UPPSALA
UNIVERSITET

Modelling complex systems

Project 1: A firing brain & Spread of memes

Peili Guo
Peili.Guo.7645@student.uu.se

April 8, 2018

1 A firing brain

In this part, a simple program in matlab was written to simulate a firing brain with two-dimensional cellular automata model on a N by N grid with periodic boundary conditions.

each grid represents a neuron. there are 3 different states for neuron: ready(0), firing(1), and resting(2). the rules for transit to the next time steps are:

1. a ready neuron fires on the next time step if there are exactly two neighbours that are firing. ($0 \rightarrow 1$, if two neighbours are 1).
2. a firing neuron goes to the resting state on the next time step ($1 \rightarrow 2$).
3. a firing neuron goes to ready state on the next time step ($2 \rightarrow 0$).

1.1 simple simulation in matlab

Below are the results for a 40 x 40 grid where initially each cell has a probability of 0.3 being a firing(1) cell and all other neurons are ready. the figures below shows how the cells looks like after 10, 20, 100 and 1000 time steps.

The initial probability of being in a firing state is 0.3, which means at t = 0, there are around 480 cells that are being in a firing state. as we can see in figure 5, the total number of firing cell decreases over time. In the beginning, the number of cell decreases very fast, and the total number of cells gets stable around t = 300 400. When simulate 100 times to t = 1000, the average firing cell at t = 400 is around 14 and at t = 1000, the average firing cell is around 12. At the equilibrium state, the shapes that remains are travelling forward at a constant rate preserving the same shape either in the same direction(up/down or left/right), or will never interact if there are shape that travel in the up/down direction and others in left/right direction. Over 100 simulation, the curve of average firing cell decreases in an exponential model, and a exponential model of y = a* exp(b*x) + c*exp(d*x) was fitted to the curve in figure 6.

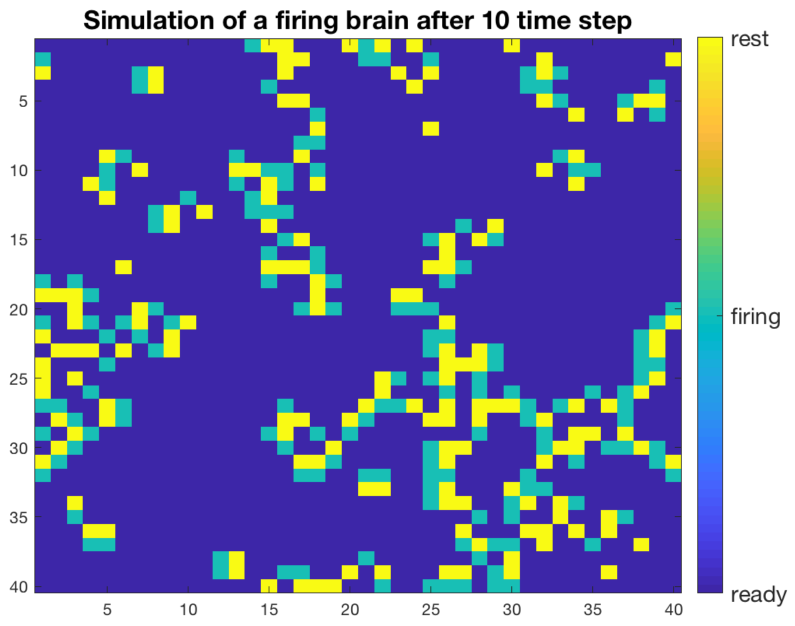

Figure 1: 40x40 cell grid at t = 10

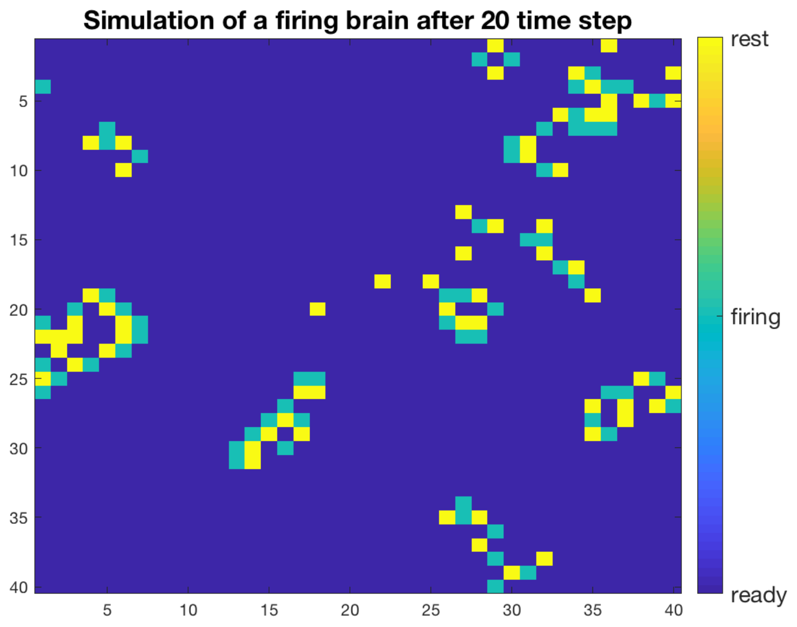

Figure 2: 40x40 cell grid at t = 20

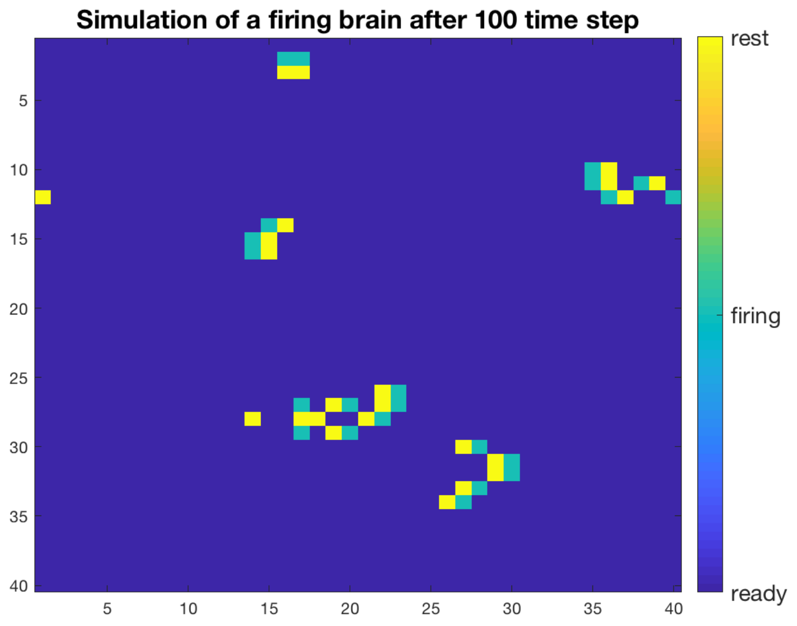

Figure 3: 40x40 cell grid at t = 100

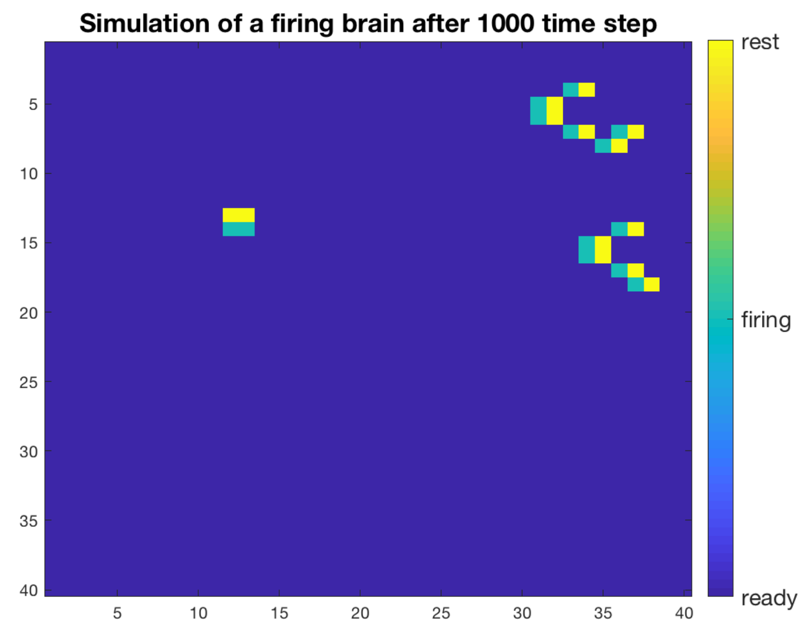

Figure 4: 40x40 cell grid at t = 1000

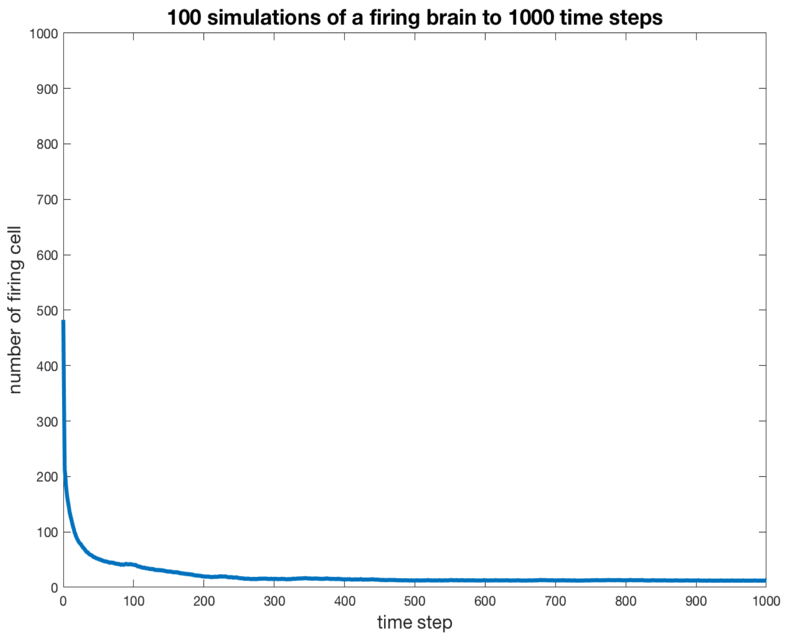

Figure 5: average number of firing cells over time with 100 simulation of different initial conditions

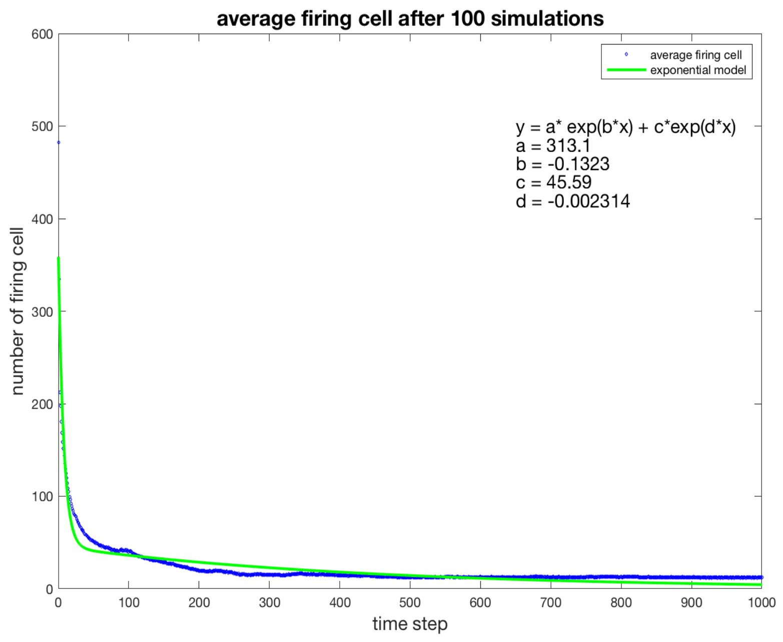

Figure 6: average number of firing cells over time with 100 simulation and exponential model fitting

1.2 example of shapes

1.2.1 move forward at a rate of one cell per time step, while preserving the same shape

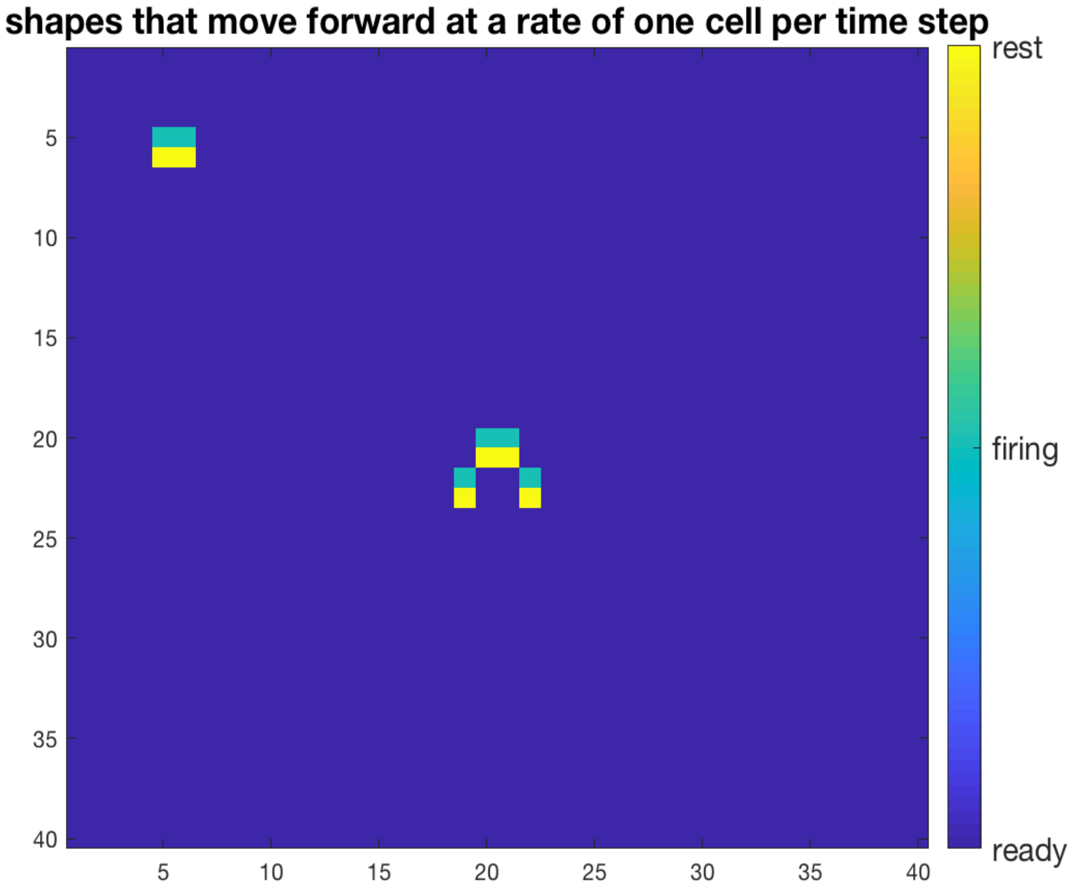

Figure 7: shapes that move forward at a rate of one cell per time step preserving the same shape

1.2.2 move forward at a rate of one cell per time step, launching other shapes behind them

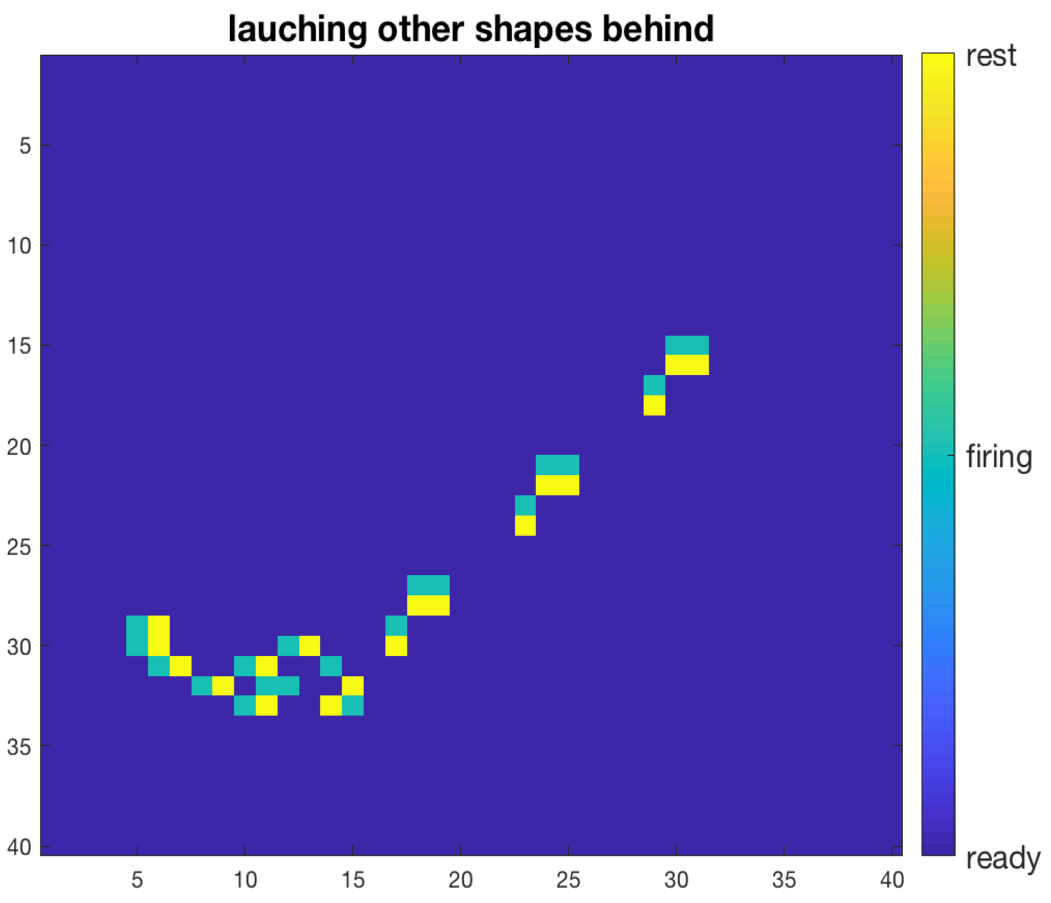

Figure 8: shapes that move forward at a rate of one cell per time step, launching other shapes behind them

1.2.3 move forward at a rate of less than one cell per time step, while returning to the same shape after some period

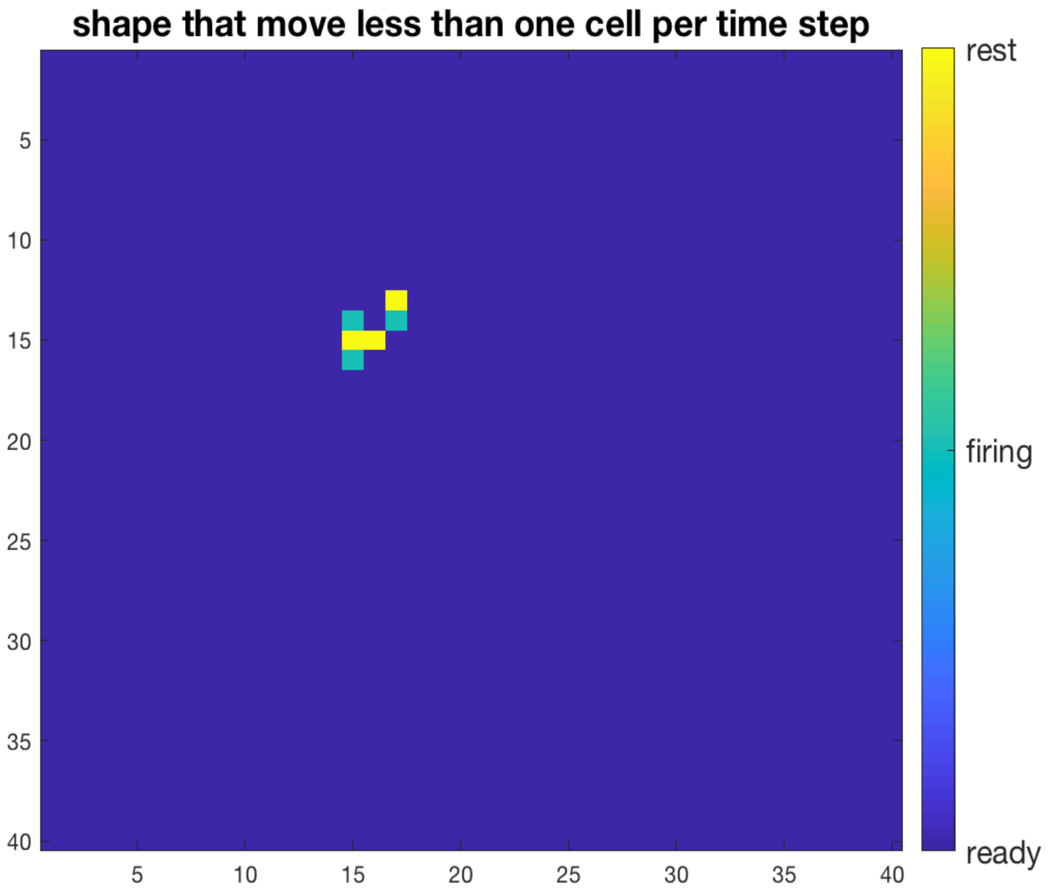

Figure 9: shapes that move less than one cell per time step, returning to same shape after some period

1.2.4 stay stationary but oscillate periodically

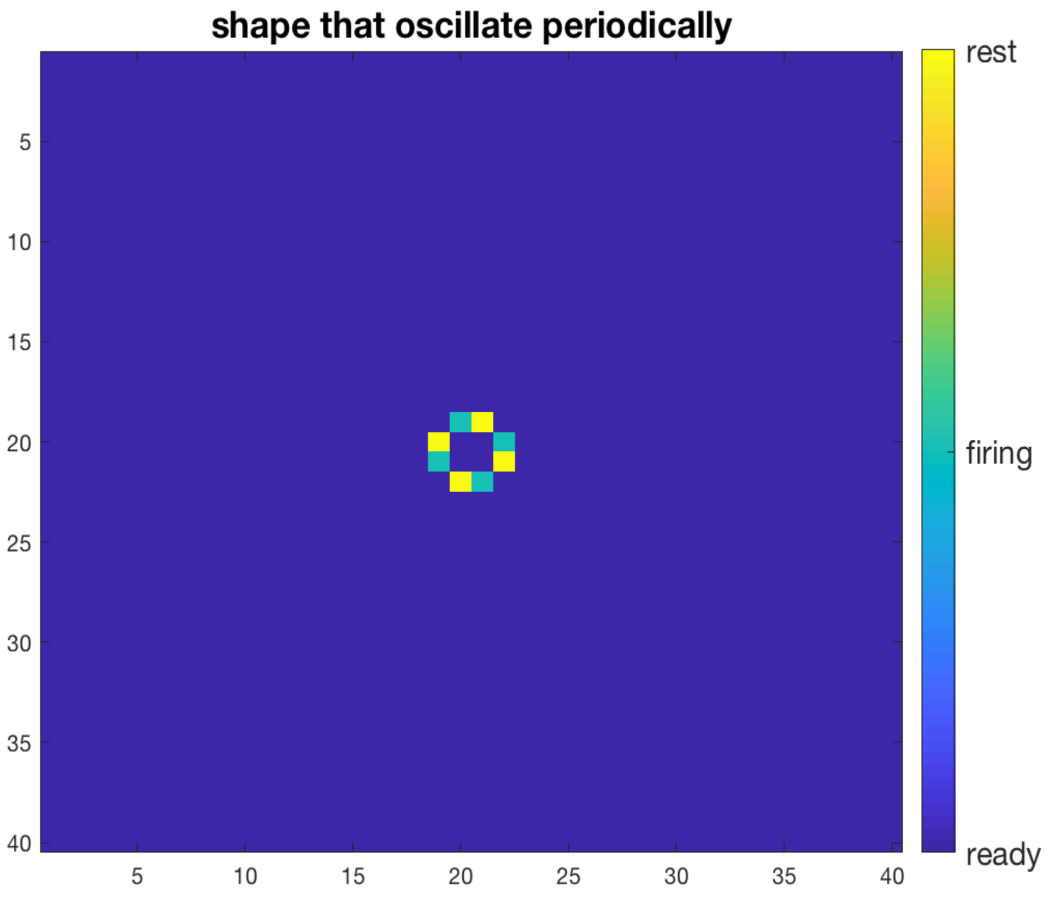

Figure 10: shapes that stay stationary but oscillate periodically

In matlab, we define a geometry by calling circleg (a unit circle) and then call initmesh to create a mesh with a defined maximum maximum mesh size. the first part of setting up a 2D analysis is by setting up a model for a time-independent problem, here the Laplace operation is $\triangle = \partial^2 \partial x_1^2 + \partial^2/\partial x_2^2$:

$$\begin{cases} -\triangle u(x) = f(x), \quad x \in B & (1) \\ u(x) = u_{exact}(x), \quad x \in \partial B & (2) \\ f(x) = 8\pi^2 sin(2\pi x_1) sin(2\pi x_2) & (3) \\ u_{exact}(x) = sin(2\pi x_1) sin(2\pi x_2) & (4) \end{cases}$$

Here we form the weak form and discrete the domain and use linear algebra to find the approximate solution for u.

The weak form:

$$-\int_\Omega \triangle uvdx = \int_\Omega fvdx \tag{5}$$

$$\int_\Omega \triangledown u \cdot \triangledown vdx - \int_{\partial\Omega} n \cdot \triangledown uvds = \int_\Omega fvdx \tag{6}$$

$$\int_\Omega \triangledown u \cdot \triangledown vdx = \int_\Omega fvdx \tag{7}$$

And constructing the stiffness matrix A, mass matrix M and load vector b are similar to the 1D FEM analysis, the details of how to construct the matrix in matlab can be found in the appendix.

to solve equation, we use different mesh size: $h_{max} = \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}\}$. and we will find the convergence rate at around 1.4 from Figure 11 which we may conclude that the convergency rate is 1. And we can see the h_{max} vs. h_{max}^γ with $\gamma = 1.4354$ in Figure 12. the solution of the Laplace equation (29) is in Figure 13.

energynorm.png

Figure 11: 2D energy norm with different mesh size

hgamma.png

Figure 12: h_{max} vs. h^{γ}_{max}

2dsolution.png

Figure 13: 2D solution with mesh size of $\frac{1}{2}$ and $\frac{1}{32}$

1.3 2D torus

In 2D analysis for the torus, we start with a unit circle and set the boundary for torus as equation (5). The derivative in time is $\partial_t u$ is approximated using the Crank-Nicolson method, k_i is the time step:

$$M\frac{\xi_i - \xi_{i-1}}{k_i} + A\frac{\xi_i + \xi_{i-1}}{2} = \frac{b_i + b_{i-1}}{2} \tag{8}$$

in the system to solve for the the equation, we have the stiffness matrix (A):

$$A_{ij} = \int_\Omega \triangledown\phi_j \cdot \triangledown\phi_i dx \tag{9}$$

Mass matrix (M):

$$M_{ij} = \int_\Omega \phi_i \cdot \phi_j dx \tag{10}$$

Load vector(b):

$$b_i = \int_\Omega f\phi_i dx \tag{11}$$

Boundary matrix (R):

$$R_{ij} = \int_{\partial\Omega} \gamma \triangledown \phi_i \cdot \triangledown\phi_j ds \tag{12}$$

Boundary vector (r):

$$r_{ij} = \int_{\partial\Omega} \gamma g\phi_i ds \tag{13}$$

For the 2D model, we will iterate each element and build the local matrix and vector and map it to the global matrix and vector, the detailed code are in the appendix. The parameters for the torus are: $\rho = 10$, $R = 0.5$, $r = 0.3$, final time

$= 30$. and we use two different mesh size $h_{max} = \frac{1}{5}$ and $h_{max} = \frac{1}{20}$. and the initial state and the final state using both mesh are shown in Figure 14 and the mass loss computed using $u_{initial} - u_h$ and the mass loss over time is ploted in Figure 15.

torus_2d_solution.png

Figure 14: 2D torus solution with two different mesh size

torus_2d_hormone_emission.png

Figure 15: 2D torus hormone emission

2 3D model

2.1 FEniCS implementation for 2D and 3D mesh

FEniCS (`https://fenicsproject.org/`) is an open source finite element software that can solve partial differential equations in any space dimensions and polynomial degrees. We will implement FEniCS with a 2D mesh for equation (1), (2), (3), and (5) and with the 3D mesh for equation (1), (2), (3), and (4). and the parameters are $\rho = 10$, $R = 0.5$, $r = 0.2$ and $T = 20$. and the results are plot below. in Figure 16 and Figure 17, the visualisation is for the 2D torus at t = 0 and t = 20. in Figure 18 and 19 the 3D torus at t = 0 are plotted along x axis and z axis. and in Figure 20 and Figure 21 are the results at t = 20 for the 3D torus along x axis and z axis. The visualisation for the shape of the torus is in Figure 22.

The tutorial of implementing FEniCS can be found on the website of the FEniCS and for the torus, the detailed code are in the appendix.

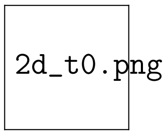

Figure 16: 2D torus at t = 0 using FEniCS

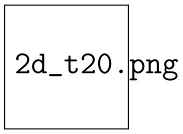

Figure 17: 2D torus at t = 20 using FEniCS

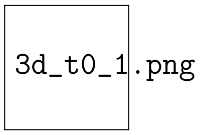

Figure 18: 3D torus at t = 0 along the x axis using FEniCS

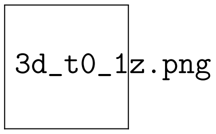

Figure 19: 3D torus at t = 20 along the z axis using FEniCS

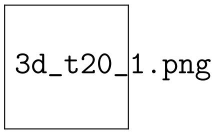

Figure 20: 3D torus at t = 20 along the x axis using FEniCS

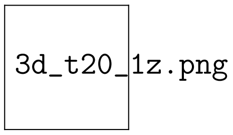

Figure 21: 3D torus at t = 20 along the z axis using FEniCS

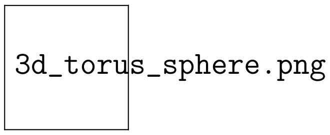

Figure 22: 3D torus view

2.2 3D torus hormone emission analysis

In order to compute the hormone emission from the torus, we can treat the hormone emission as mass loss, at each time level, we compute the mass loss inside the while loop for computing solution at each time step and save it to a file. in this part, we used the mesh sphere1.xml.

$$M = (u_{initial} - u) * dx \tag{14}$$

$$mass = assemble(M) \tag{15}$$

Here three different sets of control parameters are tested and the mass loss are plot in the same figure. the parameters are:

$$\rho = 10, R = 0.5, r = 0.2, T = 50 \tag{16}$$

$$\rho = 20, R = 0.5, r = 0.2, T = 50 \tag{17}$$

$$\rho = 40, R = 0.5, r = 0.2, T = 50 \tag{18}$$

and the results using two different mesh sphere1 and sphere2 are plot separately in Figure 23 and Figure 24. as we can see from both figures, setting a higher ρ will result in a faster mass loss. The equation for the torus is in diffusion. Over time, the hormone inside the torus will diffuse very fast in the beginning and then the speed of mass loss will slow down and get very close to 0 at t_{end}. and the larger the source term (ρ), the faster the mass loss.

mass_loss_mesh1.png

Figure 23: Torus mass loss with different ρ using mesh sphere1.xml

mass_loss_mesh2.png

Figure 24: Torus mass loss with different ρ using mesh sphere2.xml

2.3 Torus optimisation

The last part of designing the torus is that the torus will need to achieve at time $t = \{5, 7, 30\}$ dose amount $M = \{10, 15, 30\}$ mmol. Using scipy.optimize, and we compute the error of target dose at $t = \{5, 7, 30\}$ and the actual dose. with initial data of $[\rho, R, r]$ at $[20, 0.5, 0.1]$, we get the optimised result of the torus at $[40.9, 0.5, 0.3]$.

3 Conclusion

In this project, a medical torus is designed step by step from 1D to 2D and then 3D with optimised dosage. the optimised torus have the following data $[\rho, R, r]$: $[40.9, 0.5, 0.3]$.

4 Reference

1. Larson, M., Bengzon, F., 2013, Umeå, The Finite Element Methods: Theory Implementation and Applications.

2. Nazarov, M., 2017, Uppsala, Applied Finite Element Course material

5 Appendix

5.1 1D code in matlab

- my_stiffness_matrix_assembler.m

- my_load_vector_assembler.m

- laplacian.m

- errorest.m

- f.m

- my_fem_solver.m

5.2 2D code in matlab

5.2.1 2D time independent Laplace equation code in matlab

- my2dsolver.m
- assemble.m
- f1.m
- g1.m

5.2.2 2D torus code in matlab

- partB_2d_torus.m
- assemble.m
- torus_2d.m
- mass_loss.m
- f2.m

- g.m

5.3 3D code in python implement FEniCS

- projectc1_2d.py
- projectc1_3d.py
- projectc2_3d.py
- c2_mass_plot.py
- projectc3_optimization.py