BTMCS Software Manual

Version 1.2.1

Cosmina Hogea **Dongjin Kwon**

Contents

Brain Tumor Modeling - Coupled Solver (BTMCS) is a software package for tumor growth modeling employs a DiffusionSolver approach [JMB2008], [SJSC2008], [MICCAI2007]. It is a one-step further compared to the previous

purely mechanical, pressure-based approach employed in ElasticSolver and described in [PMB2007]. In addition to the previous elasticity-based approach to simulate brain tissue deformations caused by growing tumors (mass-effect), a nonlinear reaction-advection-diffusion equation to describe the tumor spatio-temporal evolution is added. This equation has a two-way coupling with the underlying tissue elastic deformation. In this approach, the forces exerted by the tumor growth and infiltration onto the underlying brain parenchyma are local ones, proportional to local tumor density gradients.

Curretnly BTMCS is used in GLioma Image SegmenTation and Registration (GLISTR) [MICCAI2014] and Pre-Operative and post-Recurrence brain Tumor Registration (PORTR) [TMI2014].

1 Download

1.1 Software License

The BrainTumorModeling_CoupledSolver software is freely available under a BSD-style open source license that is compatible with the Open Source Definition by The Open Source Initiative and contains no restrictions on use of the software. The full license text is included with the distribution package and available online.

1.2 Documentation

BTMCS Software Manual: The software manual of BrainTumorModeling_CoupledSolver in PDF.

1.3 System Requirements

Operating System: Linux, Windows (64 bit)

Memory Requirement: 4GB or more.

1.4 Register for Download

Please **register online** to receive an email with the download links of the software.

2 Installation

The following steps describe the procedure to build and install BrainTumorModeling_CoupledSolver (BTMCS).

2.1 Build Dependencies

Before building BTMCS, the following software libraries are required to be installed.

Package	Version	Description
CMake	2.8.4 or higher	To compile and build BTMCS. Use version 2.8.4 or higher.
PETSc	2.2.1 or higher	BTMCS is built upon the structure of PETSc. In Windows, use version 3.2 or higher.
ITK	3.14 or higher	For ITK 3.x or 4.x, there's no mandatory option. Required only when `BUILD_TESTING` is enabled in CMake setting of BTMCS. In Windows, build using CMake GUI with the Win64 (x64) solution platform of Visual Studio.

2.2 Build PETSc

BTMCS is built upon the PETSc library. PETSc library could be built as static libraries as follows. The advantage of static libraries is that once the application program is compiled and linked, the PETSc files could be removed as they are not required for the execution of the program. Here, we use the version 3.5.2 by an example (this procedure also works on versions >= 3.2).

Note: To open the shell in Windows, launch the command prompt by executing `Visual Studio x64 Win64 Command Prompt` in `Start > All Programs > Microsoft Visual Studio > Visual Studio Tools`. In this prompt, type `C:\cygwin\bin\bash --login` to open the Cygwin terminal. In the cygwin terminal, execute `mv /usr/bin/link.exe /usr/bin/link-cygwin.exe` to avoid the confusion with the Visual Studio link program. Alternatively, move `/usr/bin` folder to the end in the `PATH` variable so that Visual Studio's link program could be found first. See the Windows_installation_of_PETSc for further details.

Step 1. Download and extract source files:

```
wget http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.5.2.tar.gz
tar xvf petsc-3.5.2.tar.gz
cd petsc-3.5.2
```

Step 2. Configure:

```
./configure --with-mpi=0 --with-shared-libraries=0 --with-debugging=0
```

Note: The name of the compiler could be specified by `--with-cc=` and `--with-fc=` options. To download LAPACK during building, append `--download-fblaslapack`. If a fortran compiler is not available, try `--with-fc=0 --download-f2cblaslapack`. In Windows, append `--with-cc='win32fe cl' --with-fc=0 --with-cxx='win32fe cl' --download-f2cblaslapack` option.

Note: Although the MPI option is turned off above, it is okay with the MPI option enabled. For the details of the MPI option or other further details, see installation_instructions_of_PETSc.

Step 3. Build:

```
export PETSC_DIR=$PWD
export PETSC_ARCH=arch-linux2-c-opt
make
```

Note: Change `arch-linux2-c-opt` for `PETSC_ARCH` to the value corresponding to your system as reported by the configure procedure. For example, use `export PETSC_ARCH=arch-mswin-c-opt` in Windows.

2.3 Build BTMCS

Please follow commands below in a shell/terminal (e.g., Bash). They will configure and build BTMCS using GNU Make. The main CMake configuration file (`CMakeLists.txt`) is located in the root directory of the package. This CMake installation is supported on BTMCS version 1.2.1 of higher.

Note: To build in Windows, use CMake GUI with the Win64 (x64) solution platform of Visual Studio. The selected solution platform is needed to match with dependent libraries.

Step 1. Extract source files and create the build directory:

```
tar xvf braintumormodeling_coupledsolver-${version}-source.tar.gz
mkdir braintumormodeling_coupledsolver-${version}-build
cd braintumormodeling_coupledsolver-${version}-build
```

Note: In Windows, use the appropriate zip program (e.g., 7-zip) to extract.

Step 2. Run CMake to configure the build tree:

```
ccmake ../braintumormodeling_coupledsolver-${version}-source
```

In the CMake interface, follow these steps:

2.1. Change `CMAKE_INSTALL_PREFIX` to the folder you want to install BTMCS into. This folder should be outside the `braintumormodeling_coupledsolver-${version}-source` folder. Make sure you have the **write** access to this folder. Change `CMAKE_BUILD_TYPE` (or `CMAKE_CONFIGURATION_TYPES`) to `Release`. Change `PETSC_DIR` and `PETSC_ARCH` to appropriate values. Locate `ITK_DIR` to the appropriate folder if `BUILD_TESTING` set `ON`.

2.2. Keep pressing letter `c` on your keyboard until option `g` is available/displayed on the screen.

2.3. Then press `g` on your keyboard to generate the makefiles and to quit this ccmake window.

Note: In the CMake GUI, `c` and `g` correspond to `Configure` and `Generate` buttons. In Windows, use `/` for the directory seperator in `PETSC_DIR`. Also, set `C:/cygwin/bin/make.exe` for `MAKE_EXECUTABLE` if it could not be found automatically.

Step 3. Build:

```
make
```

Note: In Windows, launch the solution file of Visual Studio, select `Release` in the solution configurations (and confirm `x64` in the soultion platforms), and then perform the rebuild solution.

Step 4. Test (optional):

```
make test
```

Note: In Windows, build the `RUN_TESTS` project. To perform tests, the `BUILD_TESTING` option in the CMake configu-

ration is required to set ON.

In case of failing tests, re-run the tests, but this time by executing CTest directly with the -V option to enable verbose output and redirect the output to a text file:

```
ctest -V >& btmcs-test.log
```

And send the file btmcs-test.log as attachment of the issue report to sbia-software at uphs.upenn.edu.

Step 5. Install:

```
make install
```

Note: In Windows, build the INSTALL project.

Upon the success of the above compilation and build process, BTMCS is installed into the directory specified by the CMAKE_INSTALL_PREFIX (set during build configuration in step 3).

3 Manual

BTMCS does **not** require meshing. It is based on regular grids provided by the image itself (i.e., voxels). It is not necessary, however, to work with the full-resolution original image (e.g., 256x256x198 in Jakob's brain or 256x256x124 in the BLSA brain). On the contrary, it is recommended that you provide a coarser resolution image, by appropriately downsampling the segmented image, to make calculations fast, see the Preprocessing part. Everything in the forward (model) solver runs on top of the PETSc libraries.

3.1 Input Parameters

In the following, the list of parameters/options for this sofware is described. These inputs could be specified in `ForwardSolver.in` file or the command line arguments of the software. The `ForwardSolver.in` file must be in the current directory that executes the software. The prefix "g" here will typically denote global variables (when danger of confusion/re-declaration inside the program might exist).

`-T <double>`

Represents the length of the time interval over which the tumor grows (e.g., number of days).

`-ntimesteps <integer>`

Represents the number of time steps in which the time interval T is being divided for numerical discretization and calculations, respectively.

`-nstore <integer>`

The output (numerical solution) is being saved every other nstore steps; if nstore=ntimesteps, then only the solution at the final time t=T is being saved (recommended if the user is not interested in intermediate results); however, if an adjoint-based optimization method is being employed, then the output of the forward problem needs to be available at *each* time step for the adjoint problem (in which case nstore should be set to 1).

`-cfln <double>`

This is the CFL (Courant-Friedrichs-Levy) number for stability in the numerical scheme of the PDEs; a choice of 0.5 is typically optimal.

`-gstiffwm <double>`

(Relative) White matter stiffness; if the elasticity equation is being scaled by the absolute value of the white matter stiffness (say E=2100 Pa), then the relative stiffness is 1 and everything else in the elasticity equation is scaled accordingly.

`-gstiffgm <double>`

(Relative) Grey matter stiffness.

`-gstiffvent <double>`

(Relative) Ventricle stiffness (here ventricles are modeled as a soft compressible linear elastic material).

`-gstiffcsf <double>`

(Relative) CSF stiffness.

`-gdiffwm <double>`

Tumor cell diffusivity in the white matter.

`-gdiffgm <double>`

Tumor cell diffusivity in the grey matter; we typically assume it's 5 times lower than in the white matter, but having it as a separate parameter allows the user freedom to input whatever value desired.

`-gdiffvent <double>`

Tumor cell diffusivity in the ventricles; should be set to 0. (If desired, the user could also introduce gdiffcsf, for consistency - currently set equal to the ventricles in OriginalMatProp.c).

`-gcompresbrain <double>`

Brain tissue compressibility here assumed nearly incompressible, with a value of 0.45; note that there is no distinction introduced here between different structures (white vs. grey matter, etc.), but if desired, the user can treat these distinctly as well as it's been done in the case of stiffness above.

`-gcompresvent <double>`

Ventricle compressibility (compressible material).

`-gfileInput <string>`

Input *segmented* brain image (with or without tumor), with the following labels assumed: white matter 250, grey matter 150, ventricles 50, CSF 10, tumor 200, falx 20, background 0. Note that it is *not* mandatory to have all the labels 10-250 present, for instance, the segmented image can well be only white matter + ventricles + background (see also the Preprocessing part following).

`-gfileInputLmarkUndef <string>`

If landmarks present, this is a (.txt) file containing the original (undeformed) landmarks (x,y,z n).

`-gfileInputLmarkDef <string>`

If landmarks present, this is a (.txt) file containing the deformed (target) landmarks (x,y,z n).

`-gres_x,gres_y,gres_z <double>`

Physical resolution (voxel size) of the input image at *original resolution* (i.e., if BLSA format, then gres_x=gres_y=0.9375 mm and gres_z=1.5 mm; if Jakob's brain, then gres_x=gres_y= gres_z=1 mm).

`-nblmark <integer>`

If landmarks present, this is the total number of landmarks; otherwise, set it to 0.

`-grho <double>`

Tumor growth rate.

`-gp1, gp2, gse <double>`

Mass-effect parameters.

`-gcinit, gxc, gyc, gzc, gsigsq <double>`

These are all initial tumor parameters; here we assume a Gaussian profile for the initial tumor density (normalized between 0 and 1), with magnitude gcinit, center of coordinates (x,y,z)=(gcx,gcy,gcz) and sigma^2=gsigsq.

`-nsd <integer>`

Number of spatial dimensions. This should always remain fixed to 3 for 3D images.

`-ndimx, ndimy, ndimz <all integers>`

This is a bit tricky here. It must be understood in conjunction with the multigrid/multiresolution approach. These numbers represent the number of *nodes* at the *coarsest* level in the FE (finite element) discretization of the linear elasticity, see below. (The number of nodes in each direction equals the number of elements

+ 1; while strictly from an imaging perspective, only elements <-> voxels are meaningful, in FEA calculations, nodes are also needed; in fact, at the end of a FEA calculation, the displacement per *node* is computed, and from there the displacement per element is computed by linear interpolation).

`-imgx, imgy, imgz <all integers> (added in v1.2.0)`

The image size of input segmented brain image.

`-imgdx, imgdy, imgdz <double> (added in v1.2.0)`

The voxel dimension of input segmented brain image.

`-mgnlevels <integer, with minimum value 1>`

Represents the number of multigrid levels that the user desires. The minimal value of 1 represents the case where no actual multigrid is being used and there is only 1 original grid. If mgnlevels>1, then multigrid is being used; to ensure consistency, the user must be careful to *always* obey the following rule:

segmented input image size x = (ndimx-1) * pow(2,mgnlevels-1);

segmented input image size y = (ndimy-1) * pow(2,mgnlevels-1);

segmented input image size z = (ndimz-1) * pow(2,mgnlevels-1);

For example, if a downsampled image of the segmented Jakob brain with size 64,64,48 (x,y,z in this order) voxels is inputted, and the user selects a mgnlevels=4, then ndimx=9, ndimy=9, ndimz=7. It is the user's responsibility to provide these values consistent with the input image, otherwise, seg faults will occur. It is recommended that ndimx,ndimy,ndimz remain reasonably small, not exceeding say 20, because the algebraic system solver at that (coarsest) level is an exact one, to better precondition the finer levels. More about the segmented input image (size, requirements, recommendations, etc.) in the Preprocessing part following.

`-material_projection_required <integer, possible values 0 or 1>`

If the mgnlevels=1, then this must be set to 0; if mgnlevels>1, then this must be set to 1, meaning that there are multilevels present and the material properties must be projected at each intermediate level.

`-Lx, Ly, Lz <all real numbers, here double format>`

Represent the *physical* image size in each direction x,y and z, respectively; for ensuring consistency, it is preferable that everything is inputted in *SI* measure units, therefore Lx, Ly, Lz are in meters. For example, if the segmented input image is Jakob's brain, then Lx=0.256 m, Ly=0.256 m, Lz=0.198 m (number of voxels x voxel size), while if the segmented input image is the BLSA brain, then Lx=0.24 m, Ly=0.24 m, Lz=0.186 m. However, if in the preprocessing part, the image has been padded before downsampling (for better results), then the corresponding addition must be properly reflected in the new image physical size (also see the Preprocessing part).

The remaining parameters/options are related to the algebraic system solver that runs on top of the PETSc library and it is recommended that they are only changed (with care) if absolutely unavoidable. They have been set to some optimal values that make computations efficient.

Notes regarding choice of the input parameters

- Choice of ntimesteps for stability (i.e., how small the time step should be in order to prevent numerical oscillation and numerical solution from blowing up):

  The stability of *each* step (i.e., advection, diffusion and reaction, respectively) in the fractional time step method used (see references [JMB2008], [SJSC2008], [MICCAI2007]) with respect to the CFL number is ensured; however, there might also be additional constraints on the time step that come from the actual *coupling* of the PDE system being solved; unfortunately, there is no deterministic recipe (closed form formula) for this far, therefore in general, it must be determined by trial and error. The rule of thumb here is as follows: such problems

(coupling errors) appear when the elastic velocity increases substantially (here directly related to increasing the magnitude of the parameter p1); then the time step must be typically decreased to capture these faster changes. However, for the ranges typically tested so far in 3D, I have seen no problems with relatively large time steps, and the velocity (i.e., parameter p1) shouldn't be allowed to grow uncontrollably (we are under the assumption of linear elasticity here to begin with).

- Consistency of measure units in input parameters:

  Everything must be consistent. E.g., if lengths are in meters and time is in days, then diffusivity is in meters^2/day, growth rate (rho) is is 1/day, etc. In the elasticity equation: if elastic properties (stiffness) are in Pa, then so is p1; here, I typically scaled everything by the stiffness (Young's modulus) of the white matter (say Ew=2100 Pa), then a value of p1=5, for instance, corresponds to an actual force of 5*2100 Pa, etc. But this scaling is not mandatory, the user can input actual units as long as they are consistent with everything else.

3.2 Code Structure for the Solver

Main files

The following main files call everything else in place.

- `main_forward_solver.cpp`

  This is the actual solver, given a set of model parameters; used by makefile to build the executable `ForwardSolverDiffusion`.

- `main_optimization.cpp`

  The landmark-based optimization to estimate 4 model parameters: initial tumor density magnitude (gcinit), tumor cell diffusivity in white matter (gdiffwm), tumor growth rate (grho) and mass-effect intensity (gp1); used by makefile to build the executable `LmarkObjective`.

These two executables `ForwardSolverDiffusion` and `LmarkObjective` could be interfaced with a derivative-free optimization library from Sandia: APPSPACK or HOPSPACK.

Initialization (parameter) files

The following files are the ones that set up the forward model solver: parameters and material properties. The actual values of the parameters (options) are being given in the file `ForwardSolver.in` (see also the preprocessing part, for ensuring consistency). The file `ForwardSolver.in` contains a list of all the parameters/options needed by the program; each of these parameters/options can be overwritten/inputed by the user from the keyboard, when calling the executable (for more local flexibility). Otherwise, any parameter/option can be changed by simply changing its corresponding value in the input file `ForwardSolver.in` (no need to recompile the code).

- `initialize.cpp` (`ForwardSolver.in`)
- `InitializeAll.cpp`
- `ReadFiles.cpp`
- `OriginalMatProp.cpp`

Finite Element routines/source files

This code is based on a finite element discretization of linear elasticity, using hexahedral elements (regular grids) and linear shape functions for constructing the approximating numerical solution. Upon discretization (for more details regarding FE discretization see the references in [PMB2007], for instance), this should result into an algebraic system

of equations, with the unknowns being the displacement/deformation field at each grid node (see the Preprocessing part). Note: since we are using linear elasticity for the time being, the resulting algebraic system is linear.

The first 3 files (in the listed order) deal with building the coefficient matrix for this system; note that for preventing memory issues, here we prefer a matrix-free approach, in which the matrix itself needs not be stored, but its action on a vector (see the MatVec function in matrixfree.c).

The files pointer.c and quad.c deal with specific internal FE book-keeping and shape-function calculations.

The files penalized_neumann.c and penalized_neumann_mat.c implement boundary conditions (Neumann, Dirichlet or mixed, with the Dirichlet ones eventually regarded as penalized Neumann) using a finite element with penalty approach (see [PMB2007]). The version we've been using so far is a simple one, which assumes zero displacement at the skull (homogeneous Dirichlet boundary condition, equivalent to imposing a very stiff material outside of the brain, in the background).

Finally, the rhs.c file is responsible for computing the right-hand-side of the system (e.g., the force terms in the elasticity equation).

- `AnJacobian.cpp`
- `Jacobian.cpp`
- `matrixfree.cpp`
- `penalized_neumann.cpp`
- `penalized_neumann_mat.cpp`
- `pointer.cpp`
- `quad.cpp`
- `rhs.cpp`

Multigrid solver

Once the discrete linear algebraic system has been set into place as described in the step above, it is being solved using a multigrid acceleration technique on top of the PETSc library (see [PMB2007]). The main file, calling and interfacing with PETSc functions, is stsdamg.c . The PCShellFiles directory content is solely for the purpose of building adequate preconditioners (PC), in order to speed up convergence. The RPFiles directory content is solely for constructing the restriction/prolongation (RP) operators required by the V-cycle of multigrid methods.

- `stsdamg.cpp`
- PCShellFiles directory
- RPFiles directory

Elasticity file

- `Elasticity.cpp`

Diffusion/Advection and Reaction files

- `Advection.cpp`
- `ConservLaw.cpp`
- `Diffusion.cpp`
- `Reaction.cpp`

Utility files

These are simply collections of tools and "ingredients":

- `Auxiliary.cpp`

  Creates and stores output.

- `ComputeQuant.cpp`

  Computes various quantities, interpolating from nodal values to elemental ones whenever necessary, etc.

- `DeformLandmarks.cpp`

  Is of use in case sets of corresponding landmark pairs exist (e.g., in serial scans of the same subject) and an optimization problem based on landmark matching is to be solved.

- `OriginalLandmarks.cpp`

  See `DeformLandmarks.cpp`.

- `utility.cpp`

  Contains a variety of utility functions, required at various intermediate steps during the FE calculations.

- `global.cpp`

  Contains a (long) list of global variables (not a great idea, but the code was structured such from the very beginning).

Header files

- `common.h`

  Solely for use with the RP operators/routines.

- `RPinclude.h`

  See `common.h`.

- `ConservLaw.h`

  Required by `ConvervLaw.cpp`.

- `function.h`

  Contains an overall list of all the functions employed by the main program.

- `global.h`

  Required by `global.cpp`.

- `PCvar.h`

  Solely for use in preconditioning the algebraic system solver.

- `stsdamg.h`

  Required by `stsdamg.cpp`.

3.3 Preprocessing

All the simulations start from a *segmented* brain image (with or without tumor), with the following labels assumed: white matter 250, grey matter 150, ventricles 50, CSF 10, tumor 200, falx 20, background 0. Note that it is *not* mandatory to have all the labels 10-250 present, for instance, the segmented image can well be only white matter + ventricles + background.

Now, in order to speed up computations with the current version (no adaptivity based on octree-data structures yet incorporated), it is strongly recommended that the user works with a relatively coarse resolution segmented image, by downsampling the original (full resolution) image.

Moreover, in order to be able to use the multigrid solver properly, this downsampled image *must* have a size that can be exactly divided by 2 as many times as the specified number of multigrid levels. This is usually natural for the x and y directions, which are typically 256 in full resolution. However, it can be a problem in the z direction.

Based on extensive numerical experiments, for optimal results, it is recommended to work with a segmented input image at resolution around 64 voxels in each direction. For instance, if the input image is the BLSA segmented template (256x256x124 voxels, with voxel size 0.9375 mm x 0.9375 mm x 1.5 mm), then a good way to obtain the corresponding coarse resolution input image for the `ForwardSolver.in` is as follows:

- First pad the image in the z-direction with background (label 0) to go from 124 to 128 voxels; *note* that in this case, the physical image size parameter Lz in the z direction must be adjusted accordingly (i.e., 128x1.5x10^(-3) m);
- Then downsample the padded image by a factor of 4 in the x and y directions and 2 in the z direction, respectively; then the resulting downsampled image will have a size of 64x64x64 voxels (which will be the actual hexahedral elements in the FE discretization, with a corresponding number of 65x65x65 FE nodes);
- Then a corresponding optimal choice of the parameters ndimx,ndimy,ndimz and mgnlevels is as follows: ndimx=ndimy=ndimz=9, mgnlevels=4.

If the input image is Jakob's brain segmented template (256x256x198 voxels, with voxel size 1 mm x 1 mm x 1 mm), then a good way to obtain the corresponding coarse resolution input image for the `ForwardSolver.in` is as follows:

- First crop the image in the z-direction (background, label 0) to go from 198 to 192 voxels; *note* that in this case, the physical image size parameter Lz in the z direction must be adjusted accordingly (i.e., 192x1.0x10^(-3) m);
- Then downsample the padded image by a factor of 4 in the x, y and z directions, respectively; then the resulting downsampled image will have a size of 64x64x48 voxels (which will be the actual hexahedral elements in the FE discretization, with a corresponding number of 65x65x49 FE nodes);
- Then a corresponding optimal choice of the parameters ndimx,ndimy,ndimz and mgnlevels is as follows: ndimx=ndimy=9, ndimz=7, mgnlevels=4.

3.4 Output/Post-processing

The program can save a variety of output (tumor density, deformation fields, material properties, etc., see the file Auxiliary.c). The quantities of interest so far have been: the total (cumulative) deformation field and the (normalized) tumor density at the end of the simulation. These are currently saved as floats. The deformation field, x,y,z order, with the nomenclature DeformationField+suffix(time step number), e.g. `DeformationField.001.mhd`, in case only the final results are of interest, set nstore=ntimesteps in `ForwardSolver.in`; then the output file `DeformationField.001.mhd` represents the *total* (cumulated) displacement/deformation (i.e., at time t=T). There is *no need here to concatenate intermediate deformation fields* (like it is being done for the purely mechanical solver ElasticSolverPLE), for simplicity, the total deformation (i.e., trajectory) is being updated after each time step inside the program (based on the velocity computed by the model).

Similarly, if only the tumor density maps at the end of the simulation are of interest, then set nstore=ntimesteps in `ForwardSolver.in`; then the (float) file `TumorDensity.001.mhd` represents the final tumor density maps (i.e., at time t=T).

- The displacements resulted from FE calculations are saved *per FE node*; thus, if the downsampled input image size was 64x64x64 , then the corresponding size of the displacement field is 65x65x65 x 3 (vectorial field with 3 components xyz at each grid node, in the x,y,z order).
- The displacements resulted from FE calculations are in *physical dimensions*.
- The post-processing utilities `ResampleDeformationField` and `ResampleImage` included in GLISTR or PORTR software could be used to take such deformation fields resulted from FE calculations and construct the actual deformation fields to be used for image analysis, by resamling to the original image size and "voxelizing" them (i.e., if the padded original image size was say 256x256x128, then the resampled/voxelized deformation field should have a size of 256x256x128 x 3, and it should be in *voxels* instead of physical units).
- The tumor density here it is saved *per voxel* (by linear interpolation from the nodal values), for convenience, for further resampling/visualization in MIPAV (needs to be overlaid on top of the deformed brain image). For the moment, it's been resampled to the original image size (trilinear interpolation) in MIPAV, at the time of visualization.

4 Changelog

Version 1.2.1 (Nov 14, 2014)

- Improved the source code to be compatible with PETSc 2.2.1 - 3.5.2.
- CMake is supported.

Version 1.2.0 (Nov 5, 2014)

- Works with GLISTR 3.x and PORTR 1.x.
- Started to support **Windows**.
- Getting image sizes from the input file to produce correct dimension information in the output deformation field.
- Fix header information of output images.
- Output density and deformation field only.
- Minor bug fixes on the calculation of entire volume size.

Version 1.1.0 (May 9, 2012)

- Works with GLISTR 2.x
- Fixed build with BOPT=O.
- Fixed cumulated deformation field.
- Fixed main loop of ForwardSolverDiffusion to properly account for -ntimesteps and -nstore.

Version 1.0.1 (Apr 27, 2012)

- Fix indices used for paramforce array.
- Compile using optimization level 3 to speed things up.

Version 1.0.0 (June 10, 2011)

- First release

5 Publications

Please cite [JMB2008] when you used BTMCS in your research:

[JMB2008]
[SJSC2008]
[MICCAI2007]
[PMB2007]
[MICCAI2014]
[TMI2014]

6 People

Advisor

- Christos Davatzikos ✉
- George Biros

Software Authors

- Cosmina Hogea
  - Initiated the project.
  - Developed the original version of software.
- Dongjin Kwon ✉
  - Developed the current version of software.
  - Improved the software to be compatible with newer versions of PETSc.

References

[JMB2008] C. Hogea, C. Davatzikos and G. Biros, An image-driven parameter estimation problem for a glioma growth model with mass effects, J. Math Biol., 56(6): 793-825 (2008)

[SJSC2008] C. Hogea, C. Davatzikos and G. Biros, Brain-Tumor Interaction Biophysical Models for Medical Image Registration, SIAM J. Scientific Computing 30(6): 3050-3072 (2008)

[MICCAI2007] C. Hogea, C. Davatzikos and G. Biros, Modeling Glioma Growth and Mass Effect in 3D MR Images of the Brain, In: Proc. MICCAI (1): 642-650 (2007)

[PMB2007] C. Hogea, G. Biros, F. Abraham and C. Davatzikos, A robust framework for soft tissue simulations with application to modeling brain tumor mass effect in 3D MR images, Physics in Medicine and Biology, (2007)

[MICCAI2014] D. Kwon, R.T. Shinohara, H. Akbari, C. Davatzikos, Combining Generative Models for Multifocal Glioma Segmentation and Registration, In: Proc. MICCAI (1): 763-770 (2014)

[TMI2014] D. Kwon, M. Niethammer, H. Akbari, M. Bilello, C. Davatzikos, and K.M. Pohl, PORTR: Pre-Operative and Post-Recurrence Brain Tumor Registration, IEEE Trans. Med. Imaging 33(3): 651-667 (2014)