument( ResultFileName );
% Preprocessing step 5: Temporal High Pass Filter, includes Linear Trend
% Removal
docFMR.TemporalHighPassFilterGLMFourier(3);% cycles
ResultFileName = docFMR.FileNameOfPreprocessdFMR;
docFMR.Close; % docFMR.Remove(); // close or remove input FMR
docFMR = bv.OpenDocument( ResultFileName );
% Preprocessing step 6: Temporal Gaussian Smoothing (not recommended for
% event-related data)
docFMR.TemporalGaussianSmoothing( 3, 's'); % default value in BV
ResultFileName = docFMR.FileNameOfPreprocessdFMR;
docFMR.Close; % docFMR.Remove(); % close or remove input FMR
docFMR = bv.OpenDocument( ResultFileName );
```

Intra-session alignment

This script aligns the volumes in the source functional file to the first volume in the target file.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% intra_session_alignment.m: script to align a functional file to another
% intra-session alignment and motion correction
% Checked with BrainVoyager 22
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')
[FileName1, PathName1] = uigetfile('*.fmr','Please select the source functional file...');
sourcefmrname = fullfile(PathName1, FileName1);
[FileName2, PathName2] = uigetfile('*.fmr','Please select the target functional file...');
targetfmrname = fullfile(PathName2, FileName2);
sourcefmr = bv.OpenDocument(sourcefmrname);
sourcefmr.CorrectMotionTargetVolumeInOtherRunEx(targetfmrname, 1, 1, 1, 100, 0, 1);

% Parameter 1: Target FMR name: name of the run to which the current FMR project should be aligned.
% Parameter 2: Target volume: number of the volume to which other volumes should be aligned.
% Parameter 3: Interpolation method: 0 and 1: trilinear detection and trilinear interpolation, 2: trilinear
% detection and sinc interpolation or 3: sinc detection of motion and sinc interpolation.
% Parameter 4: Use full data set: true if yes, false if one would like to use the reduced dataset (default in
% GUI).
% Parameter 5: Maximum number of iterations: defines for how many iterations the parameters should be
% fitted. Value in GUI is default ë100í.
% Parameter 6: Generate movies: true if yes, false if no. This feature has been disabled for some time.
% Parameter 7: Generate extended log file: true if one would like the motion estimation parameters in a
% text file, false otherwise.
```

Preprocessing a normalized functional data file (*.vtc)

This script applies preprocessing to a volume time course (*.vtc) file.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Script to preprocess a VTC file in Matlab
% preprocess_vtc.m
% Works in BrainVoyager 22.2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')
bv.ShowLogTab;
bv.PrintToLog('Preprocessing VTC files from Matlab...');
doc = bv.ActiveDocument;
if (isempty(doc))
[FileName,PathName] = uigetfile('*.vmr', 'Please select the VMR file ');
vmr = bv.OpenDocument([PathName FileName]);
end
vtc = vmr.FileNameOfCurrentVTC;
if (isempty(vtc))
[FileName,PathName] = uigetfile('*.vtc', 'Please select the VTC file ');
vmr.LinkVTC([PathName FileName]);
end
% now smooth VTC with a large kernel of 10 mm:
vmr.SpatialGaussianSmoothing(10, 'mm' ); % FWHM value and unit ('mm' or 'vx')
bv.PrintToLog(['Name of spatially smoothed VTC file: ' vmr.FileNameOfCurrentVTC]);
% now we could do a linear trend removal (see code in comments)
% since high-pass temporal filter (see below) includes LTR, we skip this here
% vmr.LinearTrendRemoval(); % FWHM value and unit ('mm' or 'vx')
% bv.PrintToLog(['Name of VTC file without linear trends: ' docVMR.FileNameOfCurrentVTC]);
% now perform temporal high-pass filter
vmr.TemporalHighPassFilter(3, 'cycles'); % HP value and unit ('cycles' or 'Hz')
bv.PrintToLog(['Name of VTC file without linear trends: ' vmr.FileNameOfCurrentVTC]);
% now perform Gaussian temporal smoothing
% FWHM value and unit ('d' or 'dps' (data points) or 's' or 'secs' (seconds))
vmr.TemporalGaussianSmoothing(3, 'dps');
bv.MessageBox(['Name of temporally smoothed VTC file: ' vmr.FileNameOfCurrentVTC]);
disp('Finished preprocessing VTC file.')
```

Chapter 4

Transformations

4.1 Introduction

Several transformations in BrainVoyager can be scripted. For anatomical data, these are the transformation to isotropic voxels and the reorientation to sagittal. For functional data, these are the transformation to normalized functional (*.vtc) and normalized diffusion (*.vdw) file scripting functions; this is discussed in section 4.7.

It is also possible to write a transformation file in Matlab and apply the transformation in BrainVoyager. This will be shown in section 4.8.2.

4.2 Isovoxelation of anatomical data (*.vmr)

Anatomical (VMR) files can be automatically transformed to isovoxel size of $1 \times 1 \times 1$ mm via the function `AutoTransformToIsoVoxel(<interpolation method>, <new vmr name>)`.
The resulting VMR is written to disk. The function returns a boolean value (true or false) to indicate whether the transformation succeeded. The transformation matrix is also displayed in the BrainVoyager Log tab.
Since BrainVoyager 20.4 there is also the function TransformToIsoVoxel(),
with four arguments:
`TransformToIsoVoxel(<interpolation>, <name>, <target_resolution>, <framing_cube_dim>)`
where target resolution in millimeters (for all three dimensions) takes a float value and the framing cube dimension is one of 256, 384, 512 or 768 (will be applied for all three dimensions).

AutoTransformToIsoVoxel()

Function: AutoTransformToIsoVoxel()
Description: Transforms a non-isovoxel VMR file to isovoxel (same voxel sizes in x-dimension, y-dimension and z-dimension, in this case 1x1x1mm). The result is saved to disk with the new name.
Parameter 1: Interpolation method (integer): 1 - trilinear, 2 - cubic spline (recommended), 3 - sinc
Parameter 2: New VMR name (string): name for transformed VMR.
Returns: boolean: success.

TransformToIsoVoxel()

*Function:*TransformToIsoVoxel()
Description: Transforms a non-isovoxel VMR file to an image with same resolution in all dimensions with variable framing cube dimension. The result is saved to disk with the new name.
Parameter 1: Interpolation method (integer): 1 - trilinear, 2 - cubic spline (recommended), 3 - sinc
Parameter 2: New VMR name (string): name for transformed VMR.
Parameter 3: Target resolution: float (for all three dimensions)

Parameter 4: Framing cube dimension: one of the dimensions: 256, 384, 512, 768 (for all three dimensions)
Returns: boolean: success.

4.2.1 Scripts

The function below presents a file dialog to select a VMR file, opens this VMR file in the BrainVoyager main window, transforms the VMR file and saves it on disk in the same directory as the original VMR file with the name “isovoxel.vmr”.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% isovoxel_vmr.m
% Script to make a VMR isovoxel
%
% Works with BrainVoyager 22
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1');
bv.ShowLogTab;
[FileName,PathName] = uigetfile('*.vmr','Please select the anatomical data file...');
vmrfilename = strcat(PathName, FileName);
vmr = bv.OpenDocument(vmrfilename);
bv.PrintToLog('Start isovoxelating VMR...');
[pathstr,name,ext] = fileparts(vmrfilename);
isovmrfilename = fullfile(pathstr, [name '_ISO' ext]);
interpMethod = 2; % 1 - trilinear, 2 - cubic spline (recommended), 3 - sinc
success = vmr.AutoTransformToIsoVoxel(interpMethod, isovmrfilename);
bv.PrintToLog(['Saving ' isovmrfilename '...']);
isoproject = bv.OpenDocument(isovmrfilename); %see figure 4.1
h = msgbox(['The isovoxel file can be found at ' isovmrfilename]);
uiwait(h);
```

Figure 4.1: Isovoxel VMR

4.3 Transform anatomical data (*.vmr) to sagittal orientation

To transform the anatomical image (*.vmr) to BrainVoyager’s default sagittal orientation, the function `AutoTransformToSAG()` can be applied. The result is saved to disk with the new name. Note: this function might not work if the voxel sizes of the VMR are not equal.
The function takes one parameter: the new name (string) for the transformed VMR. The function returns a boolean to indicate whether the transformation succeeded.

4.4 Perform inhomogeneity correction

One needs only two commands, even without parameters, to perform the inhomogeneity correction and transformation to AC-PC and Talairach space; these functions are respectively `CorrectIntensityInhomogeneities()` and `AutoACPCAndTALTransformation()`. The inhomogeneity correction function will save a new VMR to disk with `IIHC` in the name (works only if first `CreateDocumentVMR()` has been invoked or when a *.v16 file with the same name is available in the same folder as the *.vmr). The transformation functions will save new VMRs with `aACPC` and `TAL` in the name, as well as the Talairach landmarks file (*.tal); the function does use SINC interpolation (figure 4.2).

4.5 Transform VMR to AC-PC, Talairach and MNI space

The function NormalizeToMNISpace() performs automatic brain normalization to MNI space using template matching. To work successfully, the calling VMR document should contain brain extracted and inhomogeneity corrected data (see CorrectIntensityInhomogeneities() command above). The boolean return value indicates whether the command could be executed successfully. If successful, a new VMR file with the data in MNI152 space are stored to disk under a name that is based on the input VMR document:"`[input-name]_MNI.vmr''`. The used MNI transformation parameters are stored to disk in the TRF file "`[input-name]_TO_MNI_a12.trf`"; furthermore the files "`[input name]_TO_MNIColin27_a12.trf`" and "`[inputname]_TO_ICBM452_a12.trf`" are stored that can be used to transform the native space VMR also to related spaces, namely to the Colin27 and ICBM 452 MNI space. These TRF files can also be used to apply the MNI transformation to other VMRs in the same native space and as part of the transformation of functional data into MNI space (see CreateVTCInMNISpace() command). The original VMR document is closed and the resulting MNI file is loaded into the BrainVoyager workspace. The loaded MNI file is made the current document and can be accessed using the `ActiveDocument` command of the BrainVoyager application object (section **??**).
Returns: True (success) or false.

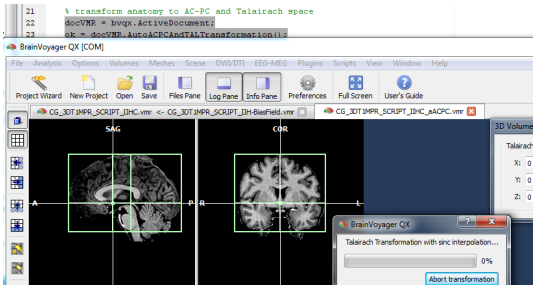


Figure 4.2: Running the"CreateVMRandAutoTAL.m" script

4.5.1 Scripts

```
% CreateVMRTransformToTALandMNI.m
% Created by Rainer Goebel, May 2013
% Adapted for Matlab HB, June 2013, April 2019, June 2021
% Latest version used with BrainVoyager 22.0

% Start BrainVoyager
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1');

% Create the anatomical document (*.vmr)
[FileName, PathName] = uigetfile('*.dcm', 'Please select the first DICOM file (*.dcm)');
dcmname = [PathName FileName];
```

```
[pathstr, name, ext] = fileparts(dcmname);
docVMR = bv.CreateDocumentVMR( 'DICOM', dcmname, 192, false, 256, 256, 2);
docVMR.SaveAs( fullfile(pathstr, 'GSG4_MPRAGE_nondist_script.vmr'));

% Prepare: perform inhomogeneity correction and isovoxelation
docVMR = bv.ActiveDocument;
ok = docVMR.CorrectIntensityInhomogeneities();
docVMR.Close();
iihcSelection = [PathName '*_IIHC.vmr'];
iihcFiles = dir(iihcSelection);
docVMR = bv.OpenDocument(iihcFiles(1).name);
[pathstr, name, ext] = fileparts(iihcFiles(1).name);
newvmrname = fullfile(pathstr, [name '_ISO.vmr']);
ok = docVMR.AutoTransformToIsoVoxel(1, fullfile(pathstr, [name '_ISO.vmr']));
docVMR.Close();

% Transform anatomy to AC-PC, Talairach and MNI space
docVMR = bv.OpenDocument(newvmrname);
ok = docVMR.AutoACPCAndTALTransformation();
ok = docVMR.NormalizeToMNISpace();
```

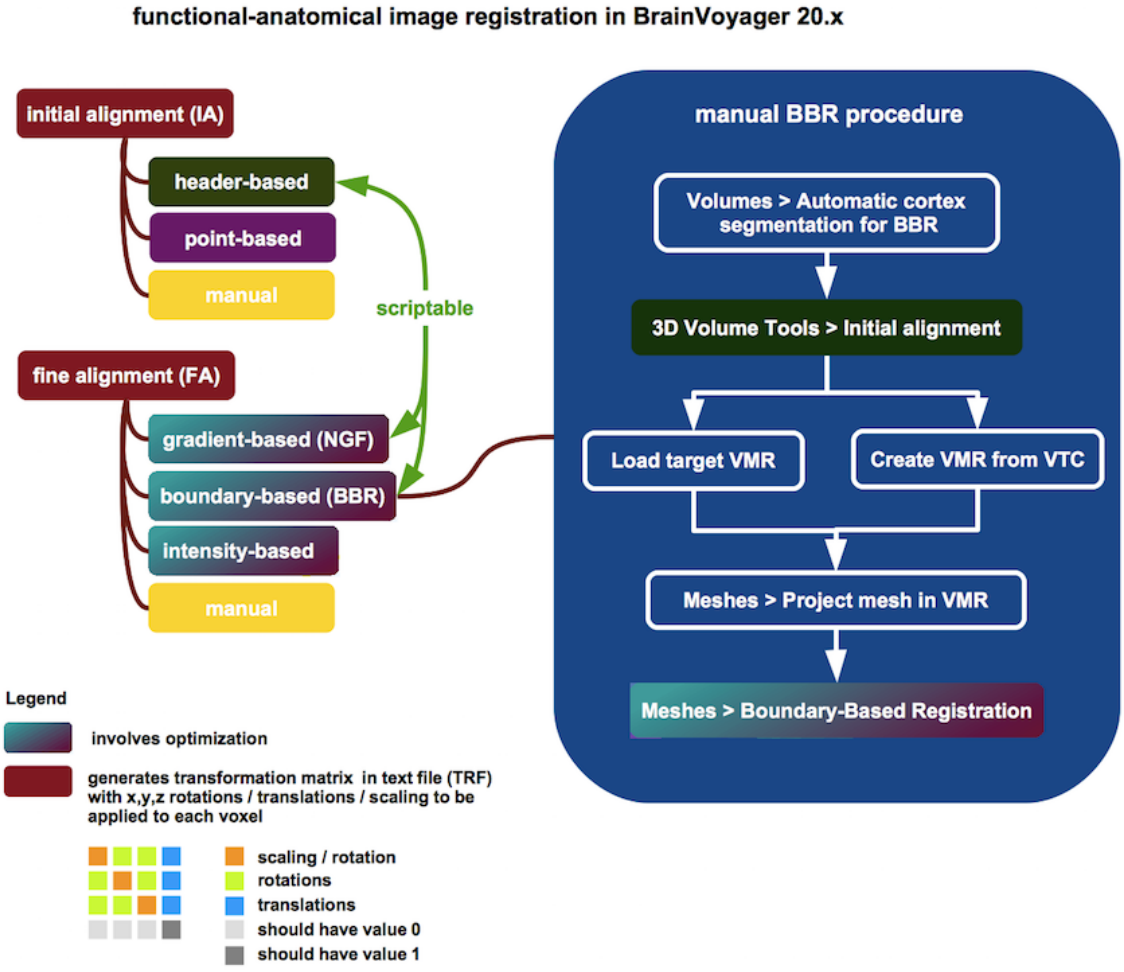
4.5.2 List of functions

Table 4.1: VMR processing and transformations

Description of function	*Function in Qt Script*	*Since*
Apply inhomogeneity (B_1) correction	CorrectIntensityInhomogeneities()	QX 2.8
Obtain voxels with equal resolution in x,y,z	TransformToIsoVoxel()	
Obtain voxels with equal resolution in x,y,z	AutoTransformToIsoVoxel()	
Transform to AC-PC and Talairach space	AutoACPCAndTALTransformation()	
Transform to MNI space	NormalizeToMNISpace()	QX 3.4 (BV20.4)

4.6 Registration of FMR to VMR

Since BrainVoyager 20.4, there are two methods on registering an FMR to VMR: `CoregisterFMRToVMR()` and `CoregisterFMRToVMRUsingBBR()`. These functions produce transformation files (*.trf), required to create normalised functional files (*.vtc).



For CoregisterFMRToVMR() this calculates a robust alignment of the specified FMR with the anatomical data of the calling VMR document object using alignment information from the headers of (DICOM) images as well as an iterative intensity gradientbased matching procedure. The boolean return value indicates whether the command could be executed successfully. The command selects a representative volume of the FMR data (see `useAttachedAMR` parameter), which is aligned to the volume data of the hosting VMR to calculate two transformation matrices, a headerbased initial alignment (IA) TRF file and a matching-based finetuning adjustment (FA) TRF file. The two generated files can be used subsequently to actually transform all volumes of the FMRSTC data in the space of the hosting VMR; the calculation of coregistration files is separated from the actual application since it is usually desired to transform the FMRSTC data into a normalized space beyond the space of the hosting VMR (see VTC creation commands below).
There are two arguments to this function:

- FMRFileName (str): String specifying the name of the input FMR file, which will be coregistered to the hosting VMR. The string should contain the full path file name. The representative functional volume

of the FMR that will be used to calculate the IA and FA coregistration matrices is determined by the useAttachedAMR parameter.

- useAttachedAMR (int): Boolean specifying whether the original FMR volume should be used (value: 0) or whether the volume in an attached AMR should be used as input (value: 1, recommended). If requested but no AMR is available, the first FMR volume will be used instead. The linked AMR is usually only a resampled (higher resolution) version of the first volume of the original functional data (thus also called a "pseudo" AMR). Using the linked (pseduo) AMR is recommended since this volume will not be modified by FMR preprocessing operations such as spatial smoothing (it will also be properly modified in case of acrossrun motion correction). In case that the linked AMR is a true T_1 weighted anatomical coplanar set of slices from an extra scan, a value of 2 should be used for this parameter, which will not invert voxel intensiities, which is otherwise performed to match the functional data volume to the anatomical VMR data volume.

This function returns a boolean.

The function CoregisterFMRToVMRUsingBBR() works as follows: Calculates a robust alignment of the specified FMR with the anatomical data of the calling VMR document object using alignment information from the headers of (DICOM) images as well as an iterative boundarybased registration (BBR) matching procedure. The boolean return value indicates whether the command could be executed successfully. This BBRbased version of the FMRVMR coregistration command takes more calculation time than the "CoregisterFMRToVMR" command (see above) since several intermediate calculations have to be performed, including a whiltegrey matter segmentation, reconstruction and smoothing of a whitegrey matter mesh; if the mesh for the VMR document is already available (from a previous run), the segmentation and mesh processing steps are skipped and only the BBR algorithm is performed. The command uses the first volume of the FMR data, which is aligned to the volume data of the hosting VMR to calculate two transformation matrices, a headerbased initial alignment (IA) TRF file and a BBR based finetuning adjustment (`BBR_FA`) TRF file. The two generated files can be used subsequently to actually transform all volumes of the FMRSTC data in the space of the hosting VMR; the calculation of coregistration files is separated from the actuall application since it is usually desired to transform the FMRSTC data into a normalized space beyond the space of the hosting VMR (see VTC creation commands below).

- FMRFileName (str): String specifying the name of the input FMR file, which will be coregistered to the hosting VMR. The string should contain the full path file name.

This function returns a boolean.
The BBR registration script function produces the following files:

SRF (3x): `ETC-7x-R5_WM_RECO`, `ETC-7x-R5_WM_RECOSM`, `ETC-7x-R5_WM_RECOSM_D300k`

TRF (3x): `BBR_3D3D.trf`, `BBR_FA.trf`, `IA.trf`

VMR (3x): `ETC-7x-R5_WM`, `ETC-7x-R5`, `For-BBR`

VWP (2x): `ETC-7x-R5_WM_RECO`, `ETC-7x-R5_WM_RECOSM`

FLT (1x): `TempCSFDistMap`

4.7 Transform functional data to a standard space (*.vtc)

The functions `CreateVTCInVMRSpace()`, `CreateVTCInACPCSpace()` and `CreateVTCInTALSpace()` will transform the time series in slices (*.stc) to time series in volumes so that the functional data are aligned to the anatomical data in one of the native (VMR), AC-PC aligned (ACPC), Talairach space (TAL) or MNI coordinate spaces (normalisation).
To invoke these functions, BrainVoyager needs to be provided with the same information as it would have when creating VTCs via the user interface. The creation of VTC files can be performed from Matlab in the following steps:

1. Start the BrainVoyager server:
   `bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')`
2. Open an anatomical file (*.vmr)
3. Set bounding box flags. It is advised to set the `ExtendedTALSpaceForVTCCreation` and the `UseBoundingBoxForVTCCreation` properties to `true` or `false`, otherwise VTCs of arbitrary size might be the result. If `UseBoundingBoxForVTCCreation` is set, the coordinates for the VTC w.r.t. the VMR can be set via `TargetVTCBoundingBoxXStart` and `TargetVTCBoundingBoxXEnd` (replace `X` with `Y` or `Z`, when appropriate). The coordinates that are not specified will get the default Talairach bounding box size.
4. Collect the parameters to provide to the `CreateVTC...()` function These are the parameters:
   - Name of FMR document This is the name of the functional data (*.fmr) that should be transformed to a VTC file.
   - Name of initial alignment file This is the name of the initial alignment file (*IA.trf); the matrix in this file represents the largest part of the mapping from the anatomical to the functional data.
   - Name of fine alignment file This is the name of the fine alignment file (*FA.trf); the matrix in this file represents additional small changes for the mapping of the anatomical to the functional data.
   - Name of AC-PC alignment file This is the name of the file that transforms the data from native space to a space where data are aligned to the plane of the anterior commissure (AC) and the posterior commissure (PC). This argument only needs to be provided in case the functions `CreateVTCInACPCSpace()` and `CreateVTCInTALSpace()` are invoked.
   - Name of Talairach landmarks file This file contains the coordinates of the 12 landmarks that are used to transform the data from AC-PC space to Talairach space. This argument only needs to be provided in case the function `CreateVTCInTALSpace()` is invoked.
   - Resolution The value of this parameter defines whether the VTC file will have a resolution of $3mm^3$, $2mm^3$ or $1mm^3$.
   - Interpolation This numeric value indicates the interpolation method. Currently the value should be set to 1, which means 'trilinear interpolation'. In future, more interpolation methods will be available for this function.
   - Bounding box threshold This value is only relevant in case the VTC is not created in Talairach space; however, the value should always be provided. The default is 100. The value represents the intensity threshold on basis of which the size of the VTC box is determined.
5. Set the extended Talairach parameter This is a property of the VMR document, and should also be set in case the VTC will not be in Talairach space.
6. Invoke the `CreateVTC...()` function

CreateVTCInMNISpace() is described as follows: Transforms the specified FMRSTC data into ACPC space producing a VTC file as output. The boolean return value indicates whether the command could be executed successfully. The calling VMR document object should be the same that was used for coregistration of the source FMR file with the calling VMR document object and the ACPC transformation file should

be the one resulting from transforming the hosting VMR into ACPC space. The IA, FA and ACPC transformation files will be combined into one transformation matrix. The resulting resolution will be determined by the spatial resolution (millimeters) of the calling VMR document object but it can be reduced by an integral value using the resolution parameter.
The command for transforming functional data to MNI space has the following arguments:

- FMRFileName (str): String specifying the name of the source FMR file; the voxel time course data in the STC file that is referenced by the FMR file will be used as input of the VTC creation process.
- IACoregTRFFileName (str): String specifying the name of the initial alignment (IA) transformation (TRF) file resulting from the preparatory FMRVMR alignment step. Specification of a full path file name is recommended and required if the TRF file is not located in the same directory as the FMR input file.
- FACoregTRFFileName (str): String specifying the name of the finetuning adjustment (FA) transformation (TRF) file resulting from the preparatory FMRVMR alignment step. Specification of a full path file name is recommended and required if the TRF file is not located in the same directory as the FMR input file.
- MNITRFFileName (str): String specifying the name of the MNI transformation (TRF) file resulting from previously transforming the hosting VMR document into MNI space (e.g. for the default MNI152 space, the file will end with `_TO_MNI_a12.trf` after standard MNI transformation, see "NormalizeToMNISpace" command above).
- VTCFileName (str): String specifying the name of the resulting volume time course (VTC) file in ACPC space. Specification of a full path file name is recommended and required if the resulting file should be stored in another directory than the one of the FMR input file.
- dataType (int): Integer in the range [1, 2] specifying the data type for the values in the resulting VTC output file; value "1" will produce integer values (2 byte), value "2" (recommended) will produce float32 (4 byte float) values.
- resolution (int): Integer in the range [1, 3] specifying the spatial resolution of the voxels of the resulting VTC data relative to the resolution of the calling VMR document object. A value of "1" indicates that the resolution of the VTC data will be the same as the resolution of the hosting VMR, value "2" indicates that 1 VTC voxel will correspond to $2 \times 2 \times 2$ (8) VMR voxels and value "3" indicates that 1 VTC voxel will correspond to $3 \times 3 \times 3$ (27) VMR voxels.
- interpolationMethod (int): Integer in the range [0, 2] specifying the interpolation method used for resampling the STC input data. Providing value "0" will select nearest neighbor interpolation, value "1" trilinear interpolation and value "2" will select sinc interpolation. Sinc interpolation will be very slow if no GPU is available and enabled for sinc interpolation.
- intensityThreshold (int): A value that is used to separate background voxels (low intensity) from brain voxels (high intensity). Note that for MNI transformation this value is not used since the bounding box of the resulting VTC data is set to the standard dimensions of the resulting MNI space.

In the following script is shown how to create multiple VTC files. To use the script, first open the script `create_multiple_vtcs.m` in Matlab (see figure 4.3).

When the script is opened in Matlab, press 'run' to start the script (see figure 4.4).

If the script is not located in the current directory, a question will appear whether to change the current directory. Select the option "Add the directory to the top of the Matlab path" (see figure 4.5).

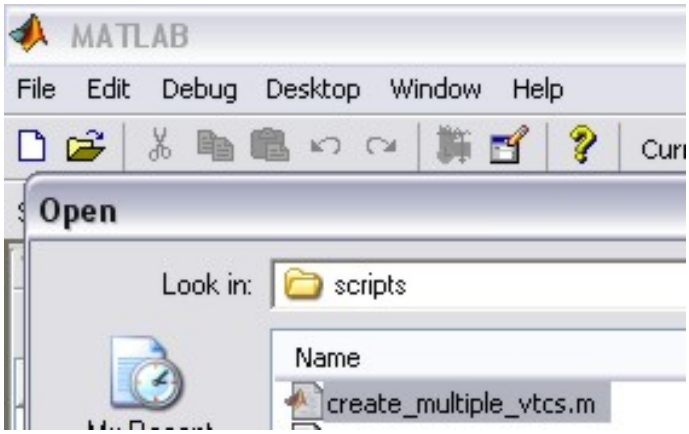


Figure 4.3: Open the script in Matlab

Figure 4.4: Starting the Matlab script by clicking 'run'

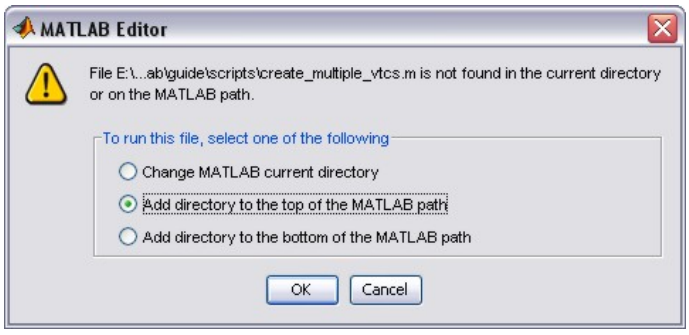


Figure 4.5: Changing the current directory by adding the path of the script

4.7.1 Script to create multiple VTC files

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create_multiple_vtcs.m
% Script to create volume time course files (*.vtc)
% For use: load in Matlab and press 'run'.
%
% Works with BrainVoyager 22
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h = msgbox(['Via this script will be multiple VTCs created. Please first define all parameters...'])
uiwait(h);
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1');
bv.ShowLogTab;
bv.PrintToLog('Creating VTC files from Matlab...');
loadparams = questdlg('Would you like to load VTC parameters (from earlier use of this script)?',...
    'Create multiple VTCs');
if strcmp(loadparams, 'Yes') == 1
    [FileName,PathName] = uigetfile('*.mat', 'Please select the VTC parameters file ');
    load([PathName FileName]);
else
    answer = inputdlg('Please enter the number of projects...','Create multiple VTCs',1);
    if isnumeric(str2num(answer{1})) && str2num(answer{1}) > 0
        nrofprojects = str2num(answer{1});
    end
    % create cell arrays to store the vtc parameters
    fmr_names = cell(nrofprojects, 1);
    ia_names = cell(nrofprojects, 1);
    fa_names = cell(nrofprojects, 1);
    acpc_names = cell(nrofprojects, 1);
    tal_names = cell(nrofprojects, 1);
    mni_names = cell(nrofprojects, 1);
    vtc_names = cell(nrofprojects, 1);
    vtcspaces = {'native', 'acpc', 'talairach', 'mni'};
    [space,v] = listdlg('PromptString','select the target space:',...
        'SelectionMode','single',...
        'Name', 'Create multiple VTCs',...
        'ListSize', [200 160],...
        'ListString', vtcspaces)
    for i=1:nrofprojects
        [FileName,PathName] = uigetfile('*.fmr', ...
            ['Please select FMR file ' num2str(i) ' of ' num2str(nrofprojects) '...']);
        fmr_names{i,1} = strcat(PathName, FileName);
        [FileName,PathName] = uigetfile('*_IA.trf', ...
            ['Please select initial alignment file ' num2str(i) ' of ' num2str(nrofprojects) '...']);
        ia_names{i,1} = strcat(PathName, FileName);
        [FileName,PathName] = uigetfile('*_FA.trf', ...
            ['Please select fine alignment file ' num2str(i) ' of ' num2str(nrofprojects) '...']);
        fa_names{i,1} = strcat(PathName, FileName);
        if ((space == 2) || (space == 3))
            [FileName,PathName] = uigetfile('*_ACPC.trf', ...
                ['Please select AC-PC alignment file ' num2str(i) ' of ' num2str(nrofprojects) '...']);
            acpc_names{i,1} = strcat(PathName, FileName);
            if (space == 3)
                [FileName,PathName] = uigetfile('*.tal', ...
                    ['Please select Talairach landmarks file ' num2str(i) ' of ' num2str(nrofprojects) '...']);
                tal_names{i,1} = strcat(PathName, FileName);
            end
        elseif (space == 4)
                [FileName,PathName] = uigetfile('*_a12.trf', ...
                    ['Please select MNI transformation file ' num2str(i) ' of ' num2str(nrofprojects) '...']);
                mni_names{i,1} = strcat(PathName, FileName);
        end

        [pathstr,name,ext] = fileparts(fmr_names{i,1});
        vtc_names{i,1} = fullfile(pathstr, [name '_' vtcspaces{space} '.vtc']);
    end
    resolution = 3;
    answer = inputdlg('Please enter the voxel resolution...','Create multiple VTCs',1,{num2str(resolution)});
    if isnumeric(str2num(answer{1})) && str2num(answer{1}) > 0
        resolution = str2num(answer{1});
    end
    interpolation = 1; % trilinear
    bbithresh = 100;
    cerebellum = 0;
    if space < 3
        answer = inputdlg('Please enter an intensity threshold...','Create multiple VTCs',1,{num2str(bbithresh)});
        if isnumeric(str2num(answer{1})) && str2num(answer{1}) > 0
            bbithreshold = str2num(answer{1});
        end
    else
        button = questdlg('Would you like an extended Talairach space (incl. cerebellum)?','Create multiple VTCs');
```

```
        if strcmp(button, 'Yes') == 1, cerebellum = 1; end
    end
    button = questdlg('Would you like high-precision data (float format)?','float or integer format');
    if strcmp(button, 'Yes') == 1
        datatype = 2; % float 32
    else
        datatype = 1; % integer 16-bit
    end
    [FileName,PathName] = uigetfile('*.vmr','Please select the anatomical data file...');
    vmrfilename = [PathName FileName];
end
vmr = bv.OpenDocument(vmrfilename);
vmr.ExtendedTALSpaceForVTCCreation = cerebellum;
for i=1:nrofprojects
    if space == 1
        success = vmr.CreateVTCInVMRSpace(fmr_names{i,1}, ia_names{i,1}, fa_names{i,1},...
            vtc_names{i,1}, datatype, resolution, interpolation, bbithresh);
    elseif space == 2
        success = vmr.CreateVTCInACPCSpace(fmr_names{i,1}, ia_names{i,1}, fa_names{i,1},...
            acpc_names{i,1}, vtc_names{i,1}, datatype, ...
            resolution, interpolation, bbithresh);
    elseif space == 3
        success = vmr.CreateVTCInTALSpace(fmr_names{i,1}, ia_names{i,1}, fa_names{i,1},...
            acpc_names{i,1}, tal_names{i,1}, vtc_names{i,1}, datatype, ...
            resolution, interpolation, bbithresh);
    elseif space == 4
        success = vmr.CreateVTCInMNISpace(fmr_names{i,1}, ia_names{i,1}, fa_names{i,1},...
            mni_names{i,1}, vtc_names{i,1}, datatype, ...
            resolution, interpolation, bbithresh);
    else
        disp('An error occurred. Our apologies');
    end
    if (success == 1)
        bv.PrintToLog(['Created ' vtc_names{i,1}]);
    else
        disp(['An error occurred while creating the VTC file' vtc_names{i,1}]);
    end
end
button = questdlg('Would you like to save the VTC parameters (for later use of this script)?','Create multiple VTCs');
if strcmp(button, 'Yes') == 1
    vtcparamsfilename = fullfile(pathstr, 'vtcparams.mat')
    save(vtcparamsfilename); % save workspace variables to a file
end
```

Dialogs of the script to create multiple VTC files

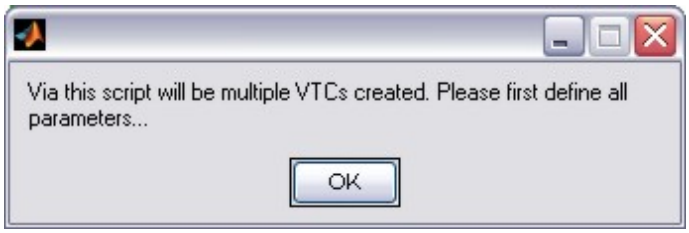


Figure 4.6: Messagebox announcing the purpose of the function

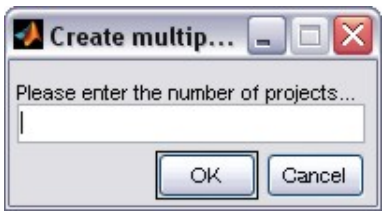


Figure 4.7: Dialog (inputdlg) for entering the number of projects

Figure 4.8: Dialog (listdlg) for selecting the VTC space

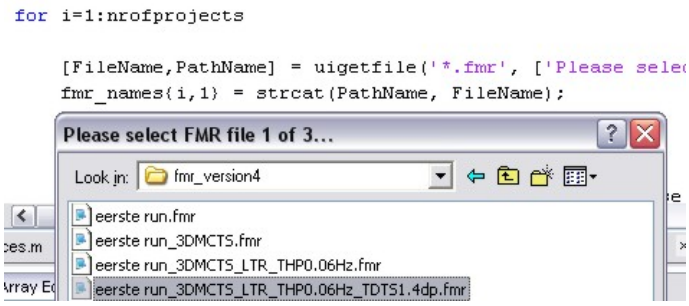


Figure 4.9: Selecting the FMR file (uigetfile)

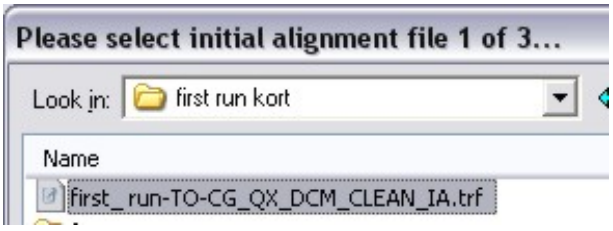


Figure 4.10: Selecting the initial alignment transformation file (uigetfile)

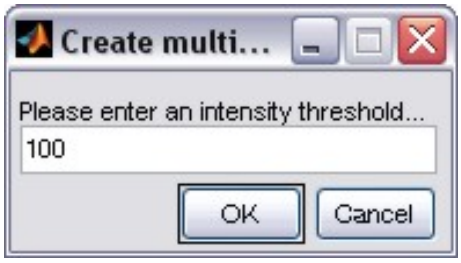


Figure 4.11: Entering the bounding box intensity threshold (inputdlg)

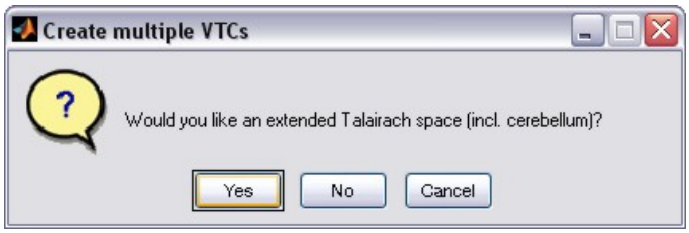


Figure 4.12: Question (questdlg) about the extended box for Talairach space

4.7.2 List of functions

Table 4.2: VTC files

Description of function	*Function in Qt Script*	*Since*
Create VTC in VMR space	CreateVTCInVMRSpace(arguments): boolean	QX 1.3
Create VTC in AC-PC space	CreateVTCInACPCSpace(arguments): boolean	QX 1.3
Create VTC in Talairach space	CreateVTCInTALSPace(arguments): boolean	QX 1.3
Create VTC in MNI space	CreateVTCInMNISpace(arguments): boolean	QX 3.4 (BV20.4)
Link VTC file to VMR	LinkVTC('name')	
Save VTC including link to protocol	SaveVTC()	QX 2.4.1
Set TAL bounding box size	ExtendedTALSpaceForVTCCreation (property)	
Get/set bounding box size w.r.t. VMR	UseBoundingBoxForVTCCreation (property)	QX 2.4.1
Get/set first x-coordinate of VTC w.r.t. VMR	TargetVTCBoundingBoxXStart (property)	QX 2.4.1
Get/set last x-coordinate of VTC w.r.t. VMR	TargetVTCBoundingBoxXEnd (property)	QX 2.4.1
Get/set first y-coordinate of VTC w.r.t. VMR	TargetVTCBoundingBoxYStart (property)	QX 2.4.1
Get/set last y-coordinate of VTC w.r.t. VMR	TargetVTCBoundingBoxYEnd (property)	QX 2.4.1
Get/set first z-coordinate of VTC w.r.t. VMR	TargetVTCBoundingBoxZStart (property)	QX 2.4.1
Get/set last z-coordinate of VTC w.r.t. VMR	TargetVTCBoundingBoxZEnd (property)	QX 2.4.1

4.8 Transform diffusion weighted data to a standard space (*.vdw)

The BrainVoyager script functions to create normalized diffusion weighted data (*.vdw) are quite similar to the functions to create normalized functional MRI data (*.vtc). Therefore, in case the bounding box sizes are not consistent, one could set the property `ExtendedTALSpaceForVTCCreation` to true or false like with VTC files; see also the information on the `UseBoundingBoxForVTCCreation` property (since BrainVoyager QX 2.4.1). The commands are listed in table 4.8.1.
The arguments are:
Parameter: Name DMR: Name of the diffusion-weighted data file (*.dmr) which should be transformed to Talairach space.
Parameter: Name IA file: Name of the initial alignment transformation file (`*_IA.trf`).
Parameter: Name FA file: Name of the fine alignment transformation file (`*_FA.trf`).
Parameter (optional): Name ACPC file: Name of the transformation file of the VMR to AC-PC space (`*_ACPC.trf`).
Parameter (optional): Name TAL file: Name of the file containing 12 landmarks used to transform the VMR to Talairach space (`*.tal`).
Parameter: Name VDW: Name for the new VDW file.
Parameter: Datatype: Create the VDW in integer 2-byte format: 1 or in float format: 2.
Parameter: Resolution: Target resolution, either '1' (1x1x1mm), '2' or '3'.
Parameter: Interpolation: '0' for nearest neighbor interpolation, '1' for trilinear interpolation, '2' for sinc interpolation.
Parameter: Threshold: intensity threshold for bounding box (is not relevant for Talairach space, but should be provided). Default value: '100'.
Returns: True (success) or false.

For an example script, please see section 2.5.3.

4.8.1 List of functions

Normalised diffusion weighted data files can be created using the following functions.

Table 4.3: VDW files

Description of function	*Function in Qt Script*	*Since*
Create VDW in VMR space	CreateVDWInVMRSpace(arguments):boolean	1.x
Create VDW in AC-PC space	CreateVDWInACPCSpace(arguments):boolean	1.x
Create VDW in Talairach space	CreateVDWInTALSpace(arguments):boolean	1.x

4.8.2 More on BrainVoyager transformation (*.trf) files

A coordinate transformation assigns new coordinates to an object in a certain space [1]. This is necessary when the object is moved (translated), when the object changes size (scaling) or when the object is rotated. This can be performed with a function that transforms the x, y and z coordinates of each voxel. The coefficients to compute the new x, y and z coordinates from the current coordinates can be expressed in a 4×4 matrix. These coefficients express how much the object is translated, rotated or scaled. When the coefficient quantity is equal to 1 or -1, the complete shape of the object is preserved, only its orientation can be different. A scaling function, however, changes the object, for example a matrix to perform isovoxelation:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

The value 1 on the diagonal of the matrix, position (0,0)[1], indicates that the position of the object on the x-axis is completely preserved in the transformation. This is also the case for the position of the voxels on the y-axis on position (1,1) of the matrix. For the z-axis however, the 2 indicates that the stepsize changes; it is computed by `new voxel resolution/current voxel resolution}`, in this case `1.0/0.5` (see figure 4.13).

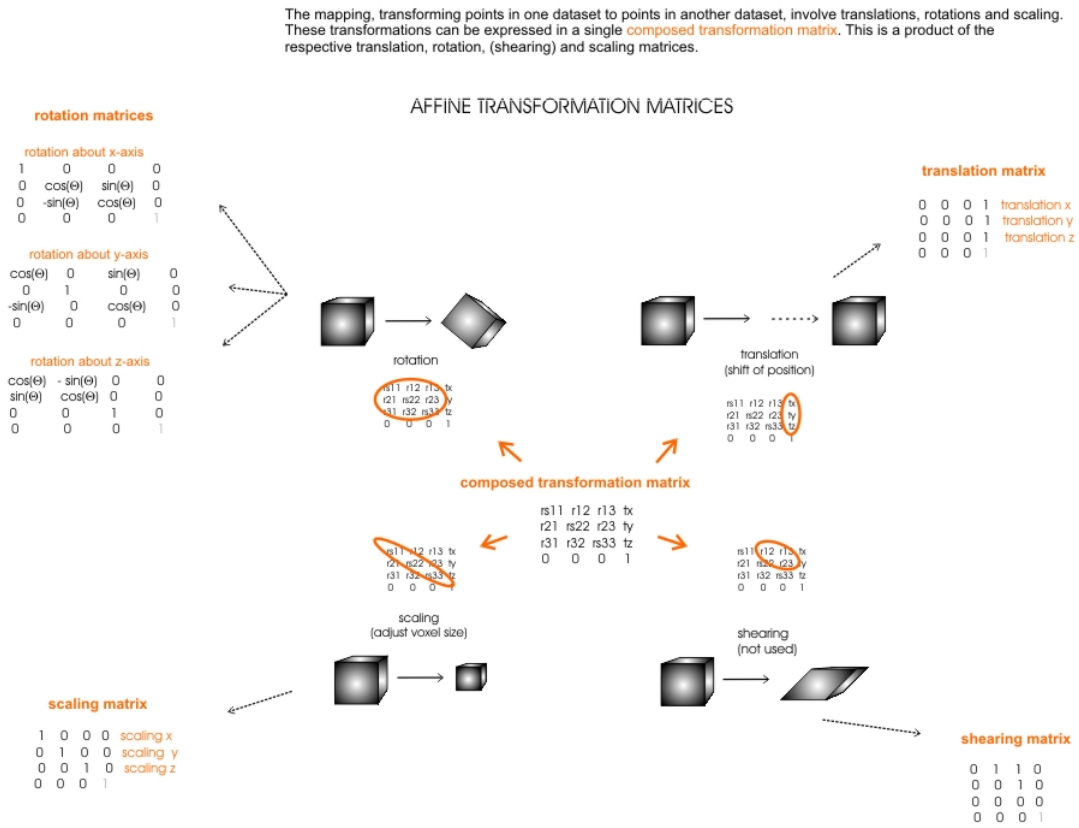


Figure 4.13: Components of a composed transformation matrix

[1] where matrix indices denoted by (row, column)

In BrainVoyager, 4×4 transformation matrices are saved in transformation files (*.trf). Via the following Matlab function can a transformation file be generated, that can be used in the creation of VTCs or to transform a anatomical (*.vmr) or statistical volume (*.vmp) via the 3D Volume Tools dialog (see figure 4.14).

Figure 4.14: Load a transformation file via the 3D Volume Tools in BrainVoyager

For example, if we would like to create a transformation file for the matrix that flips a volume, we could do the following.

First, create the 4×4 matrix via the line

```
matrix = [1 0 0 0; 0 1 0 0; 0 0 2 0; 0 0 0 1];
```

Then, enter the name of the file that should be transformed, for example

```
sourcefile = 'C:\Data\Experiment\project.vmr';
```

Split the name of the sourcefile into parts to create a new filename via

```
[pathstr,name,ext] = fileparts(sourcefile);
```

The new name can be created via the Matlab `fullfile()` command

```
transformedvmrname = fullfile(pathstr, [name '_TRF' ext]);
```

Finally, invoke the `writeTrf()` function that is provided below. Be sure that the Matlab current directory is the same as the directory where the Matlab file (*.m) containing the `writeTrf()` function.

```
writeTrf(2, matrix, 0, sourcefile, transformedvmrname)
```

4.8.3 Function to write a transformation file

```
function writeTrf(trftype, matrix, coordsys, sourcefile, targetfile, varargin)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% writeTrf.m
% Inputparameters:
% 1.   trftype: 1= rigidbody; 2=affine (f.e. tosag and isovoxel); 4=tal; 5=untal
% 2.   4 x 4 matrix (Matlab format: column oriented),...
%  1 x 16 vector [col1 col2 col3 col4] or mat filename (SPM)
% 3.   coordsys: 0 is BV internal; 1 is BV sys; 2 is Talairach.
% 4.   sourcefile; in case of transformation type 2, optional.
% 5.   targetfile (trftype=1)
% 6.  slices; number of slices of functional data(trftype=1)
% 7.  functhick: slice thickness of functional images (trftype=1)
% 8.  gap; slice gap of functional data(trftype=1)
% 9.  fmr3d; 2 or 3 (trftype=1)
% 10.  alignstep; 1: initial alignment; 2: fine alignment (trftype=1)
% 11.  matxtra: 4 x 4 matrix specifying a transformation of the anatomical data (trftype=1)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\newline
    if trftype==1
    if nargin < 10, disp('Need at least parameters for rigid body transformation file.'); return; end
    if nargin < 11
        xtratrf = 0;
    else
        matxtra = varargin(5)
        xtratrf = 1;
    end
        slices = varargin(1)
        gap = varargin(2)
        fmr3d = varargin(3)
        alignstep = varargin(4)
elseif trftype==2
    if nargin < 2, disp('Need at least 2 parameters for affine transformation file.'); return; end
    if nargin < 5, targetfile = fullfile(pwd,'somefile_TRF.vmr'); end
    if nargin < 4, sourcefile = fullfile(pwd,'somefile.vmr'); end
    if nargin < 3, coordsys = 2; end % assume TAL or MNI space
    xtratrf = 0;
end
fileversion = 5;
[srcPath,sourcePrt,srcExt,srcVersn] = fileparts(sourcefile);

    if isnumeric(matrix)
    [rows, cols] = size(matrix);
    if rows==4 && cols==4
        mat1 = reshape(matrix, 1,[]);
    elseif rows*cols~=16
        disp('Wrong number of parameters in matrix');
        return;
    else
        mat1 = matrix;
    end
elseif exist(matrix)==2
    s = load(matrix,'M');
    mat1 = reshape(s.M, 1,[]); %s.M;
else %if exist(matfile)~=2
    disp('Unrecognized input for matrix parameter');
    return;
end

    if trftype ==1
    [trgPath,targetPrt,trgExt,trgVersn] = fileparts(targetfile);
    trfName = fullfile(srcPath, [targetPrt 'TO' sourcePrt '.trf']);
else %if trftype ==2
    trfName = fullfile(srcPath, [sourcePrt '.trf']);
end

    trfFid = fopen(trfName,'wt');
fprintf(trfFid, '\n%s\t%s\n', 'FileVersion:', num2str(5));
fprintf(trfFid, '\n%s\t%s\n\n', 'DataFormat:', 'Matrix');
fprintf(trfFid, '%3.16f\t%3.16f\t%3.16f\t%3.16f\n',...
 mat1(1,1), mat1(1,5), mat1(1,9), mat1(1,13));
fprintf(trfFid, '%3.16f\t%3.16f\t%3.16f\t%3.16f\n',...
 mat1(1,2), mat1(1,6), mat1(1,10), mat1(1,14));
fprintf(trfFid, '%3.16f\t%3.16f\t%3.16f\t%3.16f\n',...
 mat1(1,3), mat1(1,7), mat1(1,11), mat1(1,15));
fprintf(trfFid, '%3.16f\t%3.16f\t%3.16f\t%3.16f\n\n',...
 mat1(1,4), mat1(1,8), mat1(1,12), mat1(1,16));
fprintf(trfFid, '%s\t%s\n', 'TransformationType:', num2str(trftype));
fprintf(trfFid, '%s\t%s\n\n', 'CoordinateSystem:', num2str(coordsys));
if trftype==1
```

```
    fprintf(trfFid, '%s\t%s\n', 'NSlicesFMRVMR:', num2str(slices));
    fprintf(trfFid, '%s\t%s\n', 'SlThickFMRVMR:', num2str(functhick));
    fprintf(trfFid, '%s\t%s\n', 'SlGapFMRVMR:', num2str(gap));
    fprintf(trfFid, '%s\t%s\n', 'CreateFMR3DMethod:', num2str(fmr3d));
    fprintf(trfFid, '%s\t%s\n\n', 'AlignmentStep:', num2str(alignstep));
    fprintf(trfFid, '%s\t%s\n\n', 'ExtraVMRTransf:', num2str(xtratrf));
    if xtratrf
        fprintf(trfFid, '%3.16f\t%3.16f\t%3.16f\t%3.16f\n',...
 matxtra(1,1), matxtra(1,5), matxtra(1,9), matxtra(1,13));
        fprintf(trfFid, '%3.16f\t%3.16f\t%3.16f\t%3.16f\n',...
matxtra(1,2), matxtra(1,6), matxtra(1,10), matxtra(1,14));
        fprintf(trfFid, '%3.16f\t%3.16f\t%3.16f\t%3.16f\n',...
matxtra(1,3), matxtra(1,7), matxtra(1,11), matxtra(1,15));
        fprintf(trfFid, '%3.16f\t%3.16f\t%3.16f\t%3.16f\n\n',...
 matxtra(1,4), matxtra(1,8), matxtra(1,12), matxtra(1,16));
    end
end
fprintf(trfFid, '%s\t%s%s%s\n', 'SourceFile:', '"', sourcefile, '"');
fprintf(trfFid, '%s\t%s%s%s\n', 'TargetFile:', '"', targetfile, '"');
fclose(trfFid);

   return;
```

Chapter 5

Experimental design

In this chapter is described how to create stimulation protocols (*.prt) and design matrices (*.sdm, *.mdm).

5.1 Creating stimulation protocols (*.prt)

The colors for each condition can be set via the function
`fmr.SetConditionColor('conditionname', red, green, blue);`
To distinguish between the different conditions, different colors can be used. The colors in BrainVoyager are specified as a combination of red, green and blue components, in this order. To lowest value for each component is 0 and the highest value is 255. When one of the components is set to 255 and the other two to 0, a primary color is obtained. When each of the components red, green and blue is set to 0, the result will be black in absence of all colors. A list with red, green and blue component values for the most common colors can be found in figure 5.1.

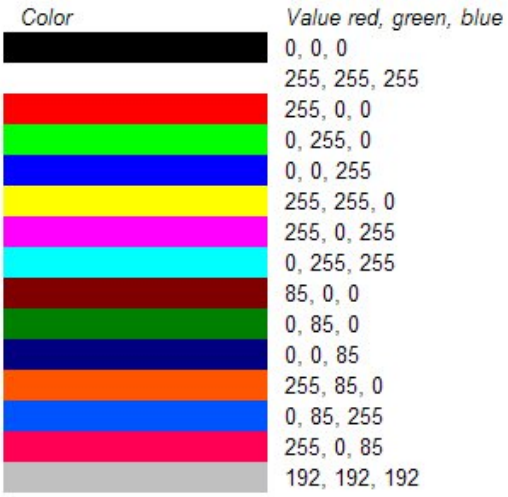


Figure 5.1: Example protocol RGB colors

Table 5.1: PRT files

Description of function	*Function in BrainVoyager API*	*Since version*
Create new protocol	ClearStimulationProtocol()	BV QX 1.3
Create new protocol	ClearProtocol()	BV 23.0
Link protocol	LinkStimulationProtocol()	
Link protocol	LinkProtocol()	BV 23.0
Add condition	AddCondition()	BV QX 1.3
Stimulation protocol name without path	StimulationProtocolFile (property)	BV QX 1.3
Get/Set parametric property	ProtocolParametric	BV 23.0
Experiment name	StimulationProtocolExperimentName (property)	
Experiment name	ProtocolExperimentName (property)	BV 23.0
Unit: 1=volumes 2=milliseconds	StimulationProtocolResolution (property)	
Background color of plot	StimulationProtocolBackgroundColorR/G/B (properties)	
Background color of plot	ProtocolBackgroundColorR/G/B (properties)	BV 23.0
Timecourse color in plot	StimulationProtocolTimeCourseColorR/G/B (properties)	
Timecourse color in plot	ProtocolTimeCourseColorR/G/B (properties)	
Text color in plot	StimulationProtocolTextColorR/G/B (properties)	
Text color in plot	ProtocolTextColorR/G/B (properties)	BV 23.0
Thickness of timecourse line	StimulationProtocolTimeCourseThickness (property)	
Thickness of timecourse line	ProtocolTimeCourseThickness (property)	23.0
Set color of condition	SetConditionColor()	
Add interval	AddInterval()	1.3
Save stimulation protocol	SaveStimulationProtocol()	BV QX 1.3
Save stimulation protocol	SaveProtocol()	BV 23.0

5.1.1 Detailed description of methods

ClearStimulationProtocol()

Function: ClearStimulationProtocol() or ClearProtocol()
Description: Function to start a new stimulation protocol. Use `ClearProtocol` in BrainVoyager 23.0 and higher.

LinkStimulationProtocol() or LinkProtocol()

Function: LinkStimulationProtocol() or LinkProtocol()
Description: Link the stimulation protocol with the provided name to the currently opened VMR file. Use `LinkProtocol` in BrainVoyager 23.0 and higher.
Parameter 1: Name protocol: name of the stimulation protocol (*.prt).

AddCondition()

Function: AddCondition() *Description:* Add a condition for the current stimulation protocol.
Parameter 1: Name for the condition (string).

SetConditionColor()

Function: SetConditionColor()
Description: To discriminate the different conditions, different colors can be used. The colors in BrainVoyager are specified as a combination of red, green and blue components, in this order. To lowest value for each component is 0 and the highest value is 255. When one of the components is set to 255 and the other two to 0, a primary color is obtained. When each of the components red, green and blue is set to 0, the result will be black in absence of all colors. Example colors are displayed in figure **??**.
Parameter 1: Name of condition (string)
Parameter 2: Red color component (0-255)
Parameter 3: Green color component (0-255)
Parameter 4: Blue color component (0-255)

AddInterval()

Function: AddInterval()
Description: Add an interval for a condition.
Example: `fmr.AddInterval("Images in RVF", 1, 2);`.
Parameter 1: Name of condition (string)
Parameter 2: Start of interval in milliseconds or volumes
Parameter 3: End of interval in milliseconds or volumes

SaveStimulationProtocol()

Function: SaveStimulationProtocol() or SaveProtocol()
Description: Save the newly created stimulation protocol with the provided name. Use `SaveProtocol()` in BrainVoyager 23.0 and higher
Parameter 1: Name for the protocol file.

SaveVTC()

Function: SaveVTC()
Description: Save the VTC, which will also save the name of the stimulation protocol in a VTC. New in BrainVoyager QX 2.4.1.
Parameter 1: Name for current VTC. When using an empty string (""), the current VTC file name will be used.

5.2 Scripts

5.2.1 Script to create stimulation protocol for BrainVoyager example dataset

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create_cgobjects_protocol.m; script to create stimulation protocol in
% Matlab
% Works with BrainVoyager 22.0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')
bv.ShowLogTab;
bv.PrintToLog('Creating a BrainVoyager stimulation protocol from Matlab...');
[FileName,PathName] = uigetfile('*.vmr', 'Please select the anatomical file ');
doc = bv.OpenDocument([PathName FileName]);
[FileName,PathName] = uigetfile('*.vtc', 'Please select the functional file ');
doc.LinkVTC(fullfile(PathName, FileName));
doc.ClearStimulationProtocol();
doc.StimulationProtocolExperimentName = 'Faces Houses in LVF, CVF, RVF';
doc.StimulationProtocolResolution = 1;
doc.AddCondition('Faces_LVF');
doc.AddCondition('Houses_RVF');
doc.AddCondition('Faces_CVF');
doc.AddCondition('Houses_LVF');
doc.AddCondition('Faces_RVF');
doc.AddCondition('Houses_CVF');
doc.AddInterval('Faces_LVF', 4, 11);
doc.AddInterval('Houses_RVF', 20, 27);
doc.AddInterval('Faces_CVF', 36, 43);
doc.AddInterval('Houses_LVF', 52, 59);
doc.AddInterval('Faces_RVF', 68, 75);
doc.AddInterval('Houses_CVF', 84, 91);
doc.AddInterval('Faces_LVF', 100, 107);
doc.AddInterval('Houses_RVF', 116, 123);
doc.AddInterval('Faces_CVF', 132, 139);
doc.AddInterval('Houses_LVF', 148, 155);
doc.AddInterval('Faces_RVF', 164, 171);
doc.AddInterval('Houses_CVF', 180, 187);
doc.AddInterval('Faces_LVF', 196, 203);
doc.AddInterval('Houses_RVF', 212, 219);
doc.AddInterval('Faces_CVF', 228, 235);
doc.AddInterval('Houses_LVF', 244, 251);
doc.AddInterval('Faces_RVF', 260, 267);
doc.AddInterval('Houses_CVF', 276, 283);
doc.SetConditionColor('Faces_LVF', 200, 43, 43);
doc.SetConditionColor('Houses_RVF', 200, 200, 43);
doc.SetConditionColor('Faces_CVF', 43, 200, 43);
doc.SetConditionColor('Houses_LVF', 43, 200, 200);
doc.SetConditionColor('Faces_RVF', 43, 43, 200);
doc.SetConditionColor('Houses_CVF', 200, 43, 200);
doc.StimulationProtocolBackgroundColorR = 0;
doc.StimulationProtocolBackgroundColorG = 0;
doc.StimulationProtocolBackgroundColorB = 0;
doc.StimulationProtocolTimeCourseColorR = 255;
doc.StimulationProtocolTimeCourseColorG = 255;
doc.StimulationProtocolTimeCourseColorB = 255;
doc.StimulationProtocolTimeCourseThickness = 2;
prtname = fullfile(PathName, 's01_sess4_blocked_vol_script.prt');
doc.SaveStimulationProtocol(prtname);
bv.PrintToLog(['Created protocol: ' prtname]);
doc.Save(); % to save link to protocol permanently
```

5.3 Creating design matrices (*.sdm, *.mdm)

Creating a single-subject design matrix (*.sdm) consists of the following steps. First, a document needs to be opened via the `OpenDocument(name)` method. This returns a BrainVoyager AMR, FMR, VMR or DMR document object. All following functions belong to this document. Then, a VTC file and a stimulation protocol are linked via the functions `LinkVTC(name)` and `LinkStimulationProtocol(name)`. Save the name of the stimulation protocol file in the VTC file via the `SaveVTC` function. Make space for the new matrix via the function `ClearDesignMatrix;`. Then, the predictors can be added via `AddPredictor('name')`. Set the predictor values via `SetPredictorValuesFromCondition`[1].
Then, apply the delay of the HRF in the model via
`ApplyHemodynamicResponseFunctionToPredictor();`[2]. Finally, the design matrix (*.sdm) can be saved with the function
`SaveSingleStudyGLMDesignMatrix('name.sdm');`. For an example script, please see below.

5.3.1 List of functions

Table 5.2: Design matrices (SDM and MDM files)

Description of function	*Function in BrainVoyager API*	*Since version*
Create new design matrix	ClearDesignMatrix()	BV QX 1.3
Add predictor	AddPredictor()	
	AutoAddConstantPredictor()	BV QX 1.3-1.9 (obsolete)
Set predictor values	SetPredictorValues()	
Set predictor values from a condition in existing protocol	SetPredictorValuesFromCondition('predictor', 'protocolcondition', value)	
Scale predictor values for a condition	ScalePredictorValues()	BV QX 1.3
Apply (Boynton) HRF to condition	ApplyHemodynamicResponseFunctionToPredictor()	BV QX 1.3
Get predictor values		
Get/set first confound predictor column	FirstConfoundPredictorOfSDM (property)	BV QX 2.1
Get/set whether constant predictor is available (boolean)	SDMContainsConstantPredictor (property)	BV QX 2.1
Load design matrix	LoadSingleStudyGLMDesignMatrix()	BV QX 1.3
Save design matrix	SaveSingleStudyGLMDesignMatrix()	BV QX 1.3
Create new multi-study design matrix	ClearMultiStudyGLMDefinition()	BV QX 1.3
Add FMR/VTC and SDM to multi-study design matrix	AddStudyAndDesignMatrix()	BV QX 1.3
Load MDM	LoadMultiStudyGLMDefinitionFile()	BV QX 1.3
Save MDM	SaveMultiStudyGLMDefinitionFile()	BV QX 1.3
Get list of VTC files	GetVTCsOfMDM()	BV 23.0

[1] arguments: 'name', 'name in protocol' and 'value' in decimals. The default value is 1.0.
[2] argument: name (string)

5.3.2 Detailed description of functions

ClearDesignMatrix()

Function: ClearDesignMatrix()
Description: Removes any current design matrices.

AddPredictor()

Function: AddPredictor()
Description: Add new regressor to design matrix.
Parameter 1: Name of regressor (string)

SetPredictorValues()

Function: SetPredictorValues()
Description: Specify the predictor value from a certain time point to the specified end time point in a specific condition.
Parameter 1: Name of condition (string)
Parameter 2: Timepoint from (integer)
Parameter 3: Timepoint to (integer)
Parameter 4: Value for the predictor

SetPredictorValuesFromCondition()

Function: SetPredictorValuesFromCondition()
Description: Set the predictor values for the provided condition using the information from the stimulation protocol.
Parameter 1: Name of predictor in design matrix (string)
Parameter 2: Name of condition in stimulation protocol (string)
Parameter 3: Maximum value for predictor. Default: '1.0'

ApplyHemodynamicResponseFunctionToPredictor()

Function: ApplyHemodynamicResponseFunctionToPredictor()
Description: Apply the hemodynamic response function (HRF) to the provided predictor (Boynton). To use the 2-gamma HRF, BrainVoyager plugins can be used (see the 'Design Matrix Access Functions' topic in the 'Plugins' chapter of the BrainVoyager User's Guide).
Parameter 1: Name of the condition that should be convolved with the HRF.

ScalePredictorValues()

Function: ScalePredictorValues()
Description: Scale the values of the provided predictor.
Parameter 1: Name of the condition that should be scaled.
Parameter 2: Maximum value for scale, for example 1.0.
Parameter 3: Boolean (true or false).

SaveSingleStudyGLMDesignMatrix()

Function: SaveSingleStudyGLMDesignMatrix()
Description: Save the single study design matrix as *.rtc or *.sdm file. *Parameter 1:* Name for the design matrix file.

ClearMultiStudyGLMDefinition()

Function: ClearMultiStudyGLMDefinition()
Description: Remove any present *.mdm file.

AddStudyAndDesignMatrix()

Function: AddStudyAndDesignMatrix()
Description: Add a combination of functional data (*.vtc) and design matrix (*.sdm).
Parameter 1: Name of the functional run (*.vtc)
Parameter 2: Name of the design matrix (*.sdm)

SaveMultiStudyGLMDefinitionFile()

Function: SaveMultiStudyGLMDefinitionFile()
Description: Save the newly created multi-study design matrix (*.mdm).
Parameter 1: Name for the multi-study design matrix file.

5.3.3 Script to create a single-subject design matrix

```
%------------------------------------------------------------------------
% bvprotocol_to_bvdesignmatrix.m
%
% Script to create a design matrix from a stimulation protocol in Matlab
% Usage: load script in Matlab, click 'Run'. File dialogs will appear to
% ask for the VMR, VTC and stimulation protocol. The single-subject design
% matrix (*.sdm) and GLM will be automatically saved to disk in the
% directory of the VTC file.
%
% Data: Getting Started Guide 4. Works with BrainVoyager 22
%------------------------------------------------------------------------
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')
bv.PrintToLog('Create design matrix in Matlab...');
[vmrName, vmrDir] = uigetfile('*.vmr','Please select the VMR file...');
vmrfilename = fullfile(vmrDir, vmrName);
vmr = bv.OpenDocument(vmrfilename);
bv.ResizeWindow(600,400);
[vtcName, vtcDir] = uigetfile('*.vtc','Please select the VTC file...');
vtcfilename = fullfile(vtcDir, vtcName);
bv.PrintToLog(['Link vtc file: ' vtcfilename]);
vmr.LinkVTC(vtcfilename);
[prtName, prtDir] = uigetfile('*.prt','Please select the protocol file...');
prtname = fullfile(prtDir, prtName);
vmr.LinkStimulationProtocol(prtname);
bv.PrintToLog(['Link stimulation protocol: ' prtname]);

vmr.ClearDesignMatrix();
vmr.AddPredictor('Faces_LVF');
vmr.SetPredictorValuesFromCondition('Faces_LVF', 'Faces_LVF', 1.0);
vmr.ApplyHemodynamicResponseFunctionToPredictor('Faces_LVF');
vmr.AddPredictor('Houses_RVF');
vmr.SetPredictorValuesFromCondition('Houses_RVF', 'Houses_RVF', 1.0);
vmr.ApplyHemodynamicResponseFunctionToPredictor('Houses_RVF');
vmr.AddPredictor('Faces_CVF');
vmr.SetPredictorValuesFromCondition('Faces_CVF', 'Faces_CVF', 1.0);
vmr.ApplyHemodynamicResponseFunctionToPredictor('Faces_CVF');
vmr.AddPredictor('Houses_LVF');
vmr.SetPredictorValuesFromCondition('Houses_LVF', 'Houses_LVF', 1.0);
vmr.ApplyHemodynamicResponseFunctionToPredictor('Houses_LVF');
vmr.AddPredictor('Faces_RVF');
vmr.SetPredictorValuesFromCondition('Faces_RVF', 'Faces_RVF', 1.0);
vmr.ApplyHemodynamicResponseFunctionToPredictor('Faces_RVF');
vmr.AddPredictor('Houses_CVF');
vmr.SetPredictorValuesFromCondition('Houses_CVF', 'Houses_CVF', 1.0);
vmr.ApplyHemodynamicResponseFunctionToPredictor('Houses_CVF');
vmr.SaveSingleStudyGLMDesignMatrix(fullfile(vtcDir, 'FacesHousesDesignMatrix.sdm'));
bv.PrintToLog(['Saved design matrix: ' fullfile(vtcDir, 'FacesHousesDesignMatrix.sdm')]);
% vmr.FirstConfoundPredictorOfSDM
% run the GLM
vmr.SDMContainsConstantPredictor = 0;
vmr.CorrectForSerialCorrelations = 2; % If set to 1, AR(1) is used, if set to 2, AR(2) is used.
vmr.ComputeSingleStudyGLM;
vmr.ShowGLM;
vmr.SaveGLM(fullfile(vtcDir, 'FacesHouses_VTC.glm'));
bv.PrintToLog(['Saved GLM: ' fullfile(vtcDir, 'FacesHouses_VTC.glm')]);
```

5.3.4 Script to create a multi-study design matrix

```
%------------------------------------------------------------------------------------------------
% create_multistudy_designmatrix.m; script to create BrainVoyager multi-study design matrix
% and compute GLM in Matlab.
% Script assumes that functional data (*.vtc) are in same folder as
% single-subject design matrix files (*.sdm) and automatically pairs the
% runs with the design matrix files based on the names (*run1*, *run2*).
% Works with BrainVoyager 22
%------------------------------------------------------------------------------------------------
% create COM server, ask for the anatomy (*.vmr) and single-subject design matrix (*.sdm) files
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')
[FileName PathName] = uigetfile('*.vmr')
doc = bv.OpenDocument([PathName FileName]);
doc.ClearMultiStudyGLMDefinition;
[FileName PathName] = uigetfile('*.sdm', 'Select the design matrix file for run 1')
sdm1 = fullfile(PathName, FileName);
[FileName PathName] = uigetfile('*.sdm', 'Select the design matrix file for run 2')
sdm2 = fullfile(PathName, FileName);
% create the multi-study design matrix
vtcstruct = dir([PathName '*.vtc']) % assumes all files are in same folder
```

```
subjset = size(vtcstruct)
for i = 1:subjset(1)
    isrun1 = strfind(vtcstruct(i).name, 'run1')
    if ~isempty(isrun1)
        doc.AddStudyAndDesignMatrix(vtcstruct(i).name, sdm1);
    else
        doc.AddStudyAndDesignMatrix(vtcstruct(i).name, sdm2);
    end
end
doc.SaveMultiStudyGLMDefinitionFile(fullfile(PathName, 'MultiStudy_fromBVMatlabScript.mdm'))
doc.LoadMultiStudyGLMDefinitionFile(fullfile(PathName, 'MultiStudy_fromBVMatlabScript.mdm'))
doc.ComputeMultiStudyGLM
```

Chapter 6

Statistical analysis

6.1 Computing General Linear Models (*.glm)

Via the `ComputeSingleStudyGLM()`, `ComputeMultiStudyGLM()` en `ComputeRFXGLM()` scripting functions, the beta weights for the model defined in the design matrix are computed (see figure 6.1). This expresses to what extent the actual values can be explained from the model formulated in the design matrix.

6.1.1 Computing a single study GLM

Since the model is saved in a design matrix file (*.sdm), this should be loaded first. The logical sequence of steps to compute a single study general linear model (GLM) is then the following.

1. Start BrainVoyager Start BrainVoyager from Matlab via
   `bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')`
2. Open a document An FMR or VMR document needs to be opened via
   `vmrdoc = bv.OpenDocument( <name> ).`
3. Link VTC In case of a VMR file, link the corresponding VTC file to the VMR with the method `vmrdoc.LinkVTC( <name> )`. Provide the function with the name of the VTC file that should be linked to the VMR file.
4. Link stimulation protocol Link the stimulation protocol defined in the *.prt file via `vmrdoc.LinkStimulationProtocol( <name without path> );` . This file should be in the same folder as the document.
5. Load design matrix Load the model in the reference time course file (*.sdm) via `vmrdoc.LoadSingleStudyGLMDesignMatrix( <name without path> );` . The file should also reside in the same folder as the document.
6. Set appropriate flags
   - Autocorrelation removal Set the flag for removal of the autocorrelation of the noise. This can be performed setting the BrainVoyager document property `vmrdoc.CorrectForSerialCorrelations` to true.
   - Normalize time courses Also can be chosen to normalize the time courses. This can be done by setting one of the document properties `PSCTransformStudies`, to transform to percent signal change, or `ZTransformStudies` or `ZTransformStudiesBaselineOnly`, to z-transform, to true.
   - Indicate whether a confound is present If a confound has been used in the design matrix, set the property `SDMContainsConstantPredictor` to true.

- Start of confounds in model To indicate from which column in the design matrix the predictors are confounds, use the property `FirstConfoundPredictorOfSDM`.

7. Compute the GLM The necessary preparations are entered now and the General Linear Model can be computed via `ComputeSingleStudyGLM()`. This method does not require additional arguments.

8. Save GLM results The results of the computation of the GLM can be saved via the `SaveGLM( <name> )` method. The argument that should be provided is the name for the *.glm file.

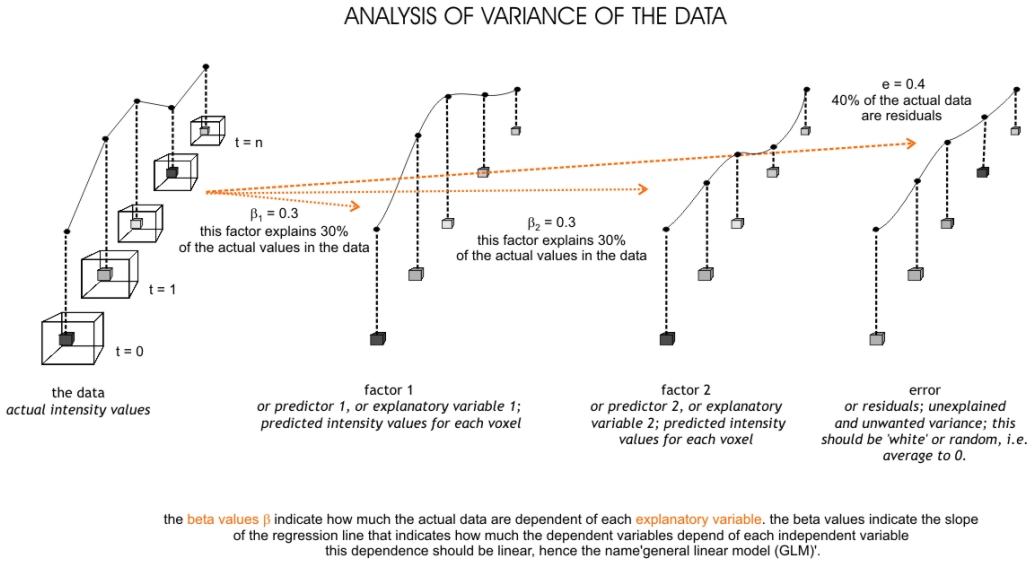


Figure 6.1: Analysis of variance

6.1.2 Computing a multi study GLM

1. Start BrainVoyager Start BrainVoyager from Matlab via
   `bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')`
2. Open a VMR document A VMR document needs to be opened via
   `vmrdoc = bv.OpenDocument( <name> ).`
3. Load design matrix Load the models (*.sdm) and functional data (*.fmr/*.vtc) listed in the multi-study GLM definition file (*.mdm) via
   `vmrdoc.LoadMultiStudyGLMDefinitionFile( <name without path> );` . The file should also reside in the same folder as the document.
4. Set appropriate flags
   - Autocorrelation removal Set the flag for removal of the autocorrelation of the noise. This can be performed setting the BrainVoyager document property `vmrdoc.CorrectForSerialCorrelations` to true.
   - Normalize time courses Also can be chosen to normalize the time courses. This can be done by setting one of the document properties `PSCTransformStudies`, to transform to percent signal change, or `ZTransformStudies` or `ZTransformStudiesBaselineOnly`, to z-transform, to true.
5. Compute the GLM The necessary preparations are entered now and the General Linear Model can be computed via `ComputeMultiStudyGLM`. This method does not require additional arguments.
6. Save GLM results The results of the computation of the GLM can be saved via the `SaveGLM( <name> )` method. The argument that should be provided is the name for the *.glm file.

6.1.3 List of functions

Table 6.1: GLM files

Description of function	*BrainVoyager API*	*Since*
Load GLM	LoadGLM()	1.3
Show GLM	ShowGLM()	1.3
Get/set serial correlation correction	CorrectForSerialCorrelations (property)	1.3
Apply z-transform	ZtransformStudies (property) ZtransformStudiesBaselineOnly (property)	
Apply percent signal change transformation	PSCTransformStudies (property)	
Get/set separate predictor for each study	SeparationOfStudyPredictors (property)	
Get/set separate predictor for each subject	SeparationOfSubjectPredictors (property)	
Indicates first confound predictor column	FirstConfoundPredictorOfSDM (property)	2.1
Indicates presence of constant predictor	SDMContainsConstantPredictor (property)	2.1
Compute GLM	ComputeSingleStudyGLM()	
Compute multi-study GLM	ComputeMultiStudyGLM()	1.3
Compute random effects GLM	ComputeRFXGLM()	
Save GLM	SaveGLM('name')	1.3

6.1.4 Scripts

The following script computes a GLM in BrainVoyager 1.9, opens the GLM in Matlab via the BVQXtools (now NeuroElf) and displays the betavalues of the middle slice in the first volume.

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% compute_glm_display_betaweights.m
% Script to compute a GLM, reads it in Matlab via NeuroElf (see www.neuroelf.net)
% and display the betaweights in Matlab
%
% Should work with BrainVoyager 20.4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[FileName,PathName] = uigetfile('*.fmr','Please select the functional data file...');
fmrfilename = strcat(PathName, FileName);
[FileName,PathName] = uigetfile('*.prt','Please select the protocol file...');
prtfilename = strcat(PathName, FileName);
[FileName,PathName] = uigetfile('*.rtc','Please select the design matrix file...');
rtcfilename = strcat(PathName, FileName);
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1');
bv.ShowLogTab;
bv.PrintToLog('Start computing GLM from Matlab...');
fmrdoc = bv.OpenDocument(fmrfilename);
fmrdoc.LinkStimulationProtocol(prtfilename);
fmrdoc.CorrectForSerialCorrelations  = 1;
fmrdoc.PSCTransformStudies              = 1;
bv.PrintToLog(['Loading ' rtcfilename '...']);
fmrdoc.LoadSingleStudyGLMDesignMatrix(rtcfilename); % design matrix (*.rtc)
fmrdoc.ComputeSingleStudyGLM();
fmrdoc.ShowGLM();
[pathstr,name,ext] = fileparts(fmrfilename);
glmfilename = fullfile(pathstr, [name '.glm']);
bv.PrintToLog(['Saving ' glmfilename '...']);
fmrdoc.SaveGLM(glmfilename);
answer = questdlg('Close document?', 'Compute GLM');
if answer == 'Yes'
    fmrdoc.Close;
end
% Now read the GLM file in via NeuroElf
h = msgbox('Please select the NeuroElf directory');
uiwait(h);
addpath(uigetdir);
glm = xff('GLM');
glm = xff(glmfilename);
glm.GLMData % shows fields in structure
betaweights4d = glm.GLMData.BetaMaps;
volsize = size(betaweights4d)
slicenr = num2str(floor(volsize(3)/2));
betaweights2d = betaweights4d(:,:,floor(volsize(3)/2),1);
figure;subplot(1,2,1); surf(double(betaweights2d));
title(['surface plot of beta values slice ' slicenr ' map 1']); axis image xy off
subplot(1,2,2); contour(betaweights2d);
title(['contourplot of beta values slice ' slicenr ' map 1']); axis image xy off
```

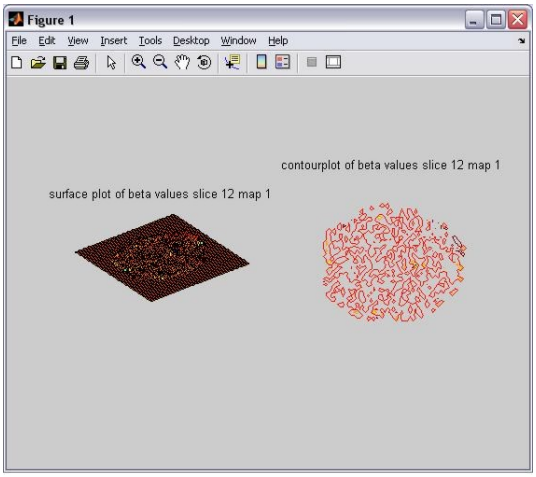


Figure 6.2: Plot of beta values

6.2 Manipulating contrasts (*.ctr)

By setting contrasts, different hypotheses can be tested after the GLM has been computed. BrainVoyager provides several methods to add contrasts and set the values of the contrasts (see the list of functions). To display the statistical map, use `ShowGLM()`.
The quickest way to set all contrasts simultaneously is via the method `SetContrastString( " values " )`. For example, setting the contrast shown in figure 6.3, requires the following steps:

- Add the contrast The contrast can be added via `AddContrast( <name> )`:
  `doc.AddContrast('Objects_in_RVF > Objects_in_LVF');`
- Set the contrast to current The current contrast
  can be set via `SetCurrentContrast( <name> )`:
  `doc.SetCurrentContrast('Objects_in_RVF > Objects_in_LVF');`
- Set the values of the contrasts for each predictor Setting the contrast values
  can be performed via indexing or strings. `doc.SetContrastString('1 -1 0');`
  Please note here that the third value in the string (text), represents the confound. The contrast value can be set to 0.

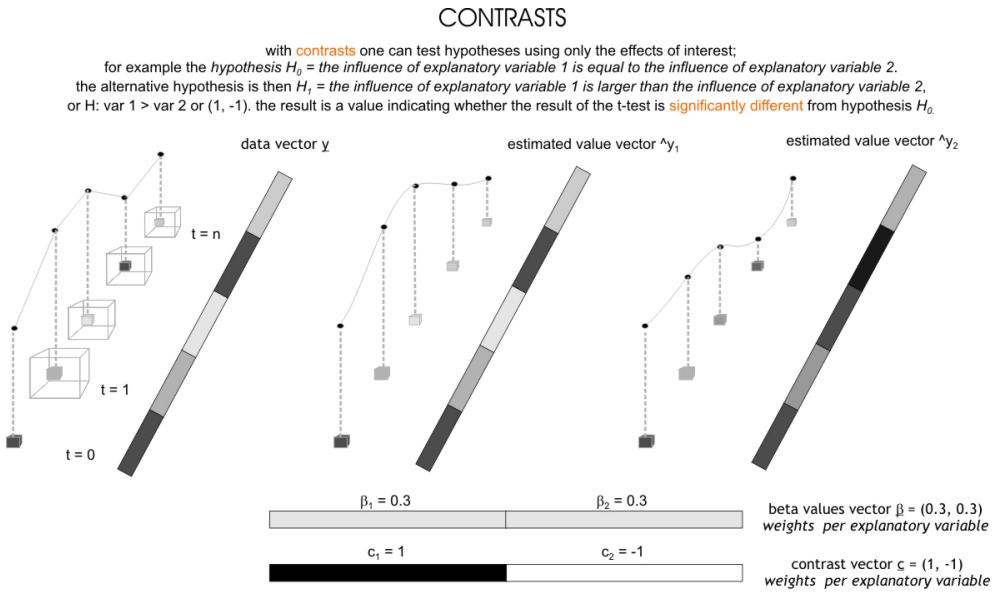


Figure 6.3: Contrasts

6.2.1 List of functions

Table 6.2: CTR files

Description of function	*Function in BrainVoyager API*	*Since*
Clear contrasts	ClearContrasts()	1.3
Add contrast	AddContrast('name')	1.3
Select contrast to change	SetCurrentContrast()	
Set contrast at index	SetCurrentContrastIndex()	1.3
Set contrast(s)	SetContrastString('text')	
Set contrast value	SetContrastValue(condition name, number)	
Set contrast value at index	SetContrastValueAtIndex(index, number)	1.3

6.3 Volume maps (*.vmp)

Since BrainVoyager 20.x the functions LoadVolumeMaps() and ShowVolumeMap() can be used. Since BrainVoyager 23, SaveVolumeMaps() has been added and functions to control the volume maps dialog.

Table 6.3: VMP files

Description of function	*BrainVoyager API*	*Since version*
Load a *.vmp file	LoadVolumeMaps(filename)	20.x
Show a map of a *.vmp file	ShowVolumeMap(index)	20.x
Get name of volume map (*.vmp())	GetNameOfVolumeMap()	BV 20.0
Save volume map (*.vmp())	SaveVolumeMaps(filename)	BV 23.0
Show volume map dialog in BV	ShowVolumeMapsDlg()	BV 23.0
Hide volume map dialog in BV	HideVolumeMapsDlg()	BV 23.0

6.4 Statistical analysis on volumes-of-interest (*.voi)

A GLM can be applied to one or several volumes-of-interest (*.voi) using the functions listed below.

6.4.1 List of functions

Please note that the VOI functions that require the number of the VOI as argument, are 0-based (e.g. the first VOI has index 0).

Table 6.4: VOI functions

Description of function	*Function in BV API*	*Since*
Load VOI file (*.voi)	LoadVOIFile('name')	2.8.4
Hide VOIs	HideAllVOIs	2.8.4
Select VOI for display	SelectVOI(index)	2.8.4
Deselect VOI	DeselectVOI(number)	2.8.4
Show selected VOIs	ShowSelectedVOIs()	2.8.4
Prepare VOI contrasts	PrepareROIContrasts('number of predictors')	2.8.4
Add a VOI contrast	AddROIContrast('name', 'contrast string')	2.8.4
Run GLM on VOI	ComputeSingleStudyGLMForVOI(VOI index, time course normalisation, serial corr correction)	2.8.4
Get name of VOI contrast	GetNameOfROIContrast(VOI index)	2.8.4
Get t-value of VOI contrast	GetTValueOfROIContrast(VOI index)	2.8.4
Get p-value of VOI contrast	GetPValueOfROIContrast(VOI index)	2.8.4
Get β-name of VOI	GetBetaNameOfROIGLM(VOI index)	2.8.4
Get β-value of VOI	GetBetaValueOfROIGLM(VOI index)	2.8.4

6.4.2 List of properties

Table 6.5: VOI properties

Description of function	*Property in BV API*	*Since*
Returns the number of volumes-of-interest (VOI)	NrOfVOIs (property)	2.8.4
Returns the name of the linked VOI	GetNameOfVOI (property)	2.8.4
Returns the number of predictors in design matrix	NrOfPredictorsInSingleStudyDM (property)	2.8.4
Returns the number of time points (volumes) in design matrix	NrOfTimePointsInSingleStudyDM (property)	2.8.4
Returns the number of VOI contrasts	NrOfROIContrasts (property)	2.8.4

6.4.3 Scripts

The following script calculates a VOI-GLM in BrainVoyager QX 2.8.4; the script can be run without modifications. The required files are requested for via file dialogs.

```
%------------------------------------------------
%  NewCommands_v284.m
%  New script commands in BrainVoyager v2.8.4 support VOI processing
%  including running single-study GLM analyses
%
%  Rainer Goebel 2014
%  Adapted for Matlab HB, November 2014
%  Checked for BVQX 2.8.4, 25-11-14; modified for BV 27-04-21
```

```
% Should work with BrainVoyager 20.4
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% Load mesh from mesh scene
% Start BrainVoyager
success = 1;
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')

% Load anatomical file (*.vmr)
[FileName, PathName] = uigetfile('*.vmr', 'Please select the anatomical file (*.vmr)');
vmrname = [PathName FileName];
% vmrname = 'K:\BVQXSampleData\ObjectsDicomGSG\ObjectsDicomGSG\CG2_3DT1FL_CLEAN.vmr';
[pathstrVMR, nameVMR, extVMR] = fileparts(vmrname);
docVMR = bv.OpenDocument(vmrname);
if (~isinterface(docVMR)), disp('Could not open VMR.'); success = 0; end;

if (success > 0)

    [FileName, PathName] = uigetfile('*.vtc', 'Please select the volume time course (*.vtc)');
    vtcName = [PathName FileName];
    [FileName, PathName] = uigetfile('*.voi', 'Please select the volumes-of-interest file (*.voi)');
    VOIFile = [PathName FileName];
    [FileName, PathName] = uigetfile({'*.sdm';'*.rtc'}, 'Please select the design matrix file (*.sdm, *.rtc)');
    sdmName = [PathName FileName];
    bv.ShowLogTab;

    docVMR.LinkVTC(vtcName);
    ok = docVMR.LoadVOIFile(VOIFile);
    if (~ok)
        bv.PrintToLog('Could not open .VOI file');
        %return;
    end

    n_vois = docVMR.NrOfVOIs;
    bv.PrintToLog(['No. of VOIs: ' num2str(n_vois)]);
    for i=0:(n_vois-1)
        voi_name = docVMR.GetNameOfVOI(i);
        bv.PrintToLog(['Name of VOI ' num2str(i+1) ': ' voi_name]);
    end

    docVMR.HideAllVOIs;
    % select all vois for display
    for i=0:(n_vois-1)
        docVMR.SelectVOI(i); % VOI index is 0-based!
    end
    docVMR.ShowSelectedVOIs;

    docVMR.LoadSingleStudyGLMDesignMatrix(sdmName);
    n_preds = docVMR.NrOfPredictorsInSingleStudyDM;
    n_points = docVMR.NrOfTimePointsInSingleStudyDM;

    docVMR.PrepareROIContrasts(n_preds); % param: number of predictors
    docVMR.AddROIContrast('Left vs Baseline', '1 0 0 0');
    docVMR.AddROIContrast('Right vs Baseline', '0 1 0 0');
    docVMR.AddROIContrast('Left vs Right', '1 -1 0 0');

    % param 1: VOI index (0-based)
    % param 2: time course normalization, 0 -> none, 1 -> percent change, 2 -> z, 3 -> z in baseline periods
    % param 3: serial correlation correction, 0 -> none, 1 -> with AR(1) model, 2 -> with AR(2) model
    docVMR.ComputeSingleStudyGLMForVOI(0, 1, 2);

    % Main stats for contrasts are accessible (full GLM/contrast results are printed in Log)
    for i=0:(docVMR.NrOfROIContrasts-1)%; i++) {
        c_name = docVMR.GetNameOfROIContrast(i);
        %c_t = docVMR.GetTValueOfROIContrast(i);
        %c_p = docVMR.GetPValueOfROIContrast(i);
        %bv.PrintToLog(['VOI-GLM result for contrast ' num2str(i+1) ...
         %' (' c_name '): t = ' num2str(c_t) '  p = ' num2str(c_p)]);
    end
end
```

Chapter 7

Surface meshes

Since BrainVoyager QX 2.8, the scripting methods available for meshes are considerately elaborate. It is possible to load meshes, inflate a mesh and morph to sphere, create and preprocess mesh time courses and perform cortex-based alignment in two ways and run single subject GLMs. The new functions can be invoked from either the new mesh object or the mesh scene object.
In some cases the mesh object might need to be called something else as 'mesh' in Matlab, for example 'meshSRF', since this might result in a structure instead of a COM interface object. To check whether the object is a COM object, use `isinterface(meshSRF)` (answer should be positive).

7.1 Loading meshes and capture the screen step-by-step

Start BrainVoyager from Matlab:

```
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')
```

Get the name of the VMR file:

```
[FileName,PathName] = uigetfile('*.vmr',...
'Please select the VMR file...');
```

Create the VMR name:

```
vmrname = [PathName FileName]
```

Open a VMR document:

```
vmr = bv.OpenDocument(vmrname).
```

Get the name of the mesh:

```
[FileName,PathName] = uigetfile('*.srf',...
'Please select the surface file...');
```

Create the SRF name:

```
srfname = [PathName FileName]
```

Load the mesh:

```
meshscene = vmr.CreateMeshScene;
meshscene.LoadMesh(srfname);
mesh = meshscene.CurrentMesh;
```

Make a screenshot by providing an image name:

```
meshscene.SaveSnapshotOf3DViewer('C:\Data\srfsnapshot.png');
```

7.1.1 List of surface functions of VMR object

Please find below a list of surface functions of the VMR object. Since BrainVoyager 20-22, the screen capture function has been replaced by the mesh scene function `SaveSnapshotOf3DViewer()` (since BrainVoyager 23: `SaveSnapshotOfViewer` and the function to update the surface window by the mesh function `UpdateAppearance()`.

Table 7.1: Surface (*.srf), mesh time course (*.mtc) and viewpoint settings (*.vwp) files

Description of function	*Function in Qt Script*	*Since*
Save SRF file	SaveMesh()	1.2
Load SRF file	LoadMesh()	1.2
Add SRF file	AddMesh()	1.2
Create mesh time course	CreateMTCFromVTC()	2.2
Link MTC file	LinkMTC()	2.2
Screen capture (obsolete)	SaveSnapshotOfSurfaceWindow()	1.x
Change viewpoint settings	ViewpointTranslationX (property)	1.2
	ViewpointTranslationY (property)	1.2
	ViewpointTranslationZ (property)	1.2
	ViewpointRotationX (property)	1.2
	ViewpointRotationY (property)	1.2
	ViewpointRotationZ (property)	1.2
Update surface window (obsolete)	UpdateSurfaceWindow()	1.x
Open surface module, get mesh scene object	CreateMeshScene()	3.0

7.1.2 List of functions of Mesh object

Table 7.2: Functions of Mesh object (since BVQX 2.8)

Function	*Parameter(s)*	*Returns*
SaveAs()	Name for mesh	-
CalculateCurvature()	-	-
CalculateCurvatureCBA()	-	*CURVATURE.smp
SmoothMesh()	No. smoothing iterations, force	-
SmoothGeometry()	No of iterations, smoothing force	success (boolean)
SmoothRecoMesh()	No. smoothing iterations, force	-
SmoothGeometrySimple()	No of iterations, smoothing force	success (boolean)
SmoothCurrentMap()	No. steps	-
SmoothMap()	index, no of cycles, onlyNonZero, onlyNonZeroNeighbours, onlyHigherThanThreshold	
SmoothMapLags()	index, no of cycles, smoothCircular, onlyNonZeroNeighbours, onlyHigherThanThreshold	
InflateMeshToSphere()	inflation steps	-
InflateGeometryToSphere()	No of iterations	Success (boolean)
InflateGeometryToSphereExt()	No of morphing iterations, morph force min, max, corr force min, max	Boolean
InflateMesh()	Morphing iterations, force, area reference mesh	-
InflateGeometry()	Morphing iterations, force, area reference mesh	
CorrectInflatedSphereMesh()	No. correction steps	
SimplifyGeometry()	Target no of vertices	String
CreateMultiScaleCurvatureMap()	level 1,2,3,4	Name curvature *.smp
SpatialSmoothing()	FHWM	-
LinearTrendRemoval()		
TemporalHighPassFilterFFT()	Cut-off value	
TemporalGaussianSmoothing()	kernel width, units (“Seconds”)	-
CreateMTCFromVTC()	Sampling VTC from, to, new *mtc name	Boolean
SaveMTC()	Name for *.mtc	
LinkMTC()	Name of *.mtc	Boolean
ClearDesignMatrix()	-	-
LoadSingleStudyGLMDesignMatrix()	Name of *.sdm file	Boolean
ComputeSingleStudyGLM()	-	-
ShowGLM()	-	-
SaveGLM()	Name for *.glm file	-
CreateSphericalCoordinatesMapFromSMP()	Name of *.smp	Name of spherical target curvature *.smp (or empty string)
SaveSingleStudyGLMDesignMatrix()	Name for design matrix (*.sdm) file	-
LoadMultiStudyGLMDefinitionFile()	Name of multi study design file (*.mdm)	
AddStudyAndDesignMatrixAndCortexMapping()	*.mtc, *.sdm, *.ssm	
SaveMultiStudyGLMDefinitionFile()	Name for multi study design file (*.mdm)	
ComputeMultiStudyGLM()	-	-
ComputeRFXGLM()	-	-
LoadGLM()	Name of *.glm file	-
SaveSurfaceMaps()	Name for surface map (*.smp) file	-
GetNameOfSurfaceMap()	Index (from 1)	Name
Update3DViewer()	Optional: process events (boolean)	-
LoadSurfaceMaps()	file name	-
ShowSurfaceMap()	map index	-
CreateSurfaceMapFromVolumeMap()	Interp, sampleOnlyNonZero	boolean
CreateSurfaceMapFromVolumeMapDepth()	Interp, sampleOnlyNonZero, sampleMaxVals, depthStart, depthEnd, stepSize	-
UpdateAppearance()	-	-

7.1.3 List of properties of Mesh object

The mesh (SRF) object has the following properties: `FileName`, `NrOfVertices`, `MorphingUpdateInterval`, `FileNameOfPreprocessdMTC`, `CorrectForSerialCorrelations`, `NrOfMTCVolumes`, `NrOfMTCTimePoints`, `NrOfSurfaceMaps`.

7.1.4 List of functions of MeshScene object

Table 7.3: Functions of MeshScene object (since BVQX 2.8)

Function	*Parameter(s)*	*Returns*
LoadMesh()	Name of mesh	Boolean
AddMesh()	Name of mesh	Boolean
MergeMeshesInScene()	-	String
MergeMeshes() (since BV 23.0)	-	String
CreateStandardSphereMesh()	-	Mesh object
MapSphereMeshFromStandardSphere()	-	*.ssm file
MapSphereFromStandardSphere()[1]	-	*.ssm file
SetStandardSphereToFoldedMesh()	Name of *SPHERE.srf	*SPH.srf name
ClearGroupCBACurvatureFiles()	-	-
AddCurvatureFileForGroupCBA()	*CURVATURE.smp	Boolean
RunRigidCBA()	Target curvature (*.smp)	Boolean
RunCBA()	-	Boolean
CreateAverageCurvatureGroupMap()	-	-
CreateAverageFoldedGroupMesh()	-	Boolean
UpdateSurfaceWindow() (obsolete)	-	-
SaveSnapshotOfSurfaceWindow() (obsolete)	Name for image file	Boolean
SaveSnapshotOf3DViewer()	Name for image file	Boolean
SaveSnapshotOfViewer()	Name for image file (BV 23.0)	Boolean

7.1.5 List of properties of MeshScene object

When invoking `meshScene.get`, one obtains the properties and values of the MeshScene object. These are `SphereResolutionCBA` (value 1: standard resolution, 2: low resolution, 3: high resolution), `ViewpointPositionX`, `ViewpointPositionY`, `ViewpointPositionZ`, `ViewpointRotationX`, `ViewpointRotationY`, `ViewpointRotationZ`, and `CurrentMesh`.

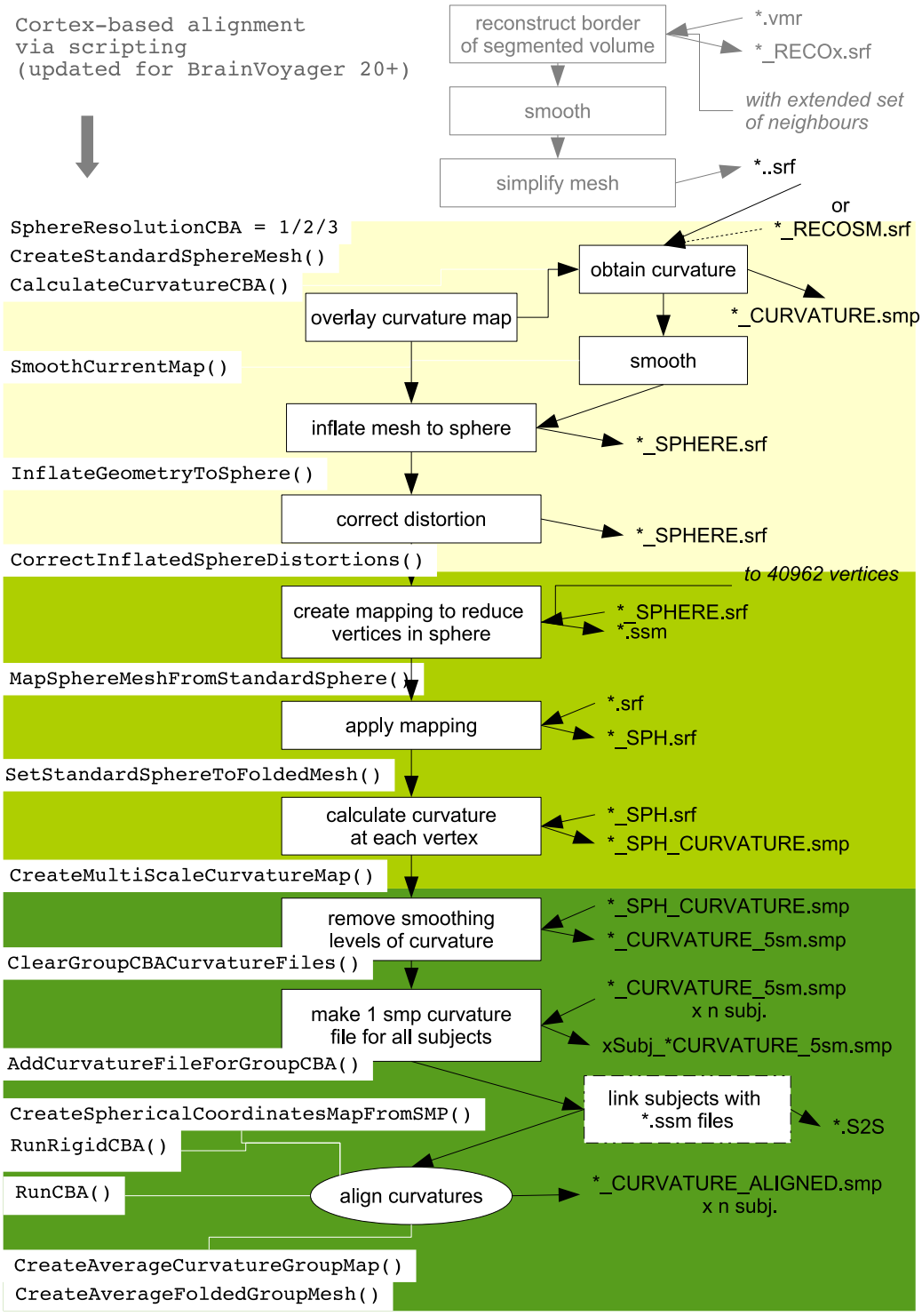


Figure 7.1: Automation of cortex-based alignment (CBA): schematic depiction of procedure and available scripts (please note: steps in grey (top of figure) are not included).

7.2 Scripts

7.2.1 Script to create MTC files

Below an example how a mesh time course (*.mtc) on a surface mesh (*.srf) could be created from functional data in a volume time course (*.vtc).
The first parameter for the `CreateMTCFromVTC()` function indicates the sampling depth in the white matter, the second in the gray matter, the third the name for the mesh time course.

```
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1');
[FileName,PathName] = uigetfile('*.vmr','Please select the anatomical data file...');
vmrfilename = strcat(PathName, FileName);
docVMR = bv.OpenDocument(vmrfilename);
docVMR.CreateMTCFromVTC(-1.0, 3.0, 'test.mtc' );
```

7.2.2 Script to load a mesh file

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MeshLoading.m
% For BrainVoyager QX 2.8
% Created by Rainer Goebel, May 2013
% Adaptations until BV 22 HB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
success = 1;
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1') % Start BrainVoyager
[FileName, PathName] = uigetfile('*.vmr', 'Please select the anatomical file (*.vmr)');
vmrname = [PathName FileName];
docVMR = bv.OpenDocument(fullfile(PathName, FileName));
if (~isinterface(docVMR)), disp('Could not open VMR.'); success = 0; end;
meshScene = docVMR.CreateMeshScene; % creates empty scene (surface view) if not available for docVMR
if (~isinterface(meshScene))
    disp('Could not create mesh scene object');
    success = 0;
end;
[FileName, PathName] = uigetfile('*.srf', 'Please select the mesh (*.srf)');
ok = meshScene.LoadMesh(fullfile(PathName, FileName));
if (ok)
    meshSRF = meshScene.CurrentMesh;
    n_vertices = meshSRF.NrOfVertices;
    bv.PrintToLog(['No. of vertices of current mesh: ' num2str(n_vertices)]);
    bv.PrintToLog(['Viewpoint position, x: ' num2str(meshScene.ViewpointPositionX)...
        ', y: ' num2str(meshScene.ViewpointPositionY) ', z: ' num2str(meshScene.ViewpointPositionZ)]);
    meshScene.ViewpointRotationX = 0;
    meshScene.ViewpointRotationY = 180;
    meshScene.ViewpointRotationZ = 0;
    for i=1:90
        meshScene.ViewpointPositionX = meshScene.ViewpointPositionX + 2;
        meshScene.ViewpointRotationZ = meshScene.ViewpointRotationZ - 1;
        meshSRF.UpdateAppearance(1);
    end;
    bv.PrintToLog(['Viewpoint position, x: ' num2str(meshScene.ViewpointPositionX)...
        ', y: ' num2str(meshScene.ViewpointPositionY) ', z: ' num2str(meshScene.ViewpointPositionZ)]);
end
```

7.2.3 Script to smooth and inflate a mesh

```
%--------------------------------------------------------------------
% MeshMorphing.m
% This script shows how a mesh can be smoothed and inflated.
% It is assumed that a "*_RECO.srf" mesh is available (e.g. loaded).
% Should work with BrainVoyager 20.4
%
% Created by Rainer Goebel, May 2013
% Adapted for Matlab by Hester Breman, June 2013, April 2021
%--------------------------------------------------------------------

% Start BrainVoyager
success = 1;
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')

% Load anatomical file (*.vmr)
[FileName, PathName] = uigetfile('*.vmr', 'Please select the anatomical file (*.vmr)');
vmrname = [PathName FileName];
```

```
[pathstrVMR, nameVMR, extVMR] = fileparts(vmrname);
docVMR = bv.OpenDocument(vmrname);
if (isempty(docVMR)), disp('Could not open VMR.'); success = 0; end;

% Obtain MeshScene object
meshScene = docVMR.CreateMeshScene;% creates empty scene (surface view)
if (~isinterface(meshScene))
    disp('Could not create mesh scene object');
    success = 0;
end;

% Select surface file (*_RECO.srf)
if (success == 1)
    [FileName, PathName] = uigetfile('*.srf', 'Please select the mesh (*_RECO.srf)');
    srfname = [PathName FileName];
    [pathstrSRF, nameSRF, extSRF, versnSRF] = fileparts(srfname);
    ok = meshScene.LoadMesh(srfname);
    if (~ok), success = 0;
    else meshSRF = docVMR.CurrentMesh; % cannot use 'mesh' as object name!
    end;
end

% Smooth and inflate the *_RECO.srf
if (success == 1)
    meshSRF.MorphingUpdateInterval = 25;
    %mesh.SmoothMesh(30, 0.07);%/ standard smoothing - mesh will shrink
    meshSRF.SmoothRecoMesh(50, 0.07);
    % SmoothRecoMesh(): special "high-frequency" smoothing: removing jags of reconstructed
    % voxel borders without shrinking mesh
    meshSRF.CalculateCurvature();
    meshSRF.SmoothCurrentMap(5);
    meshSRF.MeshScene.UpdateSurfaceWindow();
    new_name = fullfile(pathstrSRF, [nameSRF 'SM' extSRF]);
    meshSRF.SaveAs(new_name);
    meshSRF.InflateMesh(500, 0.8, '');
    % InflateMesh(): if "" used for 3rd param (area reference mesh), the current mesh
    % at the moment the function is called is used to calculate ref mesh area
end;
```

7.2.4 Script to run GLM on surface

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MeshMTCSingleStudyGLM.m
% Example script showing how to run statistics with the new "mesh" object
%
% Created by Rainer Goebel, May 2013, HB June 2013, April, June 2021
% Works with BrainVoyager 22
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Start BrainVoyager
success = 1;
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')

% Load anatomical file (*.vmr)
[FileName, PathName] = uigetfile('*.vmr', 'Please select the anatomical file (*.vmr)');
docVMR = bv.OpenDocument(fullfile(PathName, FileName));
if (~isinterface(docVMR)), disp('Could not open VMR.'); success = 0; end;

% Load surface file (*.srf)
[FileName, PathName] = uigetfile('*.srf', 'Please select the mesh (*.srf)');
srfname = [PathName FileName];
ok = docVMR.LoadMesh(srfname);
if (~isinterface(docVMR)), ok = 0;
else meshSRF = docVMR.CurrentMesh;
    if (~isinterface(meshSRF)), ok = 0; end;
end;

% Link mesh time course (*.mtc)
if (ok)
    [FileName, PathName] = uigetfile('*.mtc', 'Please select the mesh (*.mtc)');
    ok = meshSRF.LinkMTC(fullfile(PathName, FileName)); %if(!ok) return;
end;

% Load single subject design matrix (*.sdm)
if (ok)
    meshSRF.ClearDesignMatrix();
    [FileName, PathName] = uigetfile('*.sdm', 'Please select the design matrix (*.sdm)');
```

```
    sdmname = fullfile(PathName, FileName);
    [pathstrSDM, nameSDM, extSDM] = fileparts(sdmname);
    ok = meshSRF.LoadSingleStudyGLMDesignMatrix(sdmname); %if(!ok) return;
end;

% Run the single subject GLM
if (ok)
    meshSRF.CorrectForSerialCorrelations = 2; % 1 -> AR(1), 2 -> AR(2)
    meshSRF.ComputeSingleStudyGLM();
    meshSRF.ShowGLM();
    meshSRF.SaveGLM(fullfile(pathstrSDM, [nameSDM '_MeshGLM_FROMSCRIPT.glm']));
end;

% docVMR.Close();
% ok = meshSRF.LinkStimulationProtocolToMTC(ObjectsRawDataPath + 'CG_OBJECTS_FROMSCRIPT.prt');
```

7.2.5 Script to run cortex-based alignment

The procedure to perform rigid and full cortex-based alignment (CBA) via scripting is provided in the script below. A schematic presentation can be found in figure 7.1.

```
%  MeshCBA.m
%  This script shows how to prepare and run (rigid) CBA including group
%  mesh/curvature map visualization.
%
%  Created by Rainer Goebel, May 2013
%  Adapted for Matlab HB, June 2013

% Start BrainVoyager
success = 1;
bv = actxserver('BrainVoyager.BrainVoyagerScriptAccess.1')

% Load anatomical file (*.vmr)
[FileName, PathName] = uigetfile('*.vmr', 'Please select the anatomical file (*.vmr)');
vmrname = [PathName FileName];
[pathstrVMR, nameVMR, extVMR] = fileparts(vmrname);
docVMR = bv.OpenDocument(vmrname);
if (isempty(docVMR)), disp('Could not open VMR.'); success = 0; end;

% Obtain MeshScene object
meshScene = docVMR.MeshScene; % creates empty scene (surface view) if not available for docVMR
if (~isinterface(meshScene))
    disp('Could not create mesh scene object');
    success = 0;
end;

% Select surface file (*_RECOSM.srf)
if (success == 1)
    [FileName, PathName] = uigetfile('*.srf', 'Please select the mesh (*_RECOSM.srf)');
    srfname = [PathName FileName];
    [pathstrSRF, nameSRF, extSRF, versnSRF] = fileparts(srfname);
    ok = meshScene.LoadMesh(srfname);
    if (~ok), success = 0;
    else meshSRF = docVMR.CurrentMesh; % cannot use 'mesh' as object name!
    end;
end

% Create sphere from folded mesh (*_RECOSM.srf), resulting in *_SPHERE.srf
meshSRF.MorphingUpdateInterval = 50;
meshSRF.CalculateCurvatureCBA();
meshSRF.SmoothCurrentMap(5);
meshSRF.MeshScene.UpdateSurfaceWindow();
meshSRF.InflateMeshToSphere(800);
meshSRF.CorrectInflatedSphereMesh(3000);
var new_name = fullfile(pathstrSRF, [nameSRF 'SPHERE' extSRF]);
meshSRF.SaveAs(new_name);

% Create mapping between subject and standard sphere (reduced vertices)
% Results in mapping file (*.ssm)

docVMR = bv.ActiveDocument;
if (~isinterface(docVMR)), success = 0; end;
meshScene = docVMR.MeshScene;
if (~isinterface(meshScene)), success = 0; end;
meshSRF = meshScene.CurrentMesh;
if (~isinterface(meshSRF)), success = 0; end

% We assume created sphere mesh (see above)
```

```
% is available as current mesh (created or loaded) AND available on disk
% (since it is reloaded)
sphere_mesh_name =  meshSRF.FileName;
% We need name of folded original mesh below - here we assume that name
% is same as SPHERE mesh but instead of "SPHERE" has "RECOSM" at end of name
[pathstrSphere, nameSphere, extSphere, versnSphere] = fileparts(sphere_mesh_name);
folded_mesh_name = fullfile(pathstrSphere, [nameSphere '_RECOSM' extSphere]);

meshScene.SphereResolutionCBA = 1; % 1 -> standard resolution (default),
                                   % 2 -> low resolution, 3 -> high resolution
saved_ssm_file = meshScene.MapSphereMeshFromStandardSphere();
saved_foldedSPH_file = meshScene.SetStandardSphereToFoldedMesh(folded_mesh_name);
meshSRF = meshScene.CurrentMesh;
saved_curvature_file  = meshSRF.CreateMultiScaleCurvatureMap(2, 7, 20, 40);
% The obtained curvature file (automatically saved) is needed for CBA -
% store in array, one for each subject (per hemisphere)
bv.PrintToLog(['Curvature file: ' saved_curvature_file]);

% Run rigid CBA, input must be sphere (*_SPH.srf)
docVMR = bv.ActiveDocument;
if (~isinterface(docVMR)), success = 0; end;
meshScene = docVMR.MeshScene;
if (~isinterface(meshScene)), success = 0; end;
% To ensure that we have a sphere mesh, we can create one
meshScene.CreateStandardSphereMesh();
meshSRF = meshScene.CurrentMesh;
meshScene.ClearGroupCBACurvatureFiles();
NSubjects = 5;
dirName = uigetdir(pathstrSRF);
for i=1:NSubjects % loop over subjects (per hemisphere)

    % For demonstration, we simply hard-code files here - better to use/prepare array
    if (i == 0), CurvatureFile = [dirName filesep 'subj_AA/AA_TAL_LH_RECOSM_SPH_CURVATURE.smp']; end;
    if (i == 1), CurvatureFile = [dirName filesep 'subj_MR/MR_TAL_LH_RECOSM_SPH_CURVATURE.smp']; end;
    if (i == 2), CurvatureFile = [dirName filesep 'subj_PD/PD_TAL_LH_RECOSM_SPH_CURVATURE.smp']; end;
    if (i == 3), CurvatureFile = [dirName filesep 'subj_RM/RM_TAL_LH_RECOSM_SPH_CURVATURE.smp']; end;
    if (i == 4), CurvatureFile = [dirName filesep 'subj_SG/SG_TAL_LH_RECOSM_SPH_CURVATURE.smp']; end;

    ok = meshScene.AddCurvatureFileForGroupCBA(CurvatureFile);
    if (~ok), success = 0; end; % provided file not found

    % Here we use first case for rigid alignment target (any should be fine)
    if (i == 1), TargetCurvatureFile = CurvatureFile; end;
end;

SphericalTargetCurvatureFile = meshSRF.CreateSphericalCoordinatesMapFromSMP(TargetCurvatureFile);
if isempty(SphericalTargetCurvatureFile), success = 0; end; % we get empty string if it did not work

success = meshScene.RunRigidCBA(SphericalTargetCurvatureFile);
% The result of rigid alignment is automatically saved in a ".rga" file that will be used
% by subsequent CBA when using same input curvature SMP files

% Run full CBA
docVMR = bv.ActiveDocument;
if (~isinterface(docVMR)), success = 0; end;
meshScene = docVMR.MeshScene;
if (~isinterface(meshScene)), success = 0; end;

meshScene.ClearGroupCBACurvatureFiles(); % if this is run after rigid-CBA,
% we would not strictly fill curvature SMP list again
NSubjects = 5;
dirName = uigetdir(pathstrSRF);
for i=1:NSubjects % loop over subjects (per hemisphere)

    % For demonstration, we simply hard-code files here - better to use/prepare array
    if (i == 0), CurvatureFile = [dirName filesep 'subj_AA/AA_TAL_LH_RECOSM_SPH_CURVATURE.smp']; end;
    if (i == 1), CurvatureFile = [dirName filesep 'subj_MR/MR_TAL_LH_RECOSM_SPH_CURVATURE.smp']; end;
    if (i == 2), CurvatureFile = [dirName filesep 'subj_PD/PD_TAL_LH_RECOSM_SPH_CURVATURE.smp']; end;
    if (i == 3), CurvatureFile = [dirName filesep 'subj_RM/RM_TAL_LH_RECOSM_SPH_CURVATURE.smp']; end;
    if (i == 4), CurvatureFile = [dirName filesep 'subj_SG/SG_TAL_LH_RECOSM_SPH_CURVATURE.smp']; end;

    ok = meshScene.AddCurvatureFileForGroupCBA(CurvatureFile);
    if (~ok), success = 0; end; % provided file not found

end;
success = meshScene.RunCBA();

% These convenience functions can only be called after CBA (with internally stored set of .SRF, .SSM,
% .SMP file names): create group curvature map indicating quality of alignment, saves also ".cal" file
% that can be used in CBA dialog
success = meshScene.CreateAverageCurvatureGroupMap();
```

```
% Create folded group average cortex using created alignment (SSM) files, saves also ".sal" file that
% can be used in CBA dialog; the function only works if names of folded SPH meshes can be derived from
% curvature names by replacing trailing substring with "_RECOSM_SPH.srf"
success = meshScene.CreateAverageFoldedGroupMesh();
```

7.2.6 Script to read surface file

The following script reads a BrainVoyager surface file (*.srf) version 3 in Matlab and displays the mesh in different ways (please note, this script was written for older surface file versions). For all surface file versions, it is also possible to export the surface file to the surface exchange format GIFTI via the GIFTI plugin in BrainVoyager (see also the BrainVoyager plugins page and the GIFTI page on the support site: https://support.brainvoyager.com/brainvoyager/available-tools/86-available-plugins/63-gifti-conversion-plugin and http://www.nitrc.org/projects/gifti). Since BrainVoyager 23.0, GIFTI files can be imported directly into BrainVoyager.

Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% meshes.m
% Script to visualize a mesh with different axes in Matlab
% Works with *.srf files at least up to version 4.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[FileName,PathName] = uigetfile('*.srf','Please select the *.srf file...');
srfFile = strcat(PathName, FileName);
fid=fopen(srfFile,'rb'); % read only
%%%%% read in header data %%%%%
version_number = fread(fid, [1 1], 'float32'); % version number of surface file; default: 3
reserved = fread(fid, [1 1], 'uint32'); % reserved, must be '0'
NrOfVertices = fread(fid, [1 1], 'uint32'); % number of vertices
NrOfTriangles = fread(fid, [1 1], 'uint32'); % number of triangles
% COORDINATE VALUES
MeshCenterX = fread(fid, [1 1], 'float32'); % center of mesh on X-axis; default coordinate: 128
MeshCenterY = fread(fid, [1 1], 'float32'); % center of mesh on Y-axis; default coordinate: 128
MeshCenterZ = fread(fid, [1 1], 'float32'); % center of mesh on Z-axis; default coordinate: 128
VertexX = fread(fid, [1 NrOfVertices], 'float32'); % sequence of X coordinates of all vertices
VertexY = fread(fid, [1 NrOfVertices], 'float32'); % sequence of Y coordinates of all vertices
VertexZ = fread(fid, [1 NrOfVertices], 'float32'); % sequence of Z coordinates of all vertices
NormalX = fread(fid, [1 NrOfVertices], 'float32'); % sequence of X components of all vertex normals
NormalY = fread(fid, [1 NrOfVertices], 'float32'); % sequence of Y components of all vertex normals
NormalZ = fread(fid, [1 NrOfVertices], 'float32'); % sequence of Z components of all vertex normals
% COLORS OF THE SURFACE
RedConvCurv = fread(fid, [1 1], 'float32');
% R component of convex curvature color (range: 0.0 - 1-0); default: 0.322
GreenConvCurv = fread(fid, [1 1], 'float32');
% G component of convex curvature color (range: 0.0 - 1-0); default: 0.733
BlueConvCurv = fread(fid, [1 1], 'float32');
% B component of convex curvature color (range: 0.0 - 1-0); default: 0.980
AlphaConvCurv = fread(fid, [1 1], 'float32');
% Alpha component of convex curvature color (range: 0.0 - 1-0); default: 0.500
RedConvCurv = fread(fid, [1 1], 'float32');
% R component of convex curvature color (range: 0.0 - 1-0); default: 0.100
GreenConcCurv = fread(fid, [1 1], 'float32');
% G component of concave curvature color (range: 0.0 - 1-0); default: 0.240
BlueConcCurv = fread(fid, [1 1], 'float32');
% B component of concave curvature color (range: 0.0 - 1-0); default: 0.320
AlphaConcCurv = fread(fid, [1 1], 'float32');
% Alpha component of concave curvature color (range: 0.0 - 1-0); default: 0.500
MeshColor = fread(fid, [1 NrOfVertices], 'int32');
% sequence of color indices of all vertices (*1)
% ALL VERTEX NEIGHBOURS
vertexNeighbours = struct('VertexNr',{}, 'NrNeighbours',{},'Neighbours',{});
for vertexCounter=1:NrOfVertices
vertexNeighbours(vertexCounter).VertexNr = vertexCounter;
NrOfNeighbours = fread(fid, [1 1], 'int32');
vertexNeighbours(vertexCounter).NrNeighbours = NrOfNeighbours;
vertexNeighbours(vertexCounter).Neighbours = fread(fid, [1 NrOfNeighbours], 'int');
end
TriangleSequence = fread(fid, [1 NrOfTriangles*3], 'int');
% Sequence of three indices to constituting vertices of each triangle
NrOfTriangleStripElements = fread(fid, [1 1], 'int'); % Number of triangle strip elements
if (NrOfTriangleStripElements > 0)
    StripElemSequence = fread(fid, [1 NrOfTriangleStripElements], 'int32'); % Sequence of strip elements
end
fclose(fid);
% (*1) Each vertex has one color index. An index value of 0 references the convex curvature color,
% a value of 1 references the concave curvature color. Functional colors, RGB colors can be also represented.
% From example from Matlab Help documentation on multiple axes
tris = reshape(TriangleSequence,3,[])'+1;
figure; %(see Figure 7.2)
h(1) = axes('Position',[0 0 1 1]);
trimesh(tris, VertexX',VertexY',VertexZ');
```

```
h(2) = axes('Position',[0 0 .4 .6]); % closer, on the left lower corner
trimesh(tris, VertexX',VertexY',VertexZ');
h(3) = axes('Position',[0 .5 .5 .5]); % closer, on the left upper corner, stretched
trimesh(tris, VertexX',VertexY',VertexZ');
h(4) = axes('Position',[.5 0 .4 .4]); % closer, on the right lower corner, stretched
trimesh(tris, VertexX',VertexY',VertexZ');
h(5) = axes('Position',[.5 .5 .5 .3]); % closer, on the right upper corner, very stretched
trimesh(tris, VertexX',VertexY',VertexZ');
set(h,'Visible','off')
```

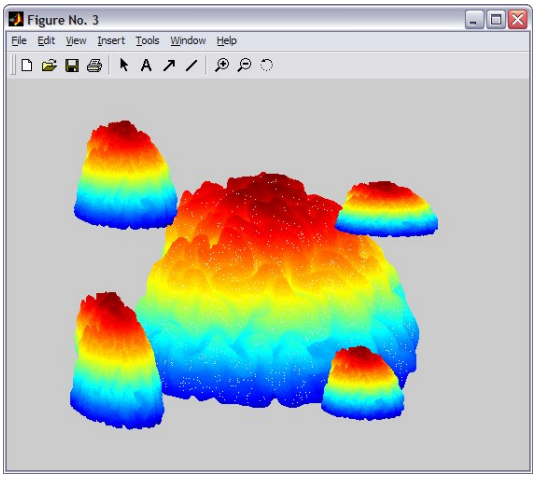


Figure 7.2: Display meshes with different axes

Chapter 8

Using NeuroElf

NeuroElf, formerly known as BVQXTools, is a toolbox created by Jochen Weber to read and manipulate BrainVoyager data in Matlab. The functionality in BrainVoyager and the NeuroElf can be nicely combined using COM (on Windows only).
For example, once a COM connection is established between Matlab and BrainVoyager, one can first instruct BrainVoyager to create and preprocess functional data, and even create VTC files. Then, the files can be read quickly into the Matlab workspace by NeuroElf.

To obtain NeuroElf, download the toolbox from
https://github.com/neuroelf (see Figure 8.1). Older versions are available at http://www.neuroelf.net/. To make NeuroElf available in Matlab, add the path of NeuroElf to Matlab. This can for example be realized via the instruction `addpath(uigetdir);`.

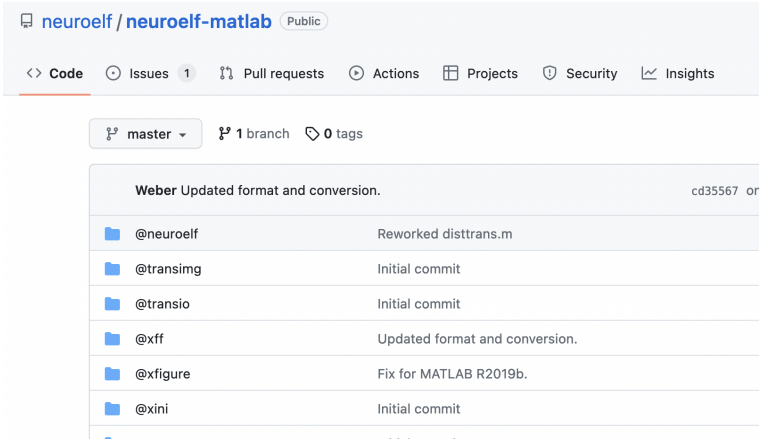


Figure 8.1: NeuroElf on GitHub

Bibliography

[1] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics; Principles and Practice*. Boston: Addison-Wesley Publishing Company, Inc., 1997.

[2] R. Goebel, F. Esposito, and E. Formisano. Analysis of FIAC data with BrainVoyager QX: From single-subject to cortically aligned group GLM analysis and self-organizing group ICA. *Human Brain Mapping*, 27(5):392–401, 2006.