Supplementary material

Numerical investigations on the BSWP model.

Cusseddu and Madzvamuse, 2022

1. Information about the numerical code

The attached code solves the BSWP model using the bulk-surface finite element method proposed in [1]. Such method allows us to find the solutions a and b at each time point in three consecutive steps, given in the equations (41)-(43) contained in [1]. At each time step, we solve three linear systems for finding, respectively, a predictor for the surface component a, the bulk component b and, finally, a corrector for a. The code is constituted by two separated python files: the main one called `BSWPmodel.py` and the file `parameters.py`, which reports the parameters used. In the latter, it is possible, for the user, to select the mesh from the mesh folder. The user is also able to choose to solve either the BSWP model or the reduced model. Both choices are passed by the user through keyboard, while the code runs. Moreover, initial conditions are defined in `parameters.py`, as well as a string variable called `filename` which reports all parameters and it will be used to later identify the files produced by the code. In this file we also define the numerical method used for solving the linear systems and a set of time points at which the solution will be exported to pvd files [1]. Our code makes use of FEniCS [2][2].

The bulk-surface finite element method is coded in the main file `BSWPmodel`. In the following, we discuss some points of the code, which might be helpful for understanding the approach we use. The notation of the code follows the one of Section 5 in [1].

1. The code imports the mesh indicated by the user. It then extracts the boundary mesh `boundary_mesh` using the function `BoundaryMesh`. For three-dimensional domains, the boundary mesh is constituted by the faces of the tetrahedra composing the boundary of the bulk domain. Once we have two different meshes, the two function spaces `V_Bulk` and `V_Boundary` are defined.

2. The first system to solve is given in equation (41) of [1] whose solution constitutes a prediction for a_h at time t. For calculating $\tilde{a}_h(t_n, \mathbf{x})$ at time t_n, in the temporal discretisation, we consider the diffusion term implicitly, while the reaction term explicitly, i.e. as a function of a and b at the previous time t_{n-1}. Since both the unknown $\tilde{a}_h(t_n, \mathbf{x})$ and the known $a_h(t_{n-1}, \mathbf{x})$ belong to the function space `V_Boundary`, while $b(t_{n-1}, \mathbf{x})$ belongs to the function space `V_Bulk`, for compatibility issues, we consider the restriction b_Γ of b over `V_Boundary`, using the function `Interpolate`. In such a way, the system is entirely constituted

[1] Use the software Paraview to visualise the pvd files, see https://www.paraview.org

[2] About FEniCS, we recommend the introductory tutorial [3]. See also the website https://fenicsproject.org.

by functions in `V_Boundary`.

3. In the second system, which is given in equation (42) of [1], the unknown function $b(t_n, \mathbf{x})$ belongs to `V_Bulk`. Similarly to what we did before, for compatibility issues, we want the system to be entirely constituted by functions of the same function space `V_Bulk`. Hence, we define a sort of extension $\tilde{a}_\Omega$ of $\tilde{a}$ from `V_Boundary` to `V_Bulk`. In particular, $\tilde{a}_\Omega$ coincides with $\tilde{a}$ over Γ and it is zero elsewhere. This does not modify the nature of the problem, since the surface component enters in the system only through the boundary condition for b. Initially, by setting `a_bulk_p = Function(V_Bulk)`, $\tilde{a}_\Omega$ is zero everywhere. We want to replace only its values on `boundary_mesh`. For doing so, since vertex order is different between boundary and bulk mesh, as well as between the two function spaces, we need to create a map. This is done by using the functions `vertex_to_dof_map` and `dof_to_vertex_map` which return a map between the mesh vertices and the degrees of freedom of the function space and viceversa. For instance, the command `array = a_bulk_p.vector().get_local()` stores in `array` the values of `a_bulk_p` with the order defined by `V_Bulk`. The command `array[vBulk_dof[vboundary_v[dofb_vboundary]]] = a_p.vector().get_local()` only replaces the values of `a_bulk_p` at the boundary with the corresponding values of `a_p` from `V_Boundary`. Since both the predicted $\tilde{a}$ and a at time t_{n-1} are used to calculate b at time t, the same procedure applies also to $a_h(t_{n-1}, \mathbf{x})$.

4. The solutions are saved as pvd files, which are possible to visualise with the software Paraview[3].

References

[1] D. Cusseddu, L. Edelstein-Keshet, J. A. Mackenzie, S. Portet, A. Madzvamuse, A coupled bulk-surface model for cell polarisation, Journal of theoretical biology 481 (2019) 119–135. doi:10.1016/j.jtbi.2018.09.008.

[2] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, G. N. Wells, The fenics project version 1.5, Archive of Numerical Software 3 (100) (2015).

[3] H. P. Langtangen, A. Logg, Solving PDEs in python: the FEniCS tutorial I, Springer Nature, 2017.

[3] https://www.paraview.org