Beamforming framework for SPM12

- Aims
  - Make the existing code accessible to all users.
  - Remove unnecessary wrappers and duplications.
  - Ease maintenance and further development.
  - Provide common API and GUI.
  - Enable combining different sub-methods in the same pipeline and easy replacement of pipeline segments.
  - Make the key code segments easily identifiable.
  - Enable easy pipeline saving and sharing.
- Constraints
  - Add new functionality with minimal changes to existing code
  - No need for immediate major code rewrites.
  - Everyone keeps their own familiar coding style for the critical code segments and has full control of their plugins. Modifying the skeleton requires some coordination.
  - Different intermediate computation results are easy to keep track of.

- Solution
  - SPM batch with automatically detectable plugins.
  - Common stable code parts are in the ‘skeleton’.
  - Plugins implement different possible operation at each of the stages of the pipeline.
  - Each plugin contains its own GUI definition for matlabbatch. These can easily be created by copying and modifying existing examples.
  - The setting are harvested by matlabbatch and passed to the plugin as input.
  - additional dependencies can be kept in /private.

- Data management (suggestion)
  - Each analysis is run in a separate directory which will contain a file BF.mat and possibly other files.
  - BF.mat contains a set of structs that are converted into fields of BF struct when loaded with bf_load. That makes it possible to only load the necessary fields.
  - If it turns out that BF.mat gets too big perhaps it should keep links to some of the stuff but the links should be relative and within the directory to enable copying.
  - Each pipeline stage has an associated BF field where its data are kept. Each stage can use the fields created by the previous stages.
  - Where possible data structs are similar to FT.
  - It is possible (but highly discouraged) for particular pipelines to have their own fields for all stages. In principle the fields should be common wherever possible.

Presently implemented stages

- data – takes D with head model and prepares everything necessary for subsequent stages.
- sources - define the source space (grid/mesh/set of sources etc.). Compute leadfields
- features – prepare the input for filter computation (e.g covariance matrix)
- inverse – compute inverse projectors (combined)
- output – use the filters to compute something useful (e.g. power on a grid). The data segments are defined anew for more flexibility.
- write – generate output (e.g. nifti images or D with source time courses).

To do

- Review and modify the skeleton structure
- Port the existing functionality to the framework.
- Validate against the old implementation
- The toolbox can be put on GoogleCode to enable easy access for everyone and separated from standard SPM releases.