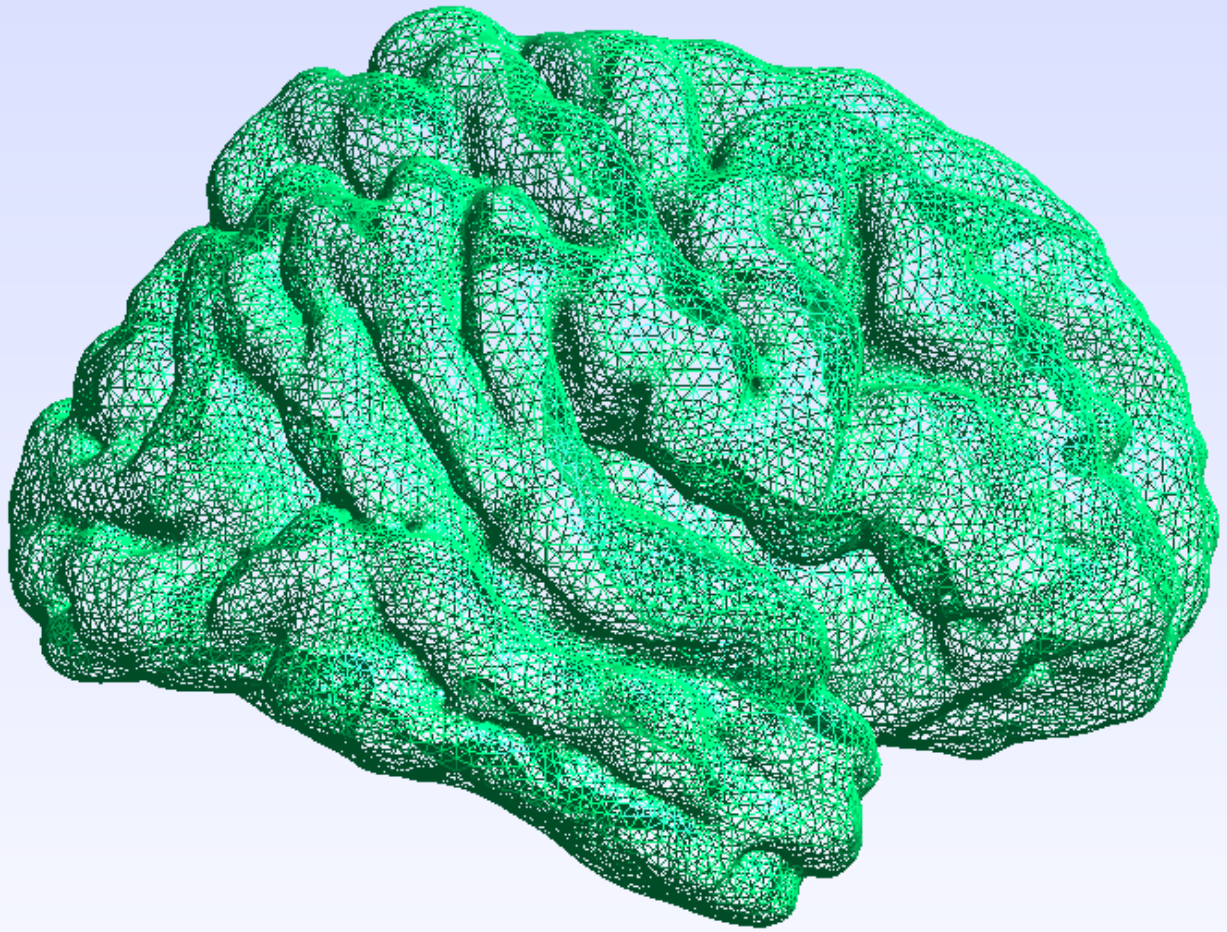


NUMERICAL METHODS FOR PARTIAL DIFFERENTIAL EQUATIONS PROJECT

PRION DISEASE

Valentina Moretti
Mattia Colbertaldo
Roberto Neglia

« A physics-based model explains the prion-like features of neurodegeneration in Alzheimer's disease, Parkinson's disease, and amyotrophic lateral sclerosis »

Johannes Weickenmeier , Mathias Jucker , Alain Goriely , Ellen Kuhl

- Prion disease is characterized by a chain reaction in which infectious misfolded proteins force native proteins into a similar pathogenic structure.
- Recent studies have reinforced the hypothesis that the prion paradigm – the templated growth and spreading of misfolded proteins – could help explain the progression of a variety of neurodegenerative disorders.
- The paper shows that a physics-based reaction-diffusion model can explain the growth and spreading of misfolded protein in Alzheimer's disease, Parkinson's disease, and amyotrophic lateral sclerosis
- To characterize the progression of misfolded proteins across the brain, we combine the classical Fisher–Kolmogorov equation for population dynamics with an anisotropic diffusion model and simulate misfolding across a sagittal section and across the entire brain.

Discretization

- The weak formulation is obtained by choosing a suitable function space V, taking a function v in V, and multiplying v to the **non-linear time-dependent diffusion reaction equation** of Fisher-Kolmogorov, and integrating this over the whole domain *B* and including homogeneous **Neumann** boundary conditions, $\nabla c \cdot n = 0$.

- We first **discretized in space** by introducing the finite element space $X^r{}_h(\Omega)$, obtained the semi-discrete form of the formulation, then **discretized in time** using **implicit Euler**.

- We then linearized the obtained equation using the **Newton** method for systems of ODE, by computing the Fréchet derivative of the residual equation.

$$\frac{\mathrm{d}c}{\mathrm{d}t} = \mathrm{Div}(\boldsymbol{D} \cdot \nabla c) + \alpha\, c\,[1 - c]\,.$$

$$\mathrm{R}_I = \mathop{\mathrm{A}}_{e=1}^{n_{\mathrm{el}}} \int_{\mathcal{B}_e} N_i \frac{1}{\Delta t}\,[c - c_{\mathrm{n}}] + \nabla N_i \cdot \boldsymbol{D} \cdot \nabla c - N_i\,\alpha\, c\,[1 - c]\,\mathrm{d}V \doteq 0_I\,,$$

$$\mathrm{K}_{IJ} = \frac{\mathrm{d}\mathrm{R}_I}{\mathrm{d}c_J} = \mathop{\mathrm{A}}_{e=1}^{n_{\mathrm{el}}} \int_{\mathcal{B}_e} N_i \frac{1}{\Delta t} N_j + \nabla N_i \cdot \boldsymbol{D} \cdot \nabla N_j - N_i\,\alpha\,[1 - 2\,c]\,N_j\,\mathrm{d}V$$

Discretization

$$\frac{\partial c}{\partial t} - \nabla \cdot (\mathrm{D}\,\nabla c) - \alpha c(1-c) = 0 \qquad \text{in} \quad \Omega \times (0,T)$$

$$\nabla c \cdot \hat{\mathrm{n}} = 0 \qquad \text{on } \partial\Omega \times (0,T)$$

$$c = c_0 \qquad \text{in} \quad \Omega \times \{0\}$$

Residual form of the weak formulation

$$\forall t \in (0,T)\ \text{find } c \in V = H^1(\Omega)\ \text{s.t.}$$

$$R(c)(v) = \int_\Omega \left(\frac{\partial c}{\partial t} v + \nabla v\, \mathrm{D}\, \nabla c - \alpha c(1-c) v \right) d\mathrm{x} = 0$$

$$\forall v \in V \text{ and } c(\mathrm{x},0) = c_0$$

Space discretization

$$\forall t \in (0,T)\ \text{find } c_h \in V_h = H^1(\Omega) \cap X_h^r(\Omega)\ \text{s.t.}$$

$$R(c_h)(v_h) = \int_\Omega \left(\frac{dc_h}{dt} v_h + \nabla v_h\, \mathrm{D}\, \nabla c_h - \alpha c_h(1-c_h) v_h \right) d\mathrm{x} = 0$$

$$\forall v_h \in V_h \text{ and } c_h(\mathrm{x},0) = c_{0,h}$$

Time discretization (implicit Euler)

$$\text{given } \Delta t = t_{n+1} - t_n,\ c_h^n = c_h(t_n),\ \text{find } c_h^{n+1} \in V_h\ \text{s.t.}$$

$$R^{n+1}\left(c_h^{n+1}\right)(v_h) = \int_\Omega \left(\frac{c_h^{n+1} - c_h^n}{\Delta t} v_h + \nabla v_h\, \mathrm{D}\, \nabla c_h^{n+1} - \alpha c_h^{n+1}\left(1 - c_h^{n+1}\right) v_h \right) d\mathrm{x} = 0$$

$$\forall v_h \in V_h \text{ and } \forall n = 0, 1, \ldots, N_T - 1$$

Newton method

Fréchet derivative

$$a\left(c_h^{n+1}\right)(\delta_h,\, v_h) = \lim_{h \to 0} \frac{1}{h}\left[R^{n+1}\left(c_h^{n+1} + \delta_h h\right)(v_h) - R^{n+1}\left(c_h^{n+1}\right)(v_h) \right]$$

$$= \ldots$$

$$= \int_\Omega \left(\delta_h \frac{1}{\Delta t} v_h + \nabla \delta_h\, \mathrm{D}\, \nabla v_h - \delta_h \alpha (1 - 2c_h^{n+1}) v_h \right) d\mathrm{x}$$

· *solve*

$$a\left(c_h^{n+1,(k)}\right)\left(\delta_h^{(k)},\, v_h\right) = -R\left(c_h^{n+1,(k)}\right)(v_h) \quad \forall v_h \in V_h$$

by finding $\delta_h^{(k)}$

· *update*

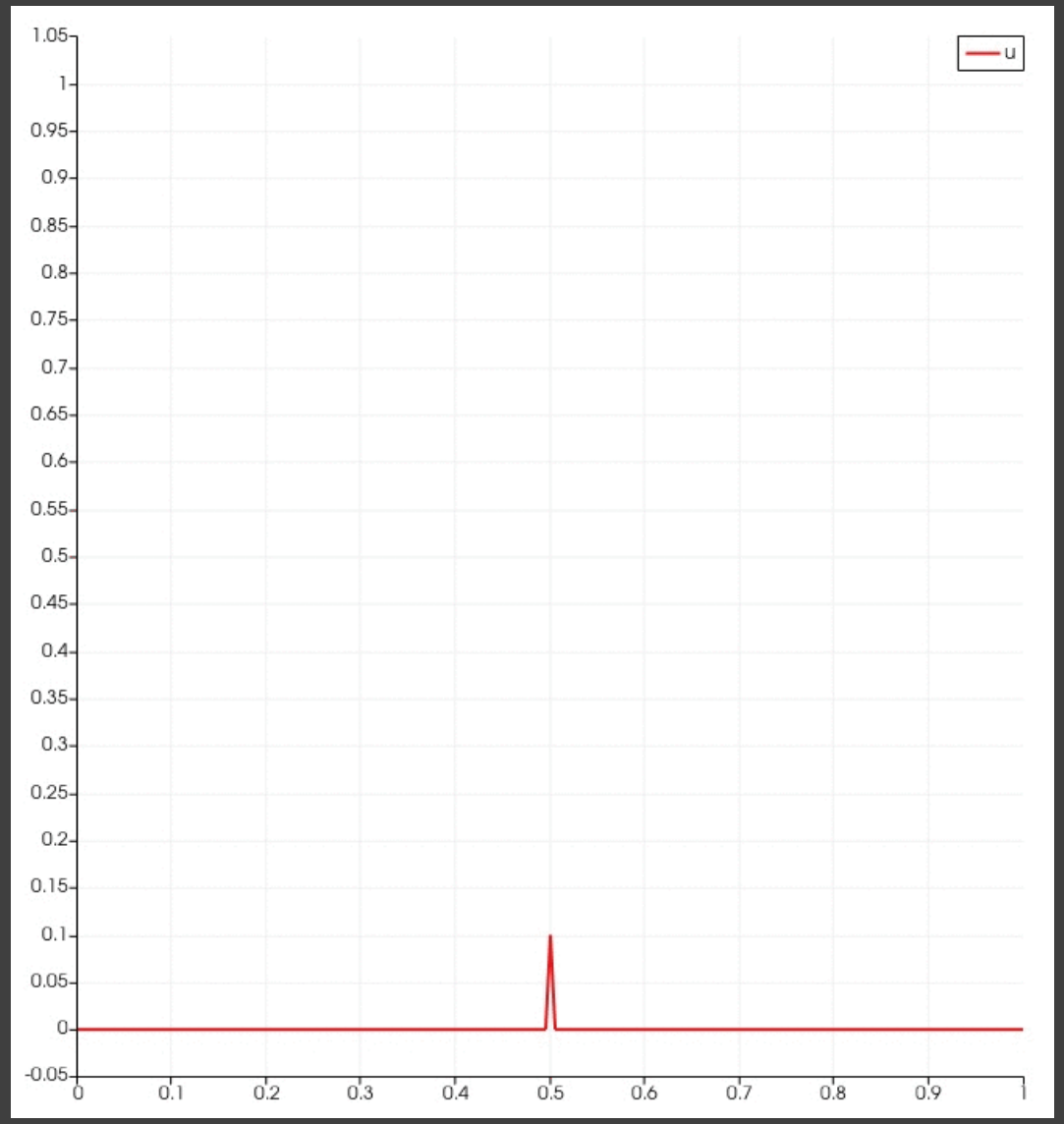
$$c_h^{n+1,(k+1)} = c_h^{n+1,(k)} + \delta_h^{(k)}$$

Mathematical interpretation

- Div(D ∇c) = diffusion term that distribute c in the space
- α c (1-c) = reaction term = generation of c = concentration interacts with itself.
- The seeding region *B* characterizes the regional onset of neurodegeneration.

- The **growth rate α** characterizes the local increase of the misfolded protein concentration
- The extracellular diffusion d^{ext} characterizes the **isotropic spreading** of misfolded protein through the extracellular space
- The axonal transport d^{axn} characterizes the **anisotropic spreading** of misfolded protein through intracellular transport
- The **axonal orientation n** characterizes the anatomic pathway of fast spreading

the concentration, and $\boldsymbol{D}$ its spreading. The Fisher–Kolmogorov equation has two steady state solutions, an unstable steady state at $c = 0$ and a stable steady state at $c = 1$. This implies that once misfolded protein is present anywhere in the brain, $c > 0$, the concentration will always be repelled from the benign state, $c = 0$, and attracted to the misfolded state, $c = 1$. We can model the seeding of misfolded protein via non-homogeneous initial conditions, $c_0 > 0$, in specific brain regions. Once



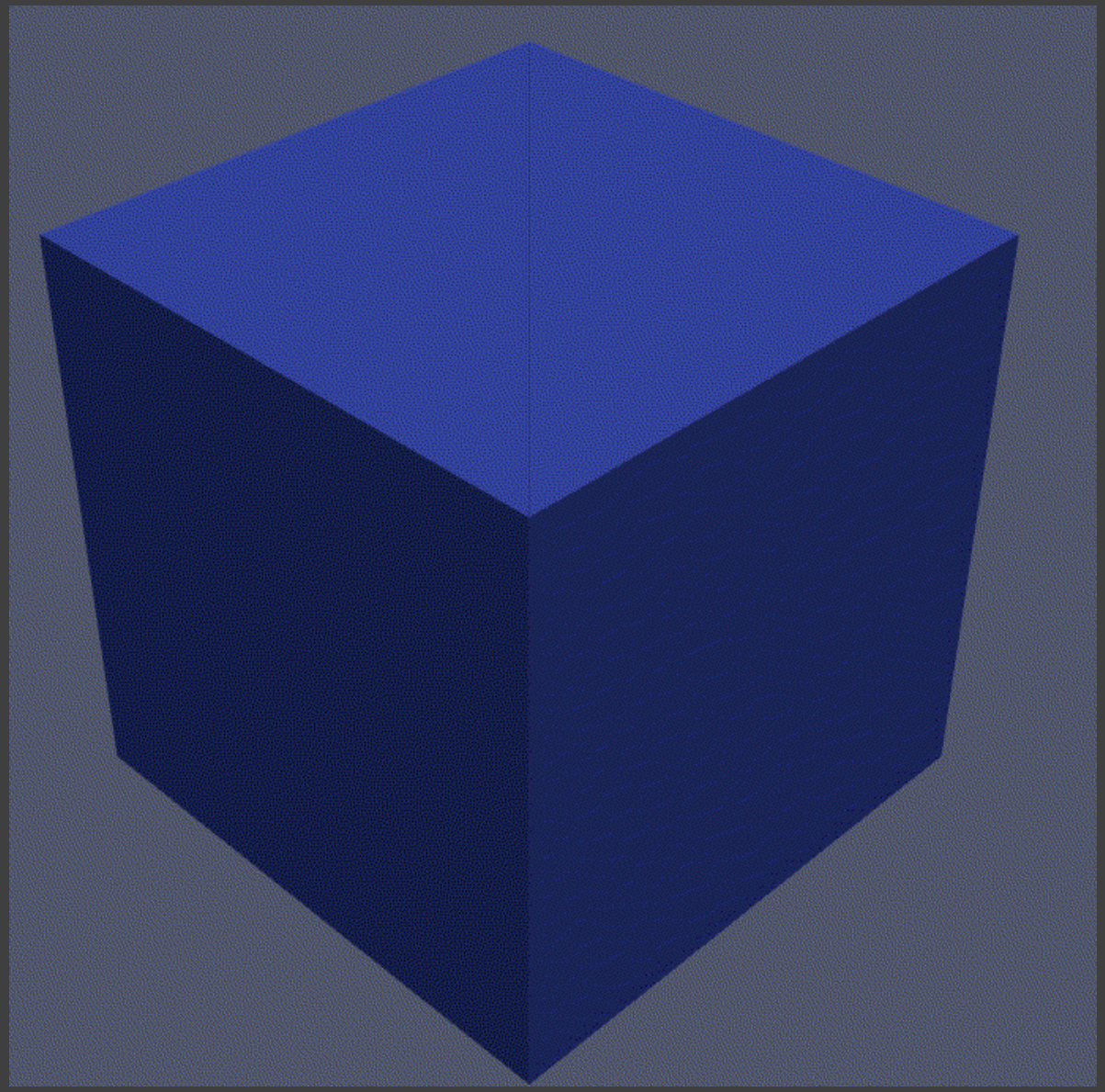
1D case

- MESH : The domain chosen is the interval $\Omega = (0,1)$, and the mesh is obtained by subdividing into N+1 elements of size $h = 1.0/(N+1)$, either by using deal.II functions or external meshing tools (like GMSH). We used deal.II GridGenerator class for simplicity, with **N+1 = 200** elements.

- The parameters of the equation chosen are
- $d^{ext} = 0.0001$ and $\alpha = 1$.

- The initial condition is given inside the range (xL, xR), with xL = 0.45 and xR = 0.55 as a gaussian function centered in xC = 0.5.

- As can be seen in the gif created, the code implemented exactly computes the spreading of the concentration from the center of the domain towards the whole domain.



3D case

MESH : The domain chosen at this step is the unit cube $\Omega = (0,1)^3$, and the mesh is obtained by introducing tetrahedrons, either with deal.II functions (or with external meshing tools).

Observations and critical analysis: (1)

- We have initially put the initial condition u_0=0.1 in the central point of the domain (in 1D in x= 0.5 and in the 3D case in x=y=z=0.5).

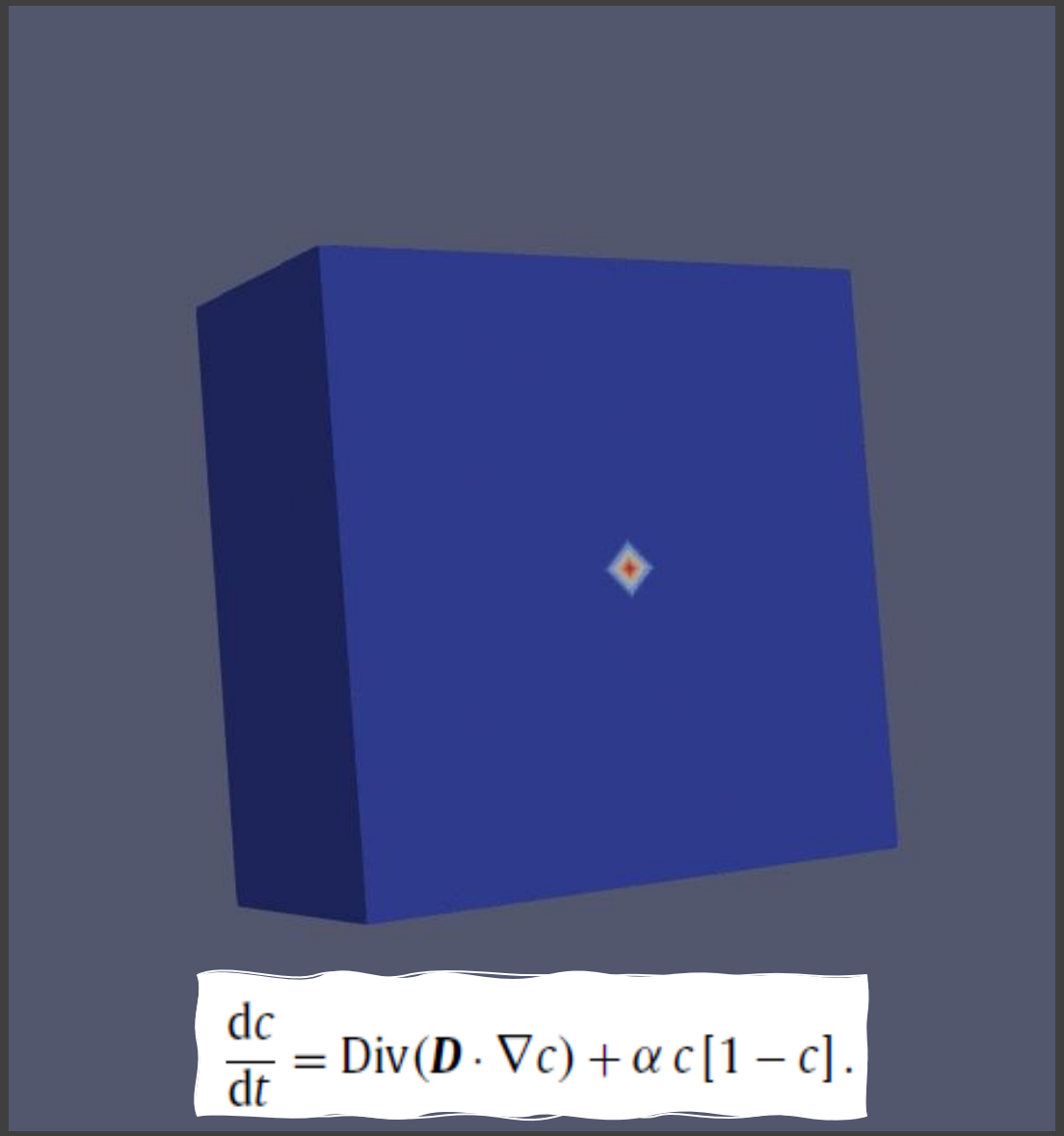
- What it was observed was that for N=partition size (that generates N+1 mesh elements) = 20 (even number) we had null concentration in the whole domain in all the timestamps. For N=19 (odd number) this was not happening and the initial condition was seen in t=0 and in the subsequent timestamps it was visible its spreading.

- A critical analysis made us discover the inaccuracy. The initial value, if defined in an unique pointer, in the odd case, even if the mesh elements are not fine, was seen by the interpolator. On the opposite case, it was not seen and no initial value vas considered. This in agreement with the mathematical theory we are working with: c is a function in the Lebesgue space and the functions in the Lebesgue space are defined at least “up to a set of measure-zero interval”. This is the reason why the initial value was not reported on an unique point in the space, as written in the paper, but is a gaussian defined in a small interval.

Observations and critical analysis: (1)

```
// Function for initial conditions.
class FunctionU0 : public Function<dim> {
public:
  virtual double
  value(const Point<dim> &p, const unsigned int /*component*/ = 0) const override {
    // for the cube mesh
    if (p[0] > 0.4 && p[0] < 0.6 && p[1] > 0.4 && p[1] < 0.6 && p[2] > 0.4 &&
        p[2] < 0.6)
      return 0.1 *
             std::exp(-std::pow(30 * (p[0] - 0.5), 2) - std::pow(30 * (p[1] - 0.5), 2) -
                      std::pow(30 * (p[2] - 0.5), 2));

    // for the brain mesh
    if (p[0] > 49 && p[0] < 51 && p[1] > 79 && p[1] < 81 && p[2] > 69 && p[2] < 71)
      return 1.0 *
             std::exp(-std::pow(2 * (p[0] - 50), 2) - std::pow(2 * (p[1] - 80), 2) -
                      std::pow(2 * (p[2] - 70), 2));

    return 0.0;
  }
};
```

Observations and critical analysis: (2)



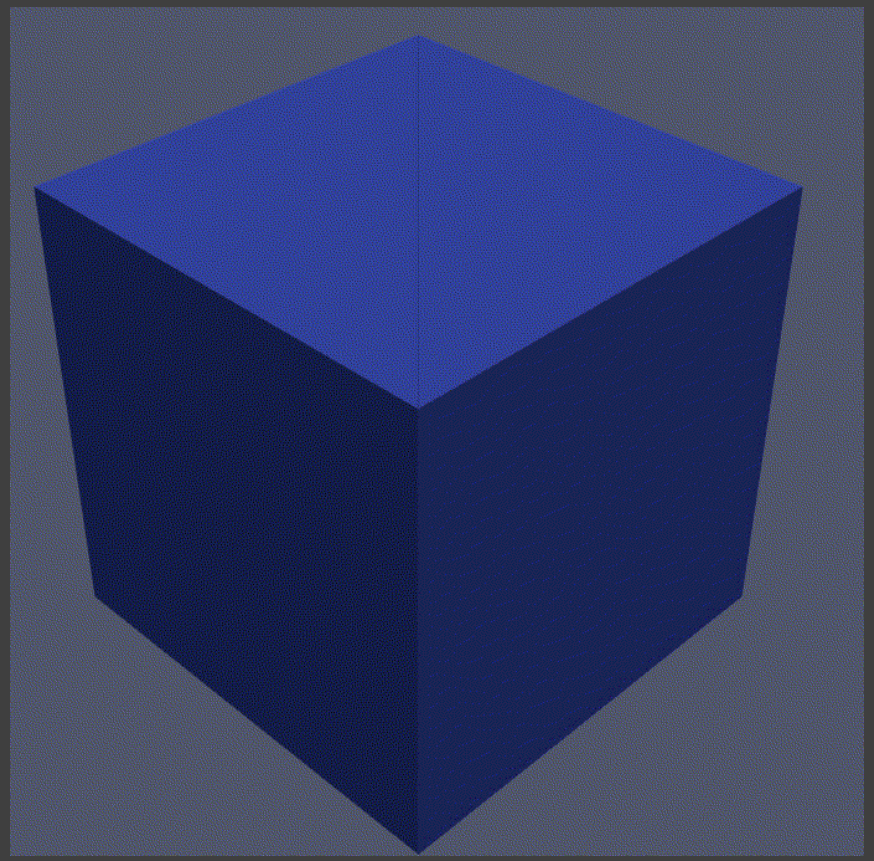
$$\frac{dc}{dt} = \mathrm{Div}(\boldsymbol{D} \cdot \nabla c) + \alpha\, c\,[1 - c]\,.$$

Our first results had a spreading that had an oval shape in the cubic mesh. This happened because we took $d^{axn} \neq 0$. With $d^{axn} = 0$ we obtain a spherical spreading.

What we could not explain of our first results was the appearance of some diffusions detached from the central spreading and divergence of our newton solver at some iteration in the time stamps' subdivision. For mathematical formulations as the one in the paper if the component $\alpha\, c\,(1\text{-}c)$ is much larger than the other $\mathrm{Div}(D\,\nabla c)$ some oscillations can occur.
$\mathrm{Div}(D\,\nabla c)$ scales as $1/h^2$ (second derivative).
So, the diffusion-reaction problem, the elimination of the oscillations is linked to $1/h^2$ and if h is big, $\mathrm{Div}(D\,\nabla c)$ is a small term.

In this way we obtain a pure reaction problem which is unstable.

Observations and critical analysis: (2)

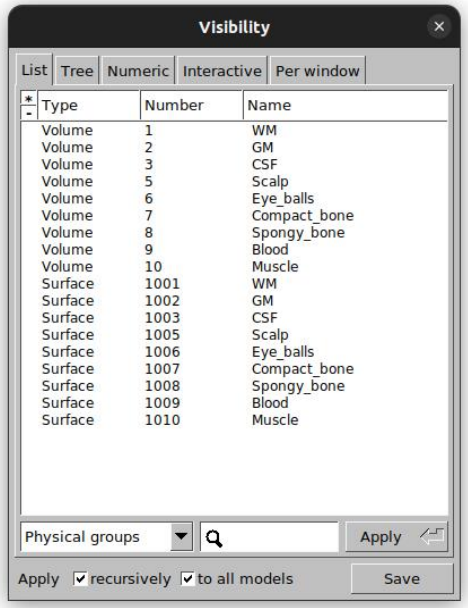


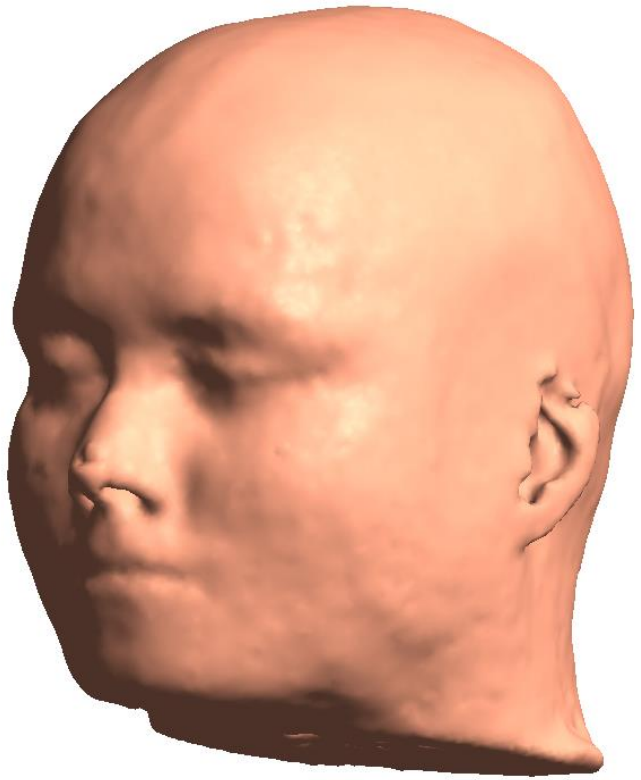
On the cubic mesh: We tried to increase the value of d^{ext} of a factor of ten and the oscillations eventually disappeared (keeping N small).

The same was done on the brain mesh: but a very big d^{ext} was needed

A better solution would be to keep d^{ext} unchanged and reduce h by increasing N.
We can do that by refine (divide a triangle in for) by splitting or with remeshing (more flexible).

Observations and critical analysis: (3)



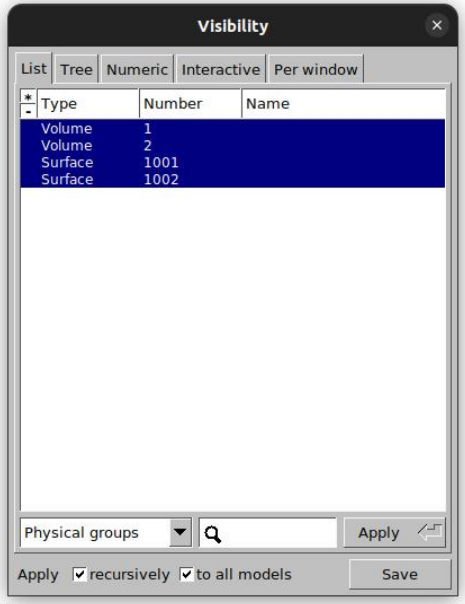


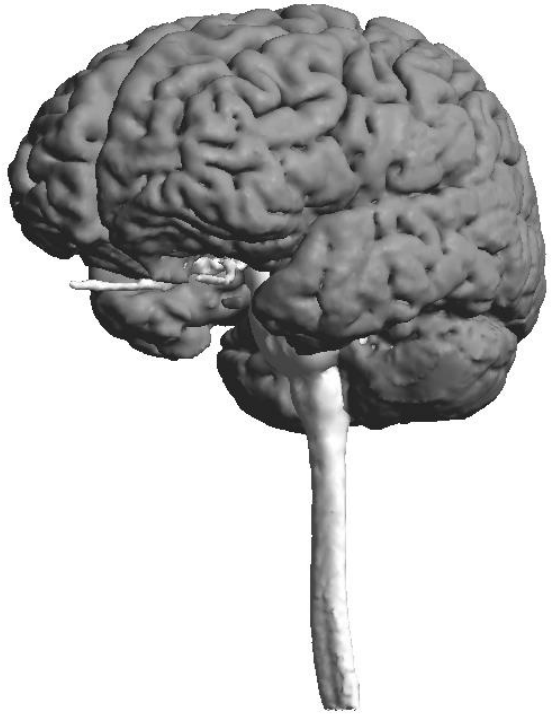
Brain mesh

First, we tried to obtain a brain mesh from MRI images taken from OpenNeuro.
We used SimNIBS (Simulation of Non-Invasive Brain Stimulation) software package to do this.

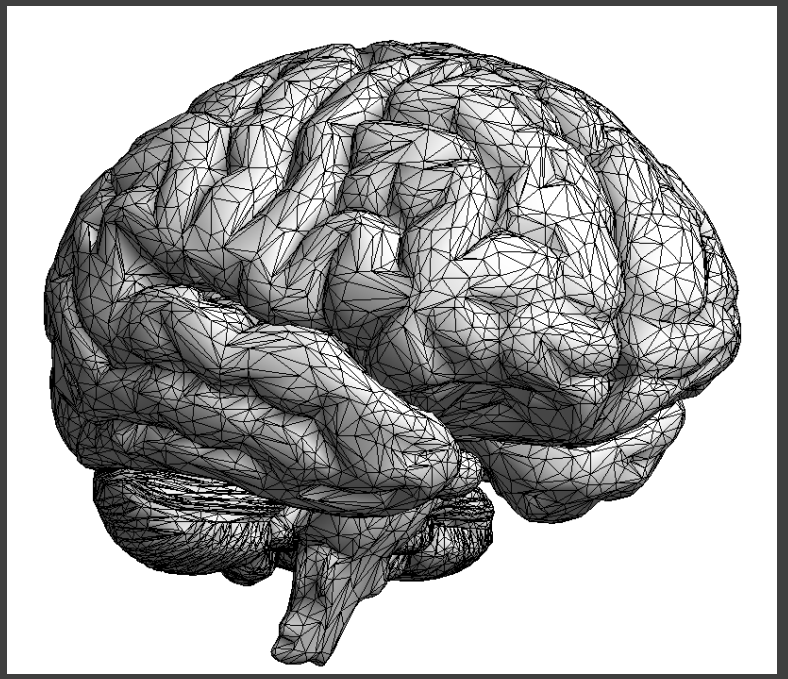
The resulting mesh was made of multiple volumes, which deal.II couldn't manage.

Observations and critical analysis: (3)



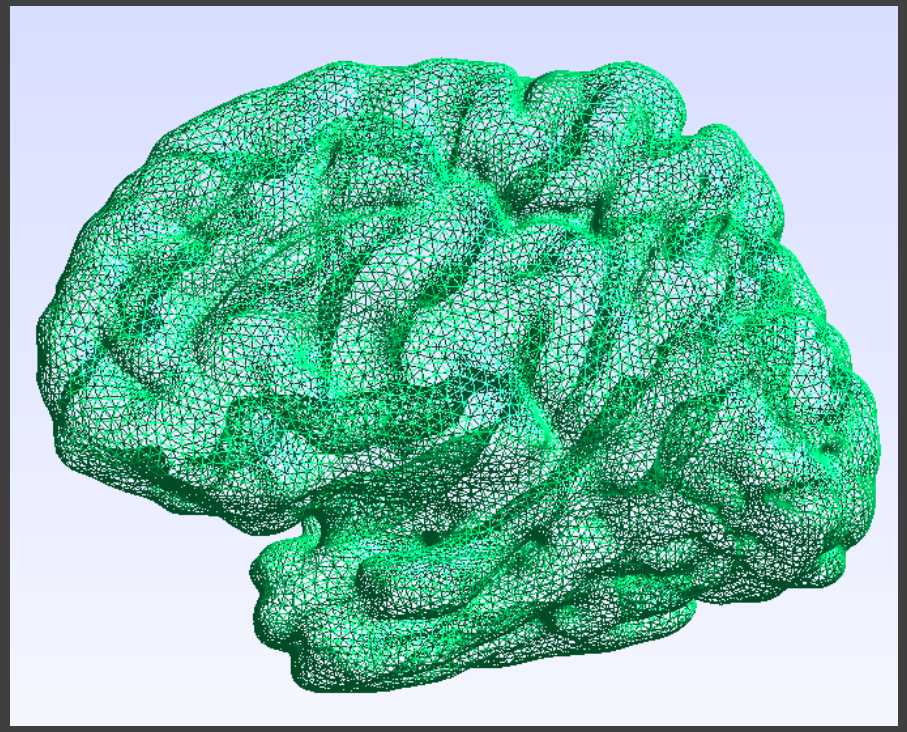


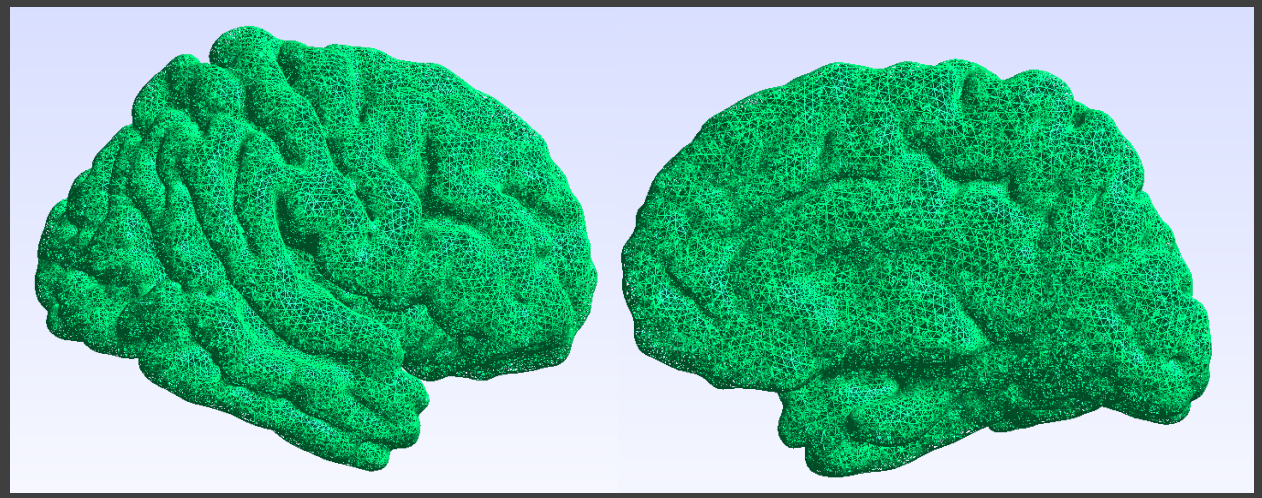
Next, we tried with some STLs found online. The problem in this case was the creation of 3D elements with Gmsh.



```
Info    : Done tetrahedrizing 15008 nodes (Wall 0.233498s, CPU 0.222516s)
Info    : Reconstructing mesh...
Info    :  - Creating surface mesh
Info    : Found two duplicated facets.
Info    :   1st: [1450, 1584, 14997] #1
Info    :   2nd: [1450, 1584, 14997] #1
Error   : Invalid boundary mesh (overlapping facets) on surface 1 surface 1
```

Observations and critical analysis: (3)

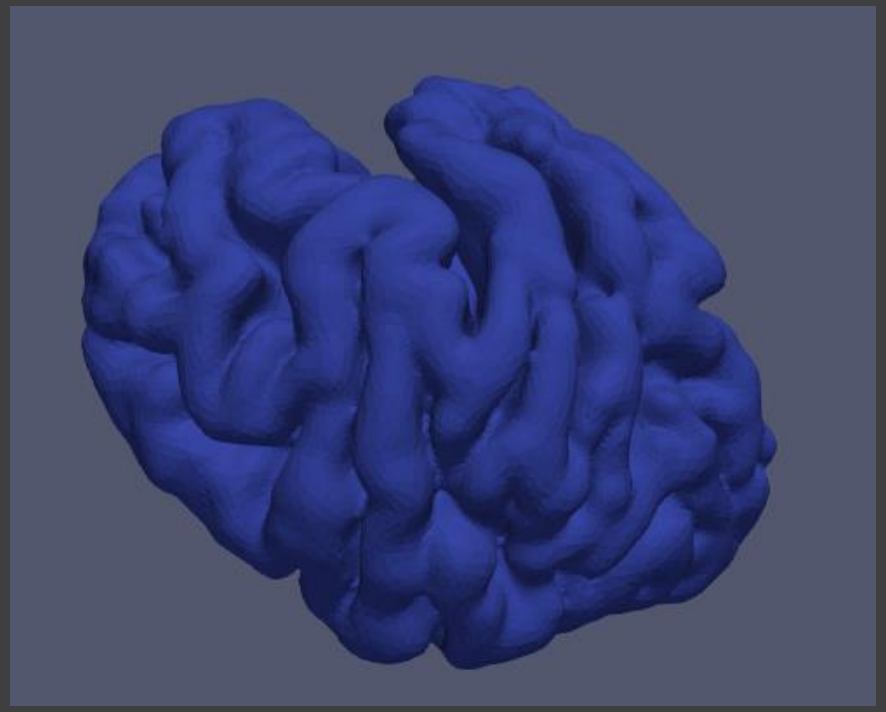




Brain mesh

At the end, a more uniform STL of the right hemisphere of a brain was given to us.
From this, we created the 3D mesh elements with Gmsh, and what we obtained was a mesh with 634'472 tetrahedrons and 111'951 degrees of freedom, which became 4'675'874 tetrahedrons and 787'888 degrees of freedom after a refinement of the mesh.

Observations and critical analysis: (3)



Finer brain mesh

```
Initializing the mesh
  Number of elements = 634472
-----------------------------------------
Initializing the finite element space
  Degree                     = 1
  DoFs per cell              = 4
  Quadrature points per cell = 4
-----------------------------------------
Initializing the DoF handler
  Number of DoFs = 111951
```

Gmsh --> refine by splitting

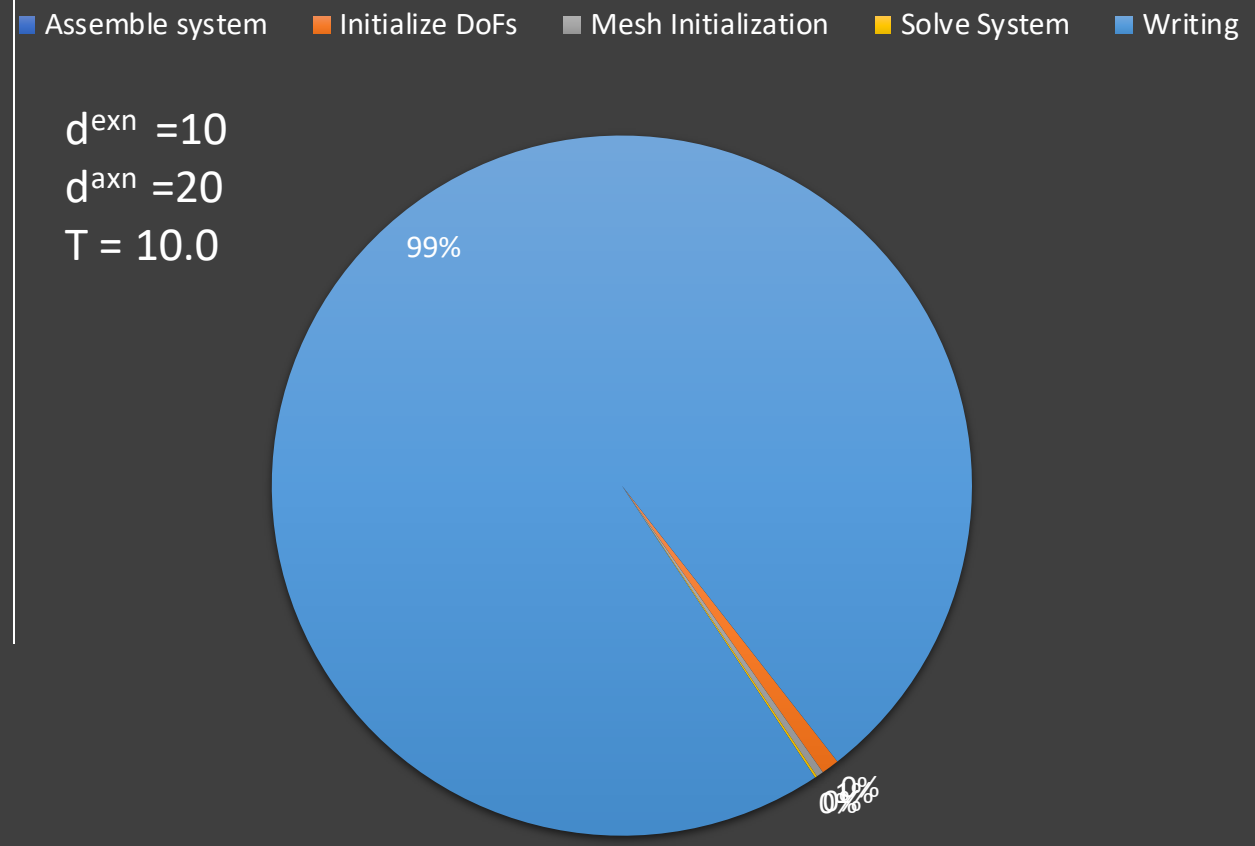
```
Initializing the mesh
  Number of elements = 4675874
-----------------------------------------
Initializing the finite element space
  Degree                     = 1
  DoFs per cell              = 4
  Quadrature points per cell = 4
-----------------------------------------
Initializing the DoF handler
  Number of DoFs = 787888
-----------------------------------------
```

Observations and critical analysis: (4)

```
while (time < T - 0.5 * deltat) {
  time += deltat;
  ++time_step;

  // Store the old solution, so that it is available for assembly.
  solution_old = solution;

  pcout << "n = " << std::setw(3) << time_step << ", t = " << std::setw(5)
        << std::fixed << time << std::endl;

  // At every time step, we invoke Newton's method to solve the non-linear
  // problem.
  solve_newton();

  timer_output.enter_subsection("Writing");
  output(time_step, time);
  timer_output.leave_subsection();

  pcout << std::endl;
}
```

```
+---------------------------------------------+------------+------------+
| Total wallclock time elapsed since start    | 2.825e+03s |            |
|                                             |            |            |
| Section                         | no. calls |  wall time | % of total |
+---------------------------------+-----------+------------+------------+
| Assemble system                 |       408 | 1.267e+02s |  4.48e+00% |
| Initialize DoFs                 |         1 | 2.218e-01s | 0.000e+00% |
| Mesh initialization             |         1 | 8.832e+00s |  3.13e-01% |
| Solve system                    |       308 | 2.742e+01s |  9.70e-01% |
| Writing                         |        11 | 2.661e+03s |  9.42e+01% |
+---------------------------------+-----------+------------+------------+
```

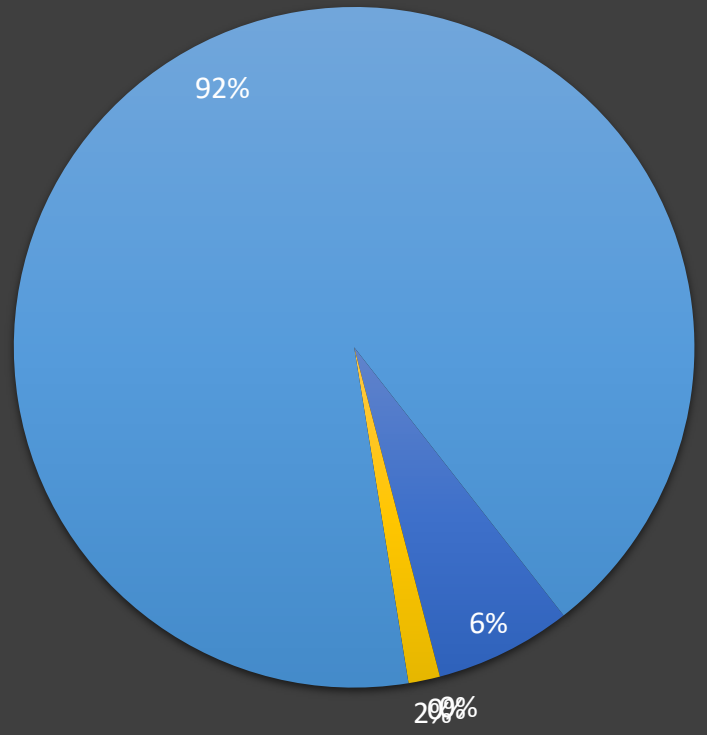
Brain mesh writing the output every 10 timesteps, :



Observations and critical analysis: (4)

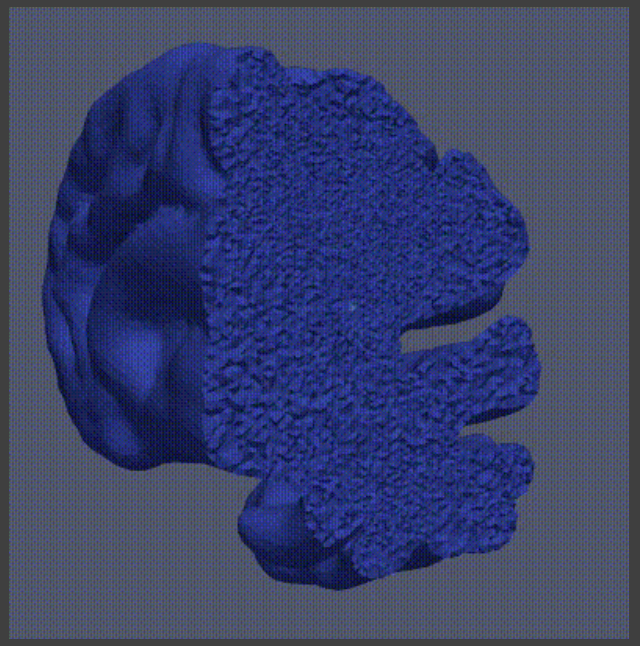
```
void
HeatNonLinear::output(const unsigned int &time_step, const double &time) const {
  DataOut<dim> data_out;
  data_out.add_data_vector(dof_handler, solution, "u");

  // std::vector<unsigned int> partition_int(mesh.n_active_cells());
  // GridTools::get_subdomain_association(mesh, partition_int);
  // const Vector<double> partitioning(partition_int.begin(), partition_int.end());
  // data_out.add_data_vector(partitioning, "partitioning");

  data_out.build_patches();

  std::string output_file_name = std::to_string(time_step);

  // Pad with zeros.
  output_file_name =
    "output-" + std::string(4 - output_file_name.size(), '0') + output_file_name;

  DataOutBase::DataOutFilter data_filter(
    DataOutBase::DataOutFilterFlags(/*filter_duplicate_vertices = */ false,
                                    /*xdmf_hdf5_output = */ true));
  data_out.write_filtered_data(data_filter);
  data_out.write_hdf5_parallel(data_filter, output_file_name + ".h5", MPI_COMM_WORLD);

  std::vector<XDMFEntry> xdmf_entries({data_out.create_xdmf_entry(
    data_filter, output_file_name + ".h5", time, MPI_COMM_WORLD)});
  data_out.write_xdmf_file(xdmf_entries, output_file_name + ".xdmf", MPI_COMM_WORLD);
}
```

```
+---------------------------------------------+------------+------------+
| Total wallclock time elapsed since start    | 2.539e+03s |            |
|                                             |            |            |
| Section                         | no. calls |  wall time | % of total |
+---------------------------------+-----------+------------+------------+
| Assemble system                 |       408 | 1.647e+02s |  6.49e+00% |
| Initialize DoFs                 |         1 | 2.100e-01s | 0.000e+00% |
| Mesh initialization             |         1 | 8.913e+00s |  3.51e-01% |
| Solve system                    |       308 | 3.269e+01s |  1.29e+00% |
| Writing                         |        11 | 2.332e+03s |  9.18e+01% |
+---------------------------------+-----------+------------+------------+
```



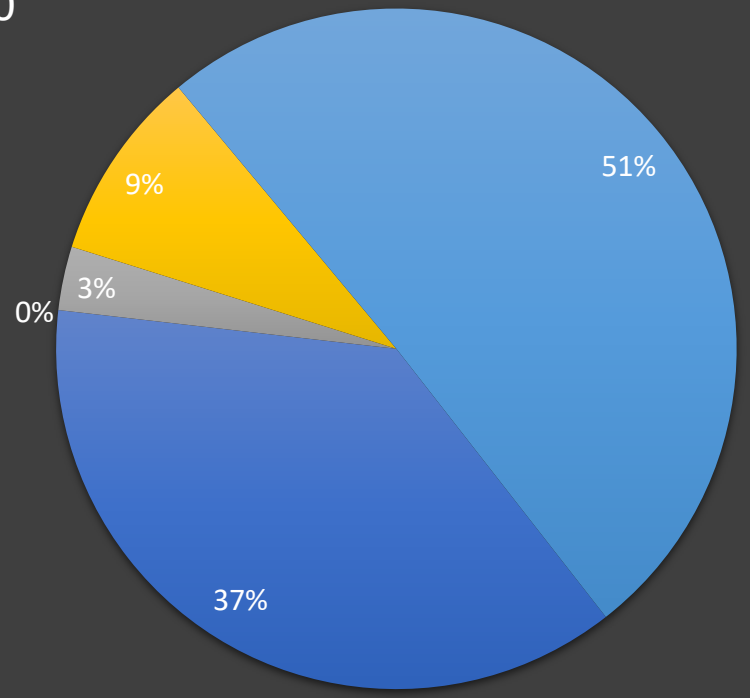
Observations and critical analysis: (4)

Total wallclock time elapsed since start	2.653e+03s		
Section	no. calls	wall time	% of total
Assemble system	391	9.864e+02s	3.72e+01%
Initialize DoFs	1	1.644e+00s	0.000e+00%
Mesh initialization	1	8.793e+01s	3.31e+00%
Solve system	291	2.279e+02s	8.59e+00%
Writing	6	1.347e+03s	5.08e+01%

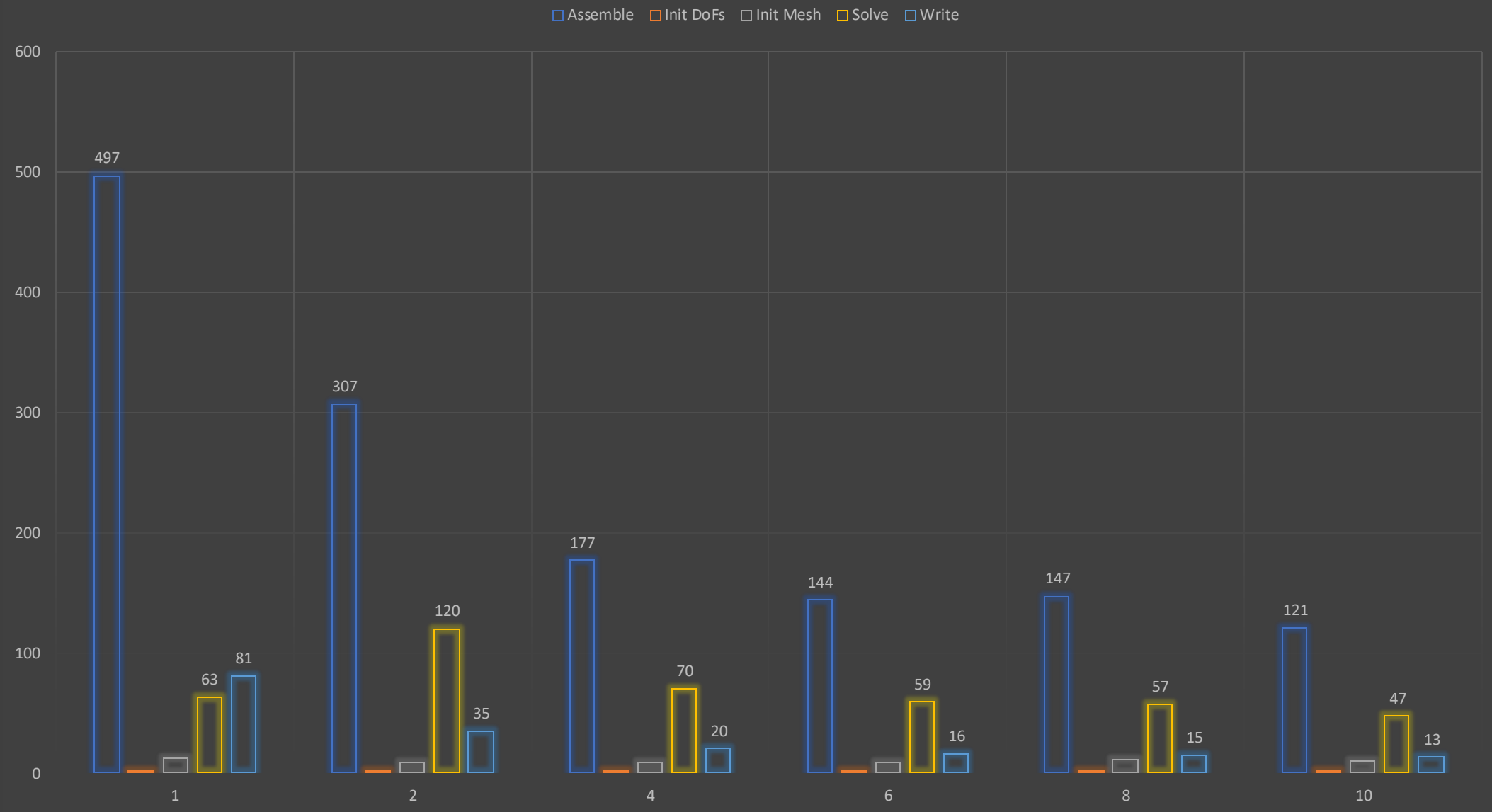


Fine brain mesh
Writing every 30 steps
d^{exn} =1
d^{axn} =2
T = 15.0

We can keep d^{exn} smaller



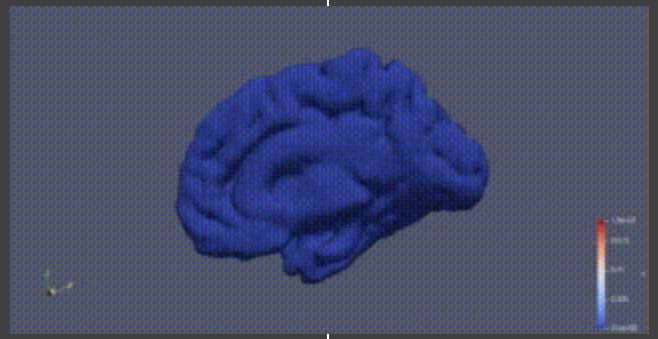
Local writing performance



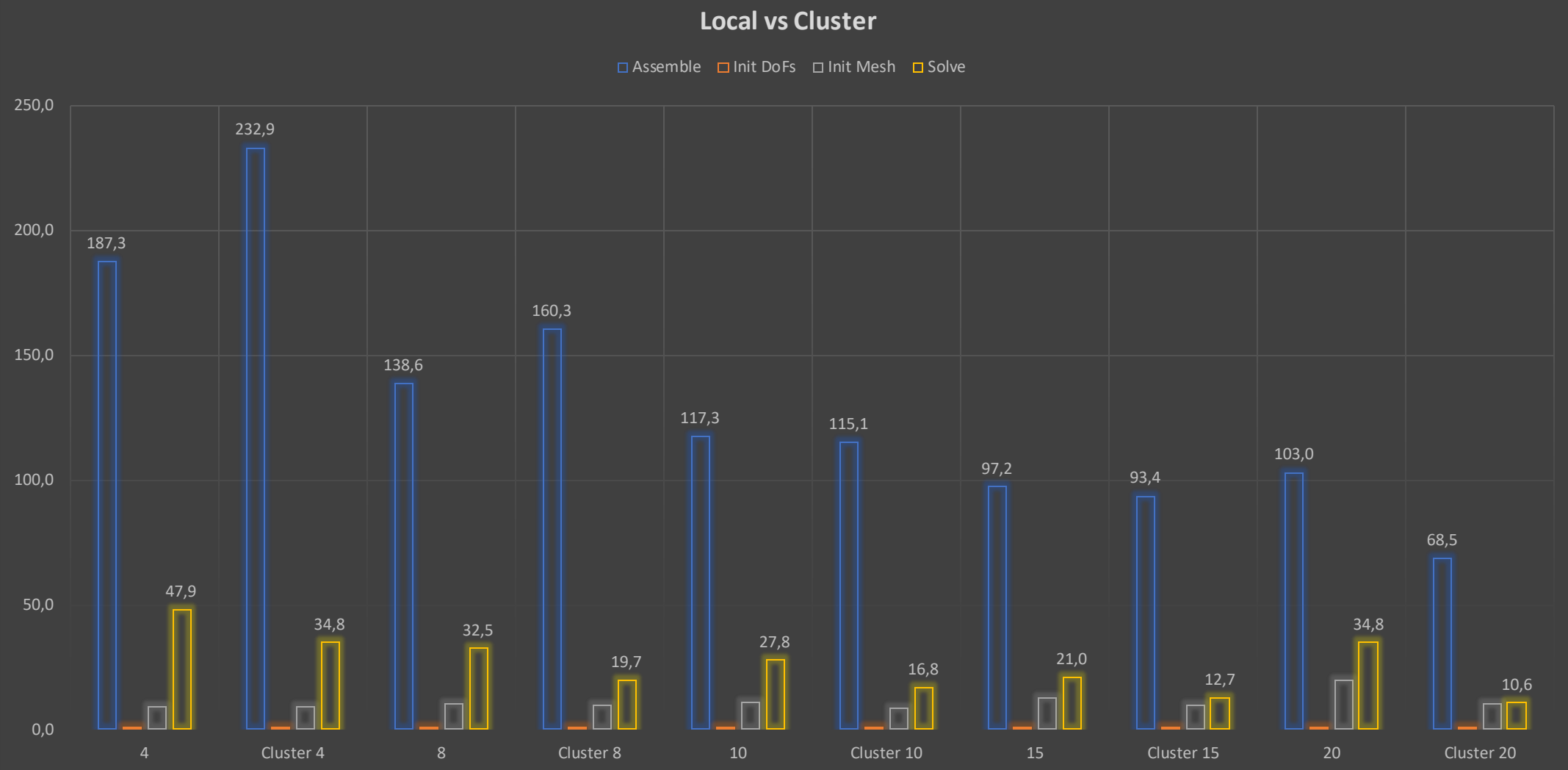
Observations and critical analysis: (4)

Total wallclock time elapsed since start	1.655e+02s		
Section	no. calls	wall time	% of total
Assemble system	408	9.445e+01s	5.71e+01%
Initialize DoFs	1	1.611e-01s	0.000e+00%
Mesh initialization	1	1.586e+01s	9.58e+00%
Solve linear system	308	4.287e+01s	2.59e+01%
Writing	101	1.064e+01s	6.43e+00%

Total wallclock time elapsed since start	2.539e+03s		
Section	no. calls	wall time	% of total
Assemble system	408	1.647e+02s	6.49e+00%
Initialize DoFs	1	2.100e-01s	0.000e+00%
Mesh initialization	1	8.913e+00s	3.51e-01%
Solve system	308	3.269e+01s	1.29e+00%
Writing	11	2.332e+03s	9.18e+01%



Local vs Cluster



Locally, 15 and 20 cpus were obtained with the mpirun command "*--use-hwthread-cpus*", that makes advantage of the number of hardware threads instead of the number of processor cores.

Observations and critical analysis: (5)

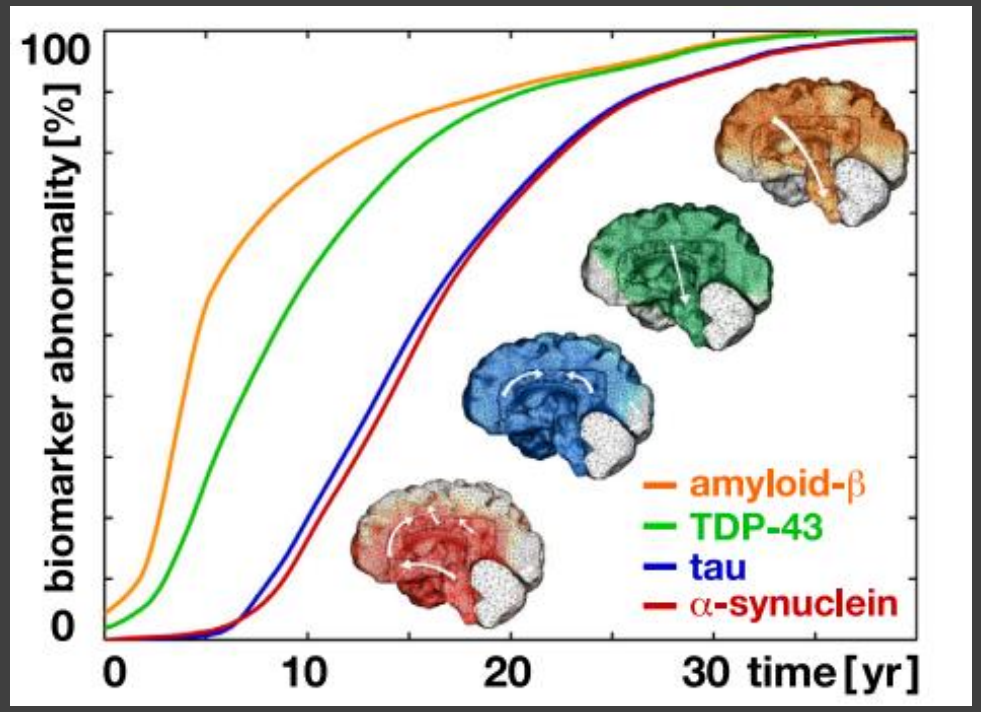
```
void
HeatNonLinear::output(const unsigned int &time_step, const double &time) const {
  DataOut<dim> data_out;
  data_out.add_data_vector(dof_handler, solution, "u");

  std::vector<unsigned int> partition_int(mesh.n_active_cells());
  GridTools::get_subdomain_association(mesh, partition_int);
  const Vector<double> partitioning(partition_int.begin(), partition_int.end());
  data_out.add_data_vector(partitioning, "partitioning");

  data_out.build_patches();

  std::string output_file_name = std::to_string(time_step);

  // Pad with zeros.
  output_file_name =
    "output-" + std::string(4 - output_file_name.size(), '0') + output_file_name;

  DataOutBase::DataOutFilter data_filter(
    DataOutBase::DataOutFilterFlags(/*filter_duplicate_vertices = */ false,
                                    /*xdmf_hdf5_output = */ true));
  data_out.write_filtered_data(data_filter);
  data_out.write_hdf5_parallel(data_filter, output_file_name + ".h5", MPI_COMM_WORLD);

  std::vector<XDMFEntry> xdmf_entries({data_out.create_xdmf_entry(
    data_filter, output_file_name + ".h5", time, MPI_COMM_WORLD)});
  data_out.write_xdmf_file(xdmf_entries, output_file_name + ".xdmf", MPI_COMM_WORLD);
}
```

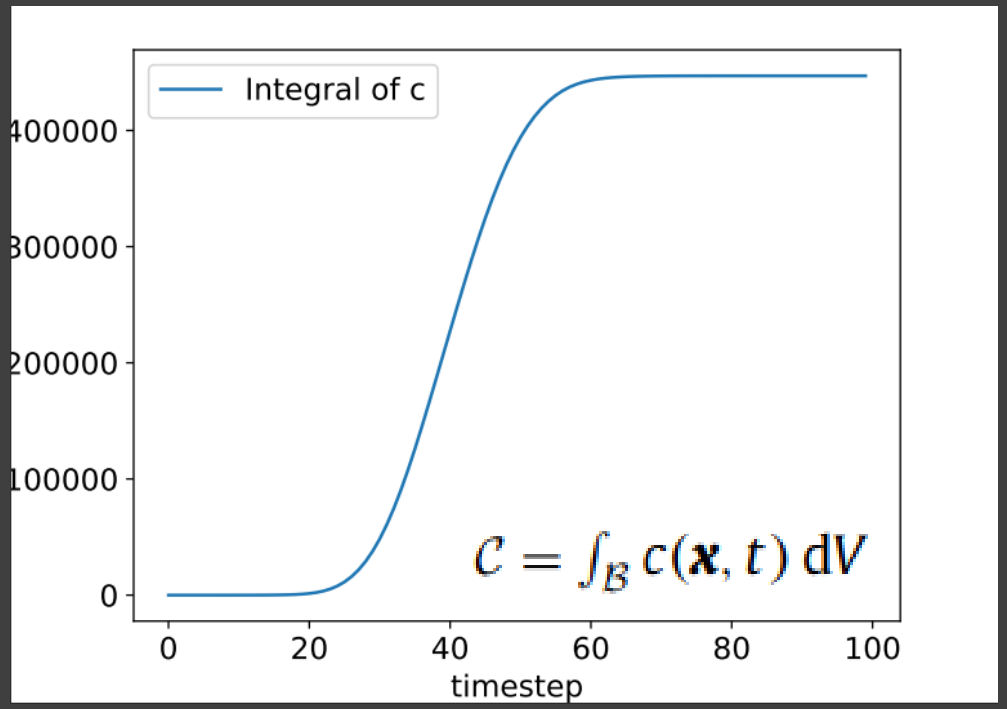
Write on scratch

Initially, using the cluster, we could not open the .xdmf files.
That was because we have changed the output_file_name in “/scratch/hpc/par8/…”.
Doing so we were changing also the parameter of xdmf_entries. That parameter is used by Paraview to find the .h5 file correspondent to the .xdmf.
Having changed the name also there, it was searched in “/scratch/hpc/par8/…”, which can’t be found on our computer

Biomarker abnormality

with the histological patterns in diseased human brains. When integrated across the brain, our concentration profiles result in biomarker curves that display a striking similarity with the sigmoid shape and qualitative timeline of clinical biomarker models. Our results sug-





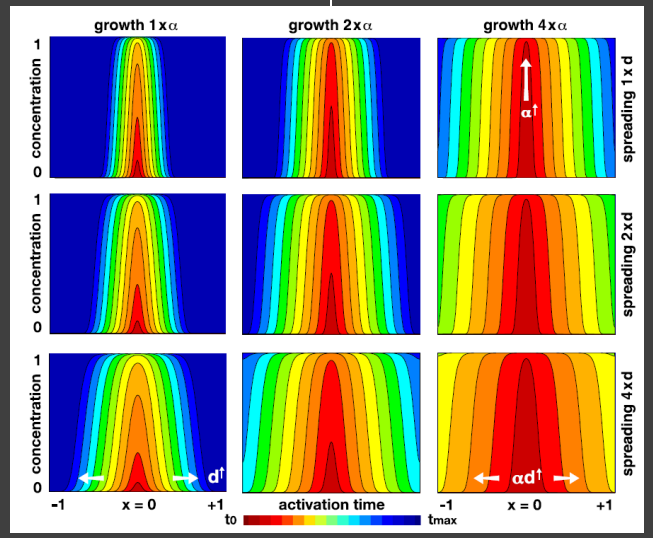
.csv + script di Phyton

“ Biomarker abnormality from prion-like spreading across across the whole brain. The biomarker-time relations display a smooth sigmoidal form, which is in agreement with clinical biomarker models of neurodegeneration “

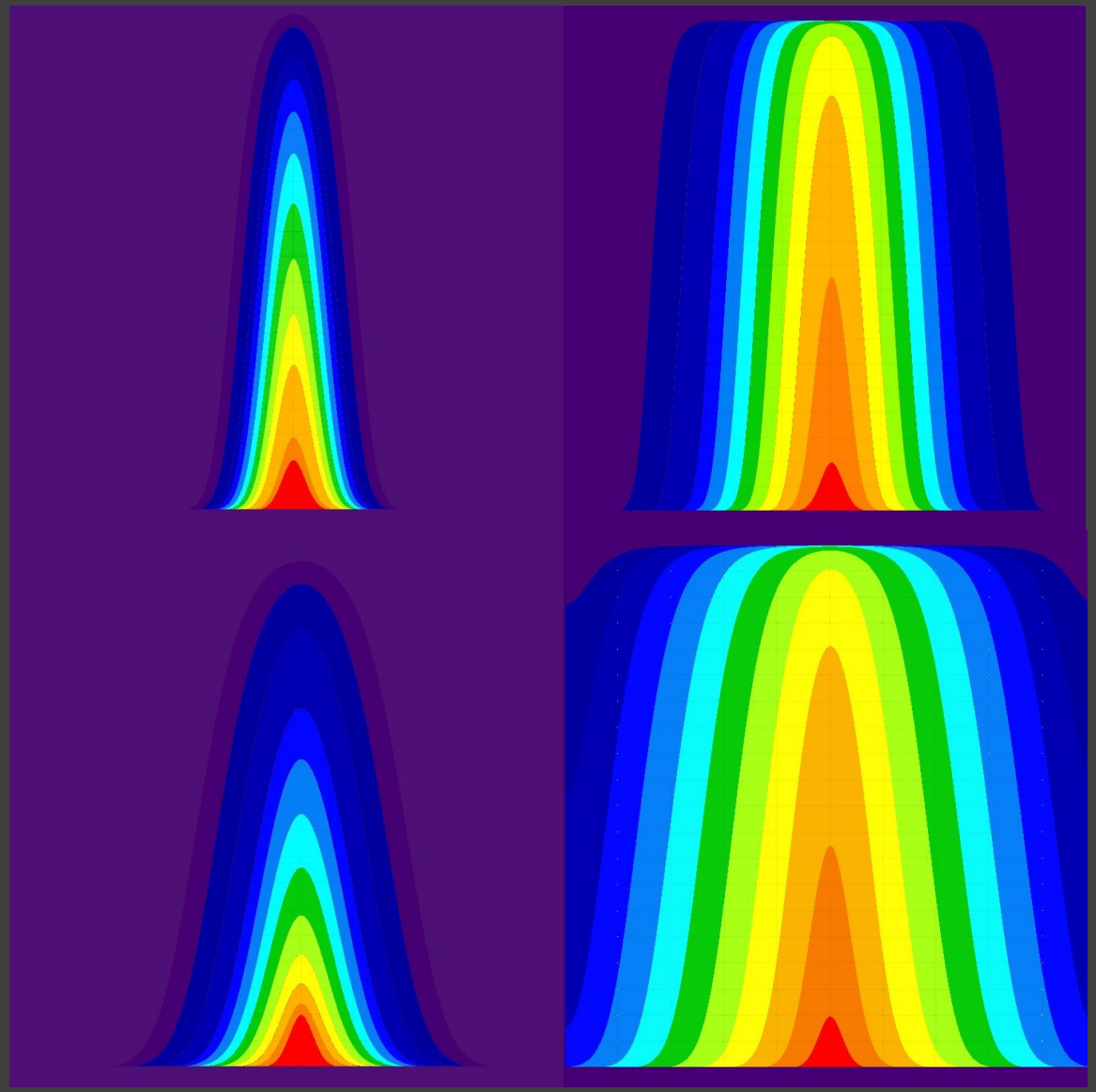
Activation pattern

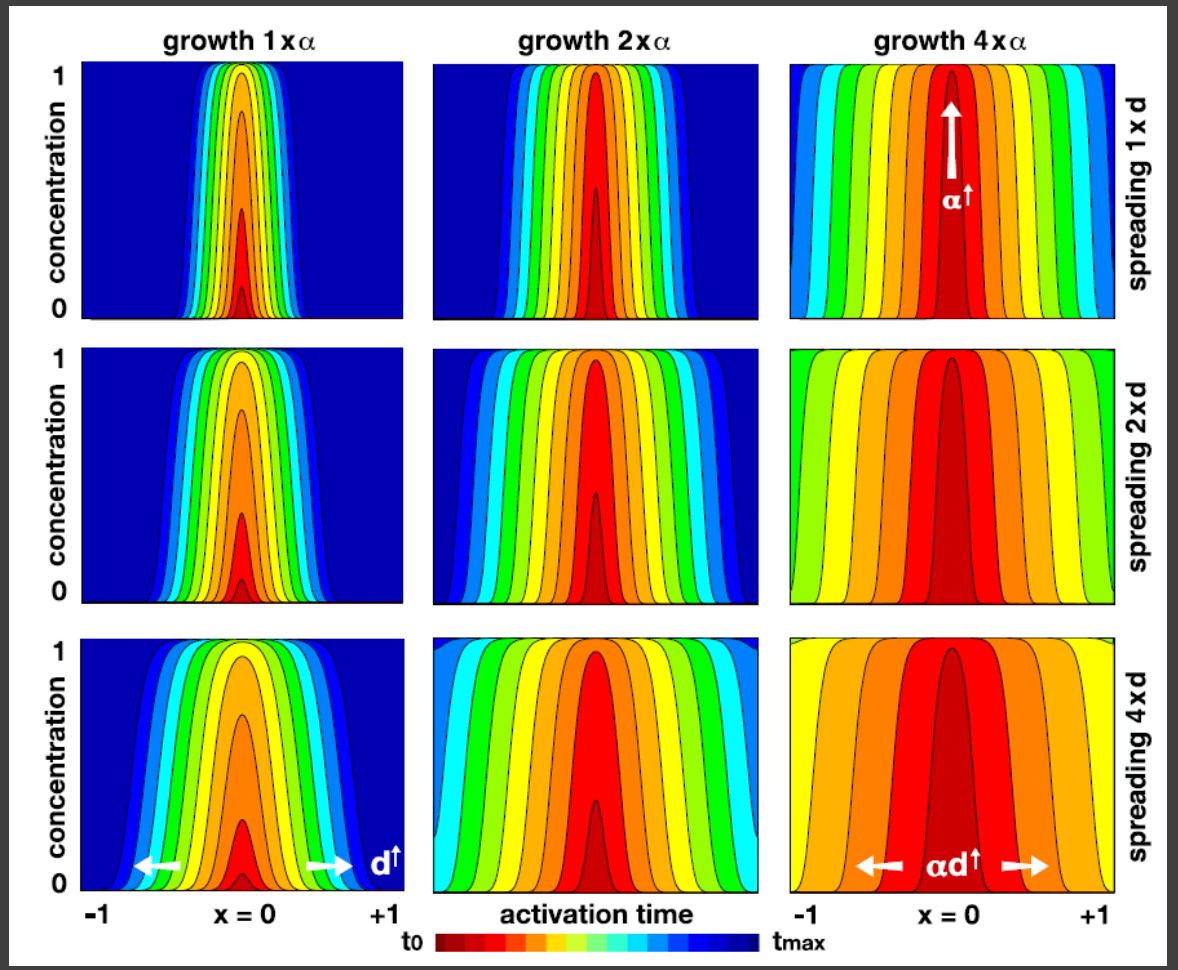
Our one-dimensional model problem in Fig. 3 suggests that the Fisher–Kolmogorov Eq. (8) accurately captures the characteristic features of prion-like diseases: (i) disease progression is inevitable after inoculation and the misfolded protein concentration inevitably converges towards saturation at $c = 1.0$ once misfolding is seeded with $c_0 > 0.0$ anywhere in the domain; (ii) the duration of the incubation period depends on both the initial seeding concentration c_0 and the growth rate α; and (iii) disease progression is characterized by a long, clinically silent incubation period, $0.0 < c < c^{\text{crit}}$, during which the disease gradually amplifies locally, followed by a rapid local increase towards $c = 1.0$, and a global spreading from the initial seeding region across the entire domain (Jucker and Walker, 2013).

of healthy protein $p_0 = k_0/k_1$, and on the clearance rate $\tilde{k}_1$. Our simulations in Fig. 8 suggest that the activation pattern is rather insensitive to the growth rate α, which mainly affects the timing of activation but not the overall pattern, and highly sensitive to both extracellular diffusion d^{ext} and axonal transport d^{axn}, which affect both timing and patterning. Our results



Activation pattern





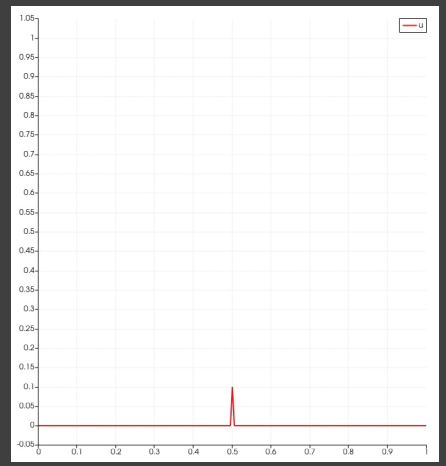
Activation pattern

- **Concentration maps** illustrate the primary variable, the concentration of misfolded protein c (x, t), at every point x across the domain B at a given point in time t , ranging from 0.0 ≤c ≤1.0 (spatial evolution of the primary unknown).

- **Activation maps** illustrate the activation time t (x), the time at which the local concentration of misfolded protein exceeds a critical value of c crit = 0 . 95 (temporal progression of a biomarker above its detection threshold c crit).

- Total concentration of misfolded protein **integrated** across the entire brain, C(t) = B c(x , t) d V (temporal evolution of the biomarker abnormality).

The rainbow color code highlights the activation time, i.e., the time t at which the regional concentration of misfolded protein exceeds the critical threshold of 95%, thus c > 0.95. Red regions activate first, blue regions activate last.



d^{ext} and d^{axn}

$$\boldsymbol{D} = d^{\mathrm{ext}}\boldsymbol{I} + d^{\mathrm{axn}}\,\boldsymbol{n}\otimes\boldsymbol{n}\,.$$

Once seeded, misfolded protein can spread across the brain by two distinct mechanisms, extracellular diffusion and axonal transport . Here we associate extracellular diffusion d^{ext} with the isotropic diffusion of misfolded protein through the extracellular space and axonal transport d^{axn} with the anisotropic diffusion of misfolded protein along the local axonal direction n

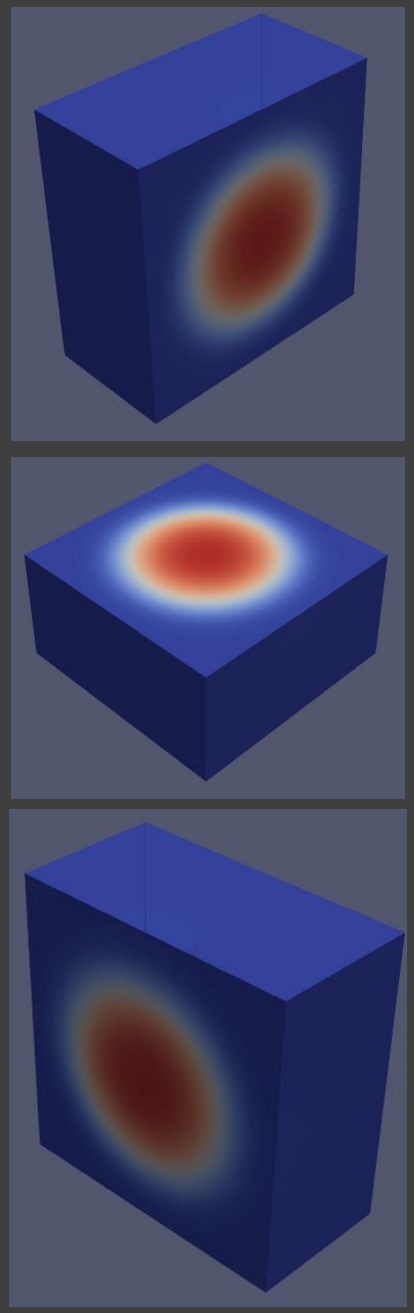
The axonal direction depends on the fibers model used. Initially only the extracellular diffusion was taken into account (D = IdentityMatrix * d^{ext}, with d^{ext} = 0.002).

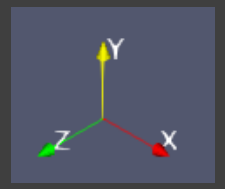
The tensor D regulates how c is distributed in the space in the different directions. If D = Identity * constant the diffusion is equal in all the directions. To visualize a bit of anisotropy we can choose a constant versor **n** to visualize a preferred route for the diffusion.

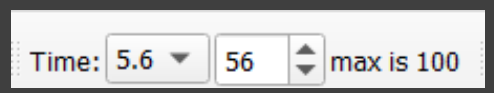
where $\boldsymbol{D} = \boldsymbol{D}_{\tilde{p}}$ denotes the spreading of misfolded protein that can be either purely isotropic, such that $\mathrm{Div}(\boldsymbol{D}\cdot\nabla c) = d\,\nabla c$, or anisotropic along pronounced communication networks within the brain. Importantly, in this model, the growth rate

n_axn variation

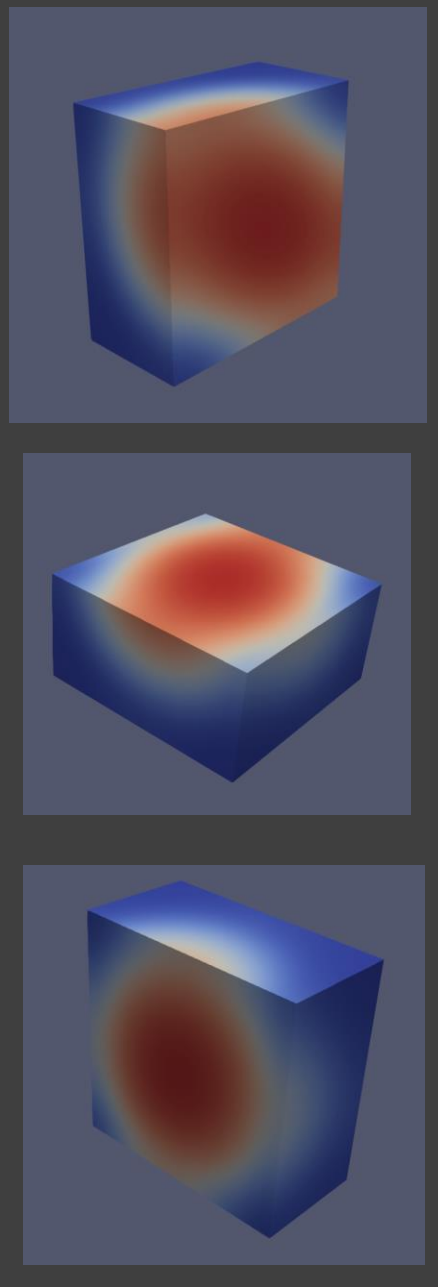
n_axn = 0;



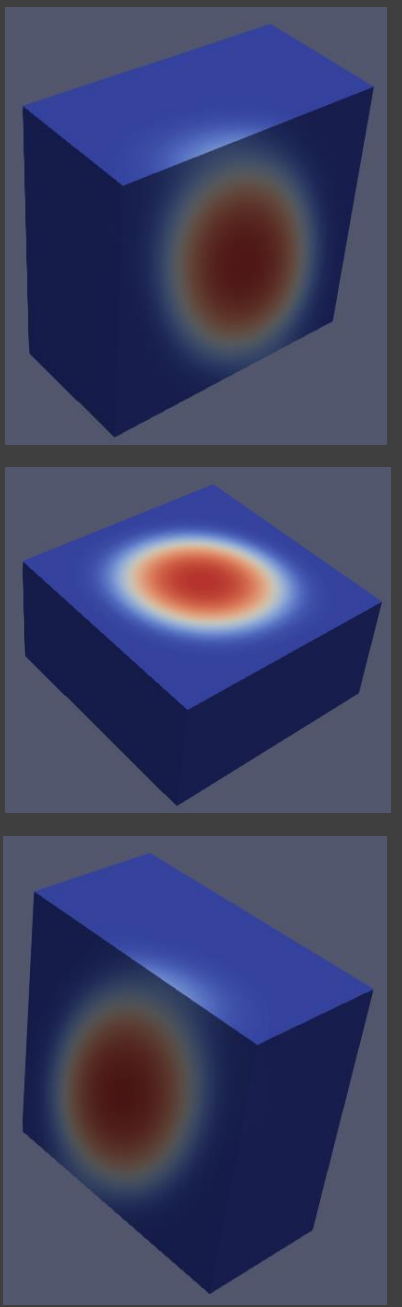




n_axn =1/sqrt(14) $(1, 2, 3)^T$



n_axn = $(0, 1, 0)^T$

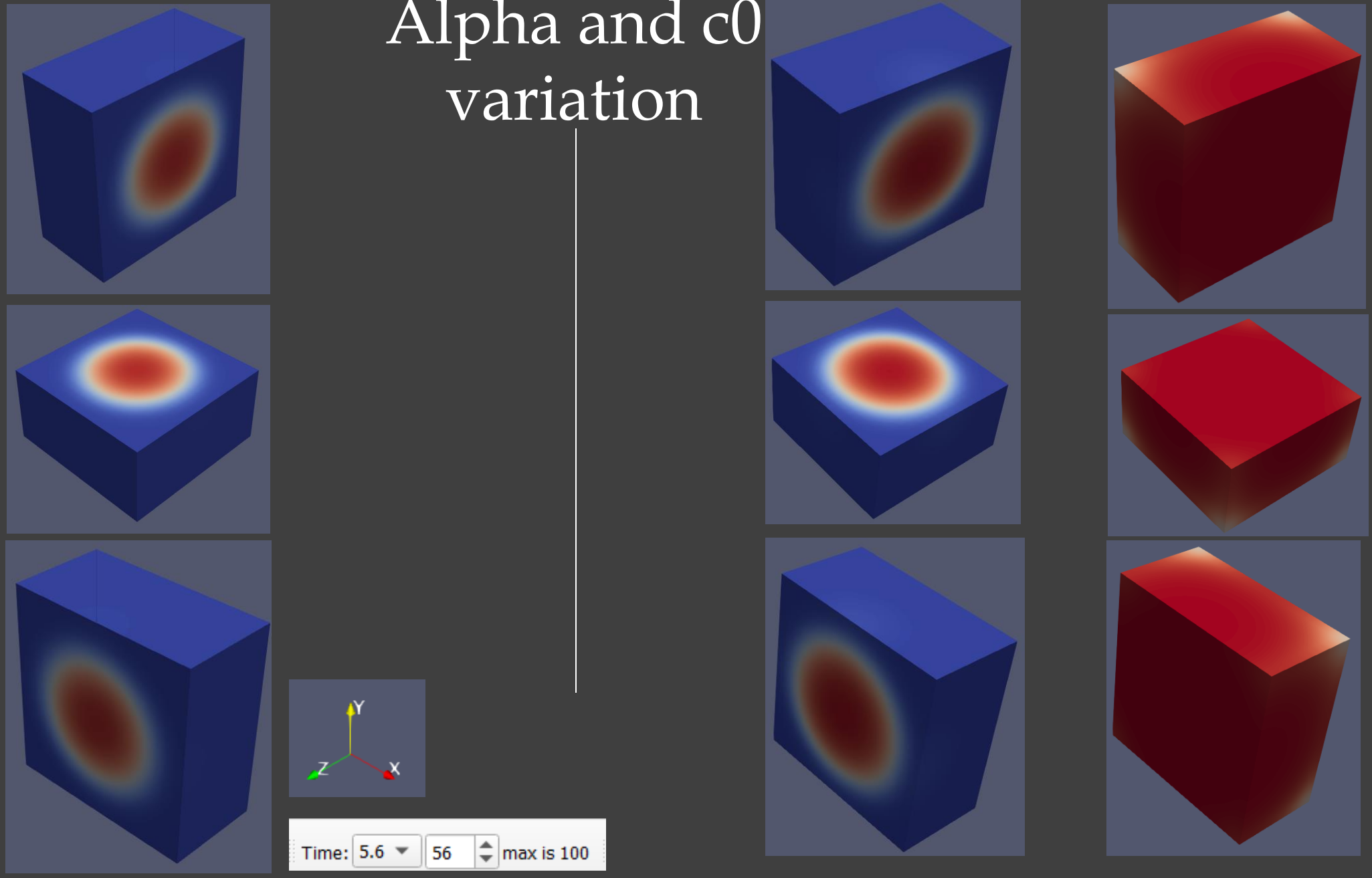


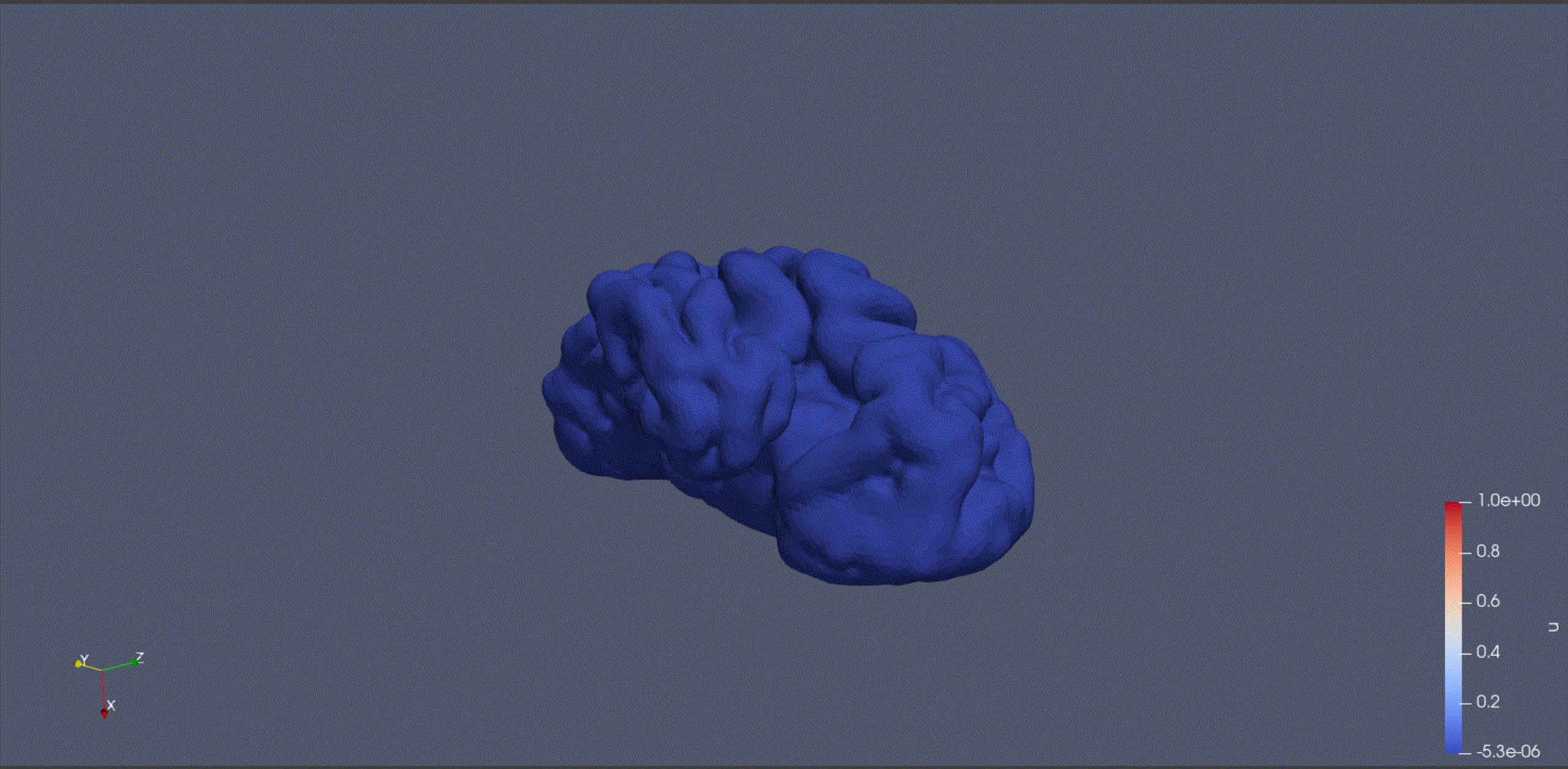
Alpha and c0 variation

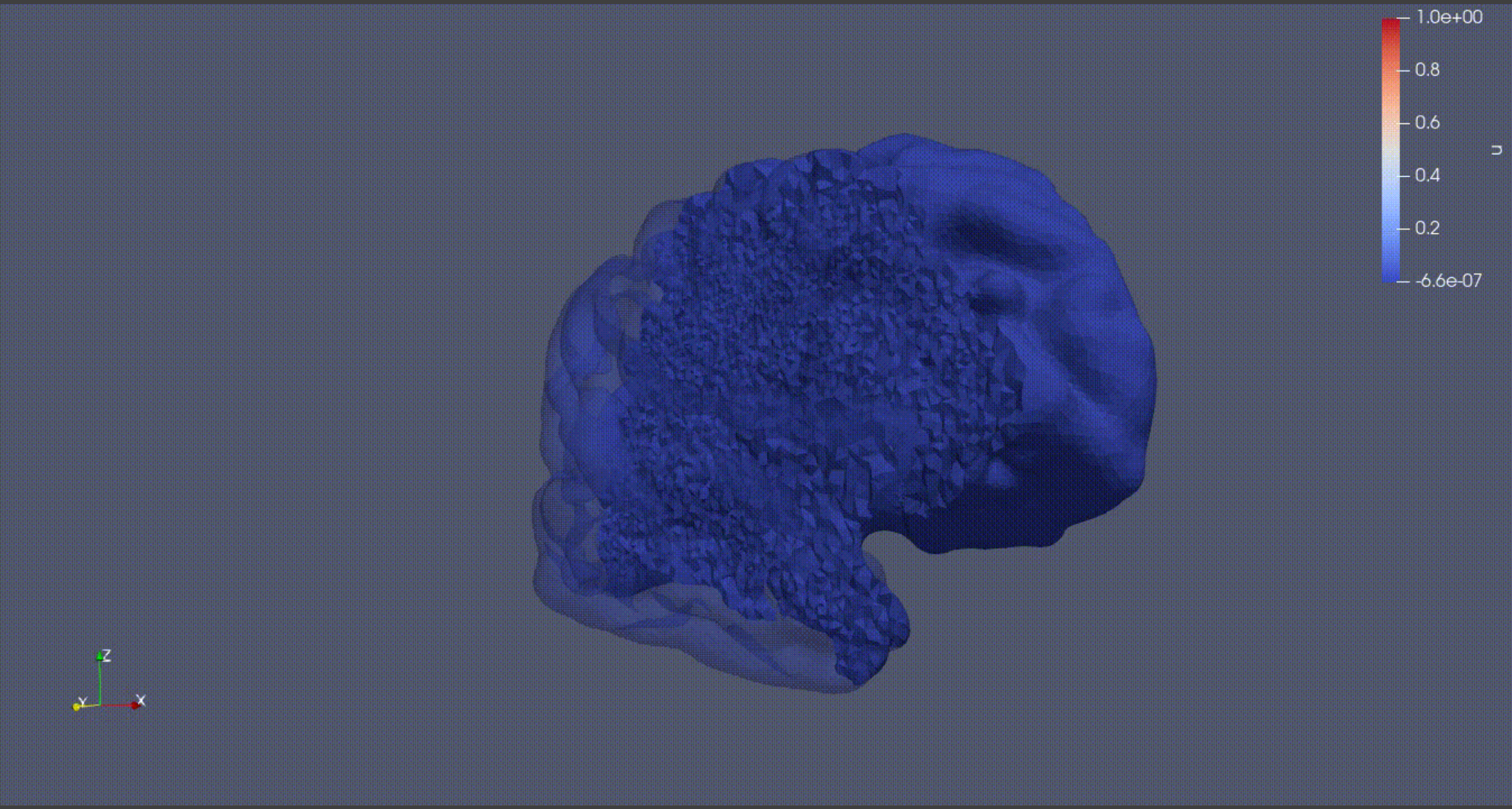
n_axn = 0;

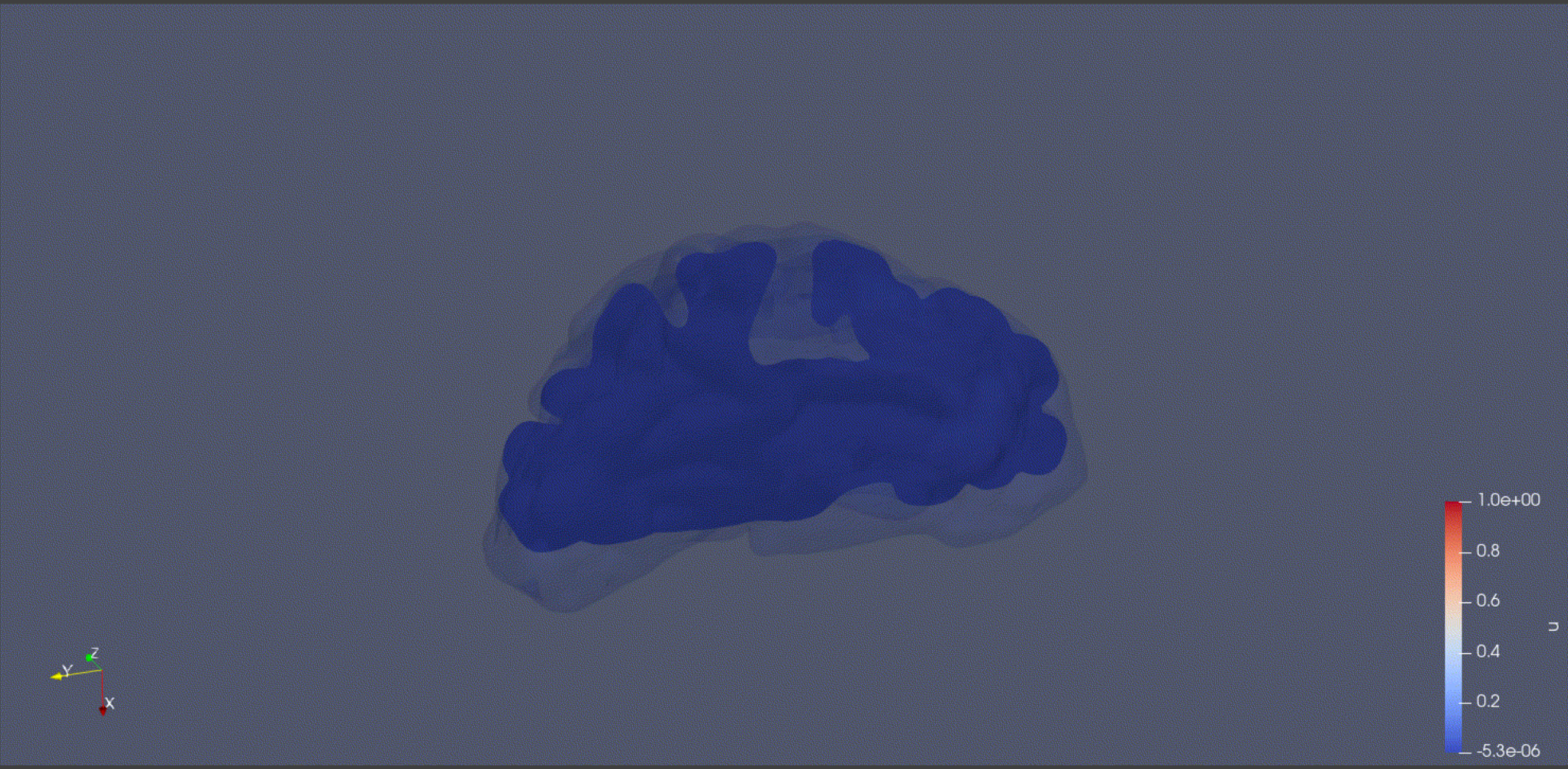
n_axn = 0 && C_0*5

n_axn = 0 && alpha*2

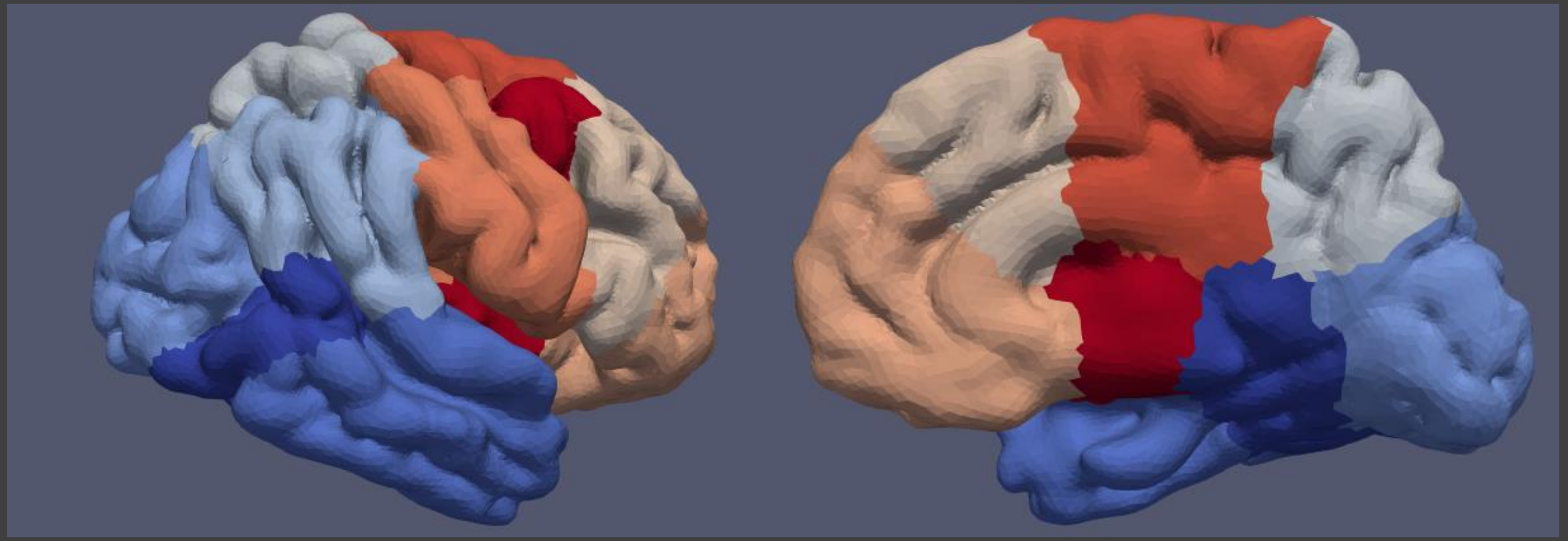



Y
Z
X
1.0e+00
0.8
0.6
0.4
0.2
-5.3e-06
U


1.0e+00
0.8
0.6
0.4
0.2
-6.6e-07
Z
Y
X


Z
Y
X
1.0e+00
0.8
0.6
0.4
0.2
-5.3e-06

Brain partitioning



OUR PROBLEM: FISHER-KOLMOGOROV EQUATIONS WITH HOMOGENEOUS NEUMANN BOUNDARY CONDITIONS

$$\begin{cases} \dfrac{\partial c}{\partial t} = \nabla \cdot (D \nabla c) + \alpha c (1 - c) & \text{IN } \Omega \times (0,T) \\ \nabla c \cdot \hat{n} = 0 & \text{ON } \partial\Omega \times (0,T) \\ c = c_0 & \text{IN } \Omega \times \{0\} \end{cases}$$

c: RELATIVE CONCENTRATION OF MISFOLDED PROTEINS

α: GROWTH OF THE CONCENTRATION

D: SPREADING TENSOR

$$D = d^{ext} \cdot I + d^{axn} \cdot \hat{n} \otimes \hat{n}$$

d^{ext}: ISOTROPIC DIFFUSION

d^{axn}: ANISOTROPIC DIFFUSION

$\Omega \subset \mathbb{R}^3$, Ω IS THE BRAIN, $c = c(\bar{x}, t)$

WEAK FORMULATION:

LET $v \in V$ BE A TEST FUNCTION INSIDE A SUITABLE FUNCTION SPACE V (T.B.D.)

MULTIPLY v TO EQUATION 1, AND INTEGRATE OVER Ω

$$\int_\Omega \frac{\partial c}{\partial t} v \, d\bar{x} = \int_\Omega \nabla \cdot (D \nabla c) v \, d\bar{x} + \int_\Omega \alpha c (1 - c) v \, d\bar{x}$$

PARTIAL INTEGRATION

SINCE $\nabla \cdot ((D \nabla c) \cdot v) = \nabla \cdot (D \nabla c) v + \nabla v \, D \nabla c$

$$\Rightarrow \nabla \cdot (D \nabla c) v = \nabla \cdot ((D \nabla c) \cdot v) - \nabla v \, D \nabla c$$

AND THE INTEGRAL BECOMES

$$\int_\Omega \nabla \cdot (D \nabla c) v \, d\bar{x} = \int_\Omega \nabla \cdot ((D \nabla c) \cdot v) \, d\bar{x} - \int_\Omega \nabla v \, D \nabla c \, d\bar{x}$$

DIVERGENCE THEOREM

$$= \int_{\partial\Omega} (D \nabla c) \cdot \hat{n} \, v \, d\sigma - \int_\Omega \nabla v \, D \nabla c \, d\bar{x}$$

NEUMANN BOUNDARY CONDITIONS

FOR THESE INTEGRALS TO BE WELL-DEFINED, I NEED $v, c \in H^1(\Omega)$
SO I TAKE $V = H^1(\Omega)$ AND THE WEAK FORMULATION READS:

WF: $\forall t \in (0,T)$, FIND $c \in V$ s.t.

$$\int_\Omega \frac{\partial c}{\partial t} v \, d\bar{x} = -\int_\Omega \nabla v \, D \nabla c \, d\bar{x} + \int_\Omega \alpha c (1-c) v \, d\bar{x} \quad \forall v \in V, \quad c(\bar{x},0) = c_0$$

I CAN TAKE THE RESIDUAL FORM

$$\int_\Omega \frac{\partial c}{\partial t} v \, d\bar{x} + \int_\Omega \nabla v \, D \nabla c \, d\bar{x} - \int_\Omega \alpha c (1-c) v \, d\bar{x} = 0$$

AND BY CALLING $b(c)(v) = \int_\Omega \nabla v \, D \nabla c \, d\bar{x} - \int_\Omega \alpha c (1-c) v \, d\bar{x}$

AND $R(c)(v) = \int_\Omega \frac{\partial c}{\partial t} v \, d\bar{x} + b(c)(v)$

$\forall t \in (0,T)$, FIND $c \in V$ s.t. $R(c)(v) = 0 \quad \forall v \in V$

SPACE DISCRETIZATION

INTRODUCE THE FINITE ELEMENT SPACE $X_h^r(\Omega)$ OF PIECEWISE POLINOMIALS OF MAXIMUM DEGREE r: $\dim\{X_h^r\} = \tilde{N}_h < +\infty$

LETS CALL $V_h = V \cap X_h^r(\Omega)$: I KNOW $\dim\{V_h\} = N_h \leq \tilde{N}_h < +\infty$

AND LET $\{\varphi_j\}_{j=1}^{N_h}$ BE A SET OF LAGRANGIAN BASIS FUNCTIONS FOR V_h,

WHERE φ_j IS A PIECEWISE POLYNOMIAL OF DEGREE r

I CAN NOW WRITE THE SEMI-DISCRETE FORM OF THE PROBLEM AS

$$\forall t \in (0,T), \text{ FIND } c_h \in V_h \text{ s.t. } R(c_h)(v_h) = 0 \quad \forall v_h \in V_h$$

TIME DISCRETIZATION

PARTITION $(0,T)$ IN SUB-INTERVALS OF SIZE Δt, $(t_n, t_{n+1}]$, $n = 0, 1, \dots, N_T - 1$

LET'S CALL $c_h^n = c_h(t_n)$

LET'S USE FINITE DIFFERENCES TO DISCRETIZE IN TIME, AND LET'S USE IMPLICIT EULER TO DO THIS

$$R^{n+1}(c_h^{n+1})(v_h) = \int_\Omega \frac{c_h^{n+1} - c_h^n}{\Delta t} v_h \, d\bar{x} + b(c_h^{n+1})(v_h) = 0 \quad \forall v_h \in V_h, \ \forall n = 0, 1, \dots, N_T - 1$$

THIS IS A NON LINEAR PROBLEM AND I CAN USE THE NEWTON METHOD TO SOLVE IT

BY COMPUTING THE FRÉCHET DERIVATIVE AS

$$a(c_h^n)(\delta_h, v_h) = \lim_{h \to 0} \frac{1}{h} \left[R^{n+1}(c_h^{n+1} + \delta_h h)(v_h) - R^{n+1}(c_h^{n+1})(v_h) \right]$$

$$= \dots$$

$$= \int_\Omega \delta_h \frac{1}{\Delta t} v_h \, d\bar{x} + \int_\Omega \nabla \delta_h \, D \, \nabla v_h \, d\bar{x} - \int_\Omega \delta_h \alpha (1 - 2 c_h^{n+1}) v_h \, d\bar{x}$$

THE NEWTON METHOD PROCEEDS AS:

GIVEN THE INITIAL SOLUCTION c_h^0, ITERATE OVER $n = 0, 1, \dots, N_T - 1$ TO COMPUTE c_h^{n+1} BY SOLVING OUR FULLY DISCRETE PROBLEM

AT TIME STEP n, THE NEWTON ITERATIONS PERFORM THE FOLLOWING:

GIVEN AN INITIAL GUESS $c_h^{n+1\,(0)} = c_h^n$, ITERATE UNTIL CONVERGENCE

- SOLVE THE LINEAR SYSTEM

  $$a(c_h^{n+1\,(k)})(\delta_h^{(k)}, v_h) = -R(c_h^{n+1\,(k)})(v_h) \quad \forall v_h \in V_h$$

  BY FINDING $\delta_h^{(k)}$
- UPDATE $c_h^{n+1\,(k+1)} = c_h^{n+1\,(k)} + \delta_h^{(k)}$

THE LINEAR SYSTEM IS SOLVED WITH AN ITERATIVE SOLVER AND HAS COMPONENTS:

- $J^{(k)}$ IS THE SYSTEM MATRIX,

  $$(J)_{ij} = \int_\Omega \varphi_j \frac{1}{\Delta t} \varphi_i \, d\bar{x} + \int_\Omega \nabla\varphi_j \, D \, \nabla\varphi_i \, d\bar{x} - \int_\Omega \varphi_j \, \alpha (1 - 2\, c_h^{n+1\,(k)}) \, \varphi_i \, d\bar{x}$$

- $\bar{R}^{(k)}$ IS THE RESIDUAL VECTOR,

  $$(\bar{R})_i = -\int_\Omega \varphi_i \frac{c_h^{n+1\,(k)} - c_h^n}{\Delta t} d\bar{x} - \int_\Omega \nabla\varphi_i \, D \, \nabla c_h^{n+1} \, d\bar{x} + \int_\Omega \varphi_i \, \alpha \, c_h^{n+1\,(k)} (1 - c_h^{n+1\,(k)}) \, d\bar{x}$$

$$\Rightarrow J^{(k)} \delta^{(k)} = R^{(k)}$$