Project

Design a Medical Torus

Yangmei Lin, Haote Liu

January 4, 2024

Contents

1 Introduction

This project is about designing a medical torus that releases hormones through diffusion. The objective is to identify appropriate parameters to shape the torus, control diffusion velocity, and ensure the delivery of a specified dose within a predetermined time frame.

The project consists of three parts. Part A is a simpler 1D model that introduces adaptive mesh refinement based on a posteriori error estimation. Part B involves implementing a piece-wise linear finite element approximation in two dimensions and studying the convergence analysis of a finite element method. Finally, in Part C, we will solve the 3D problem using FEniCS.

Through this project, we can deepen our understanding of utilizing finite element methods for solving physics and numerical models.

2 Part A

In part A, we consider a simpler 1D model and perform adaptive mesh refinement based on a posteriori error estimation.

2.1 Problem 1

Set $e = u - u_h$, then we have:

$$||e^{'}||^2_{L^2(I)} = \int_I e^{'2}\, dx$$

With Galerkin orthogonality:

$$\int_I (u-u_h)^{'} v^{'} = 0,\, \forall v \in \mathbf{V}_{h,0}$$
$$\int_I e^{'} \pi e^{'} dx = 0,\, \pi e \in V_{h,0}$$

Then we can obtain that

$$\begin{aligned} ||e^{'}||^2_{L^2(I)} &= \int_I e^{'2} - e^{'} \pi e^{'}\, dx \\ &= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} e^{'}(e-\pi e)^{'}\, dx \\ &= [IBP] \\ &= \sum_{i=1}^{n} (\int_{x_{i-1}}^{x_i} -e^{''}(e-\pi e)\, dx + [e'(e-\pi e)]_{x_{i-1}}^{x_i}) \end{aligned}$$

Since e and πe coincide at the nodes, so

$$||e^{'}||^2_{L^2(I)} = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} -e^{''}(e - \pi e)\, dx \tag{1}$$

For $-e^{''}$ on I_i (α is the diffusion coefficient, so $\alpha > 0$):

$$-\alpha e^{''} = -\alpha(u - u_h)^{''} = f + \alpha u_h^{''}$$

Plug this equation into (1):

$$\begin{aligned}
\alpha||e^{'}||^2_{L^2(I)} &= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} (f + \alpha u_h^{''})(e - \pi e) dx \\
&\leq \sum_{i=1}^{n} ||f + \alpha u_h^{''}||_{L^2(I_i)} ||e - \pi e||_{L^2(I_i)} \quad \text{(continuous Cauchy-Schwarz inequality)} \\
&\leq \sum_{i=1}^{n} ||f + \alpha u_h^{''}||_{L^2(I_i)} C_0 h_i ||e^{'}||_{L^2(I_i)} \\
&\leq C_0 (h_i^2 \sum_{i=1}^{n} ||f + \alpha u_h^{''}||^2_{L^2(I_i)})^{\frac{1}{2}} (\sum_{i=1}^{n} ||e^{'}||^2_{L^2(I_i)})^{\frac{1}{2}} \quad \text{(discrete Cauchy-Schwarz inequality)} \\
&= C_0 (h_i^2 \sum_{i=1}^{n} ||f + \alpha u_h^{''}||^2_{L^2(I_i)})^{\frac{1}{2}} ||e^{'}||_{L^2(I)}
\end{aligned}$$

Dividing both sides by $\alpha||e^{'}||_{L^2(I)}$ concludes the proof after squaring:

$$||e^{'}||^2_{L^2(I)} \leq (\frac{C_0}{\alpha})^2 \sum_{i=1}^{n} h_i^2 ||f + \alpha u_h^{''}||^2_{L^2(I_i)} \leq C \sum_{i=1}^{n} \eta_i^2$$

2.2 Problem 2

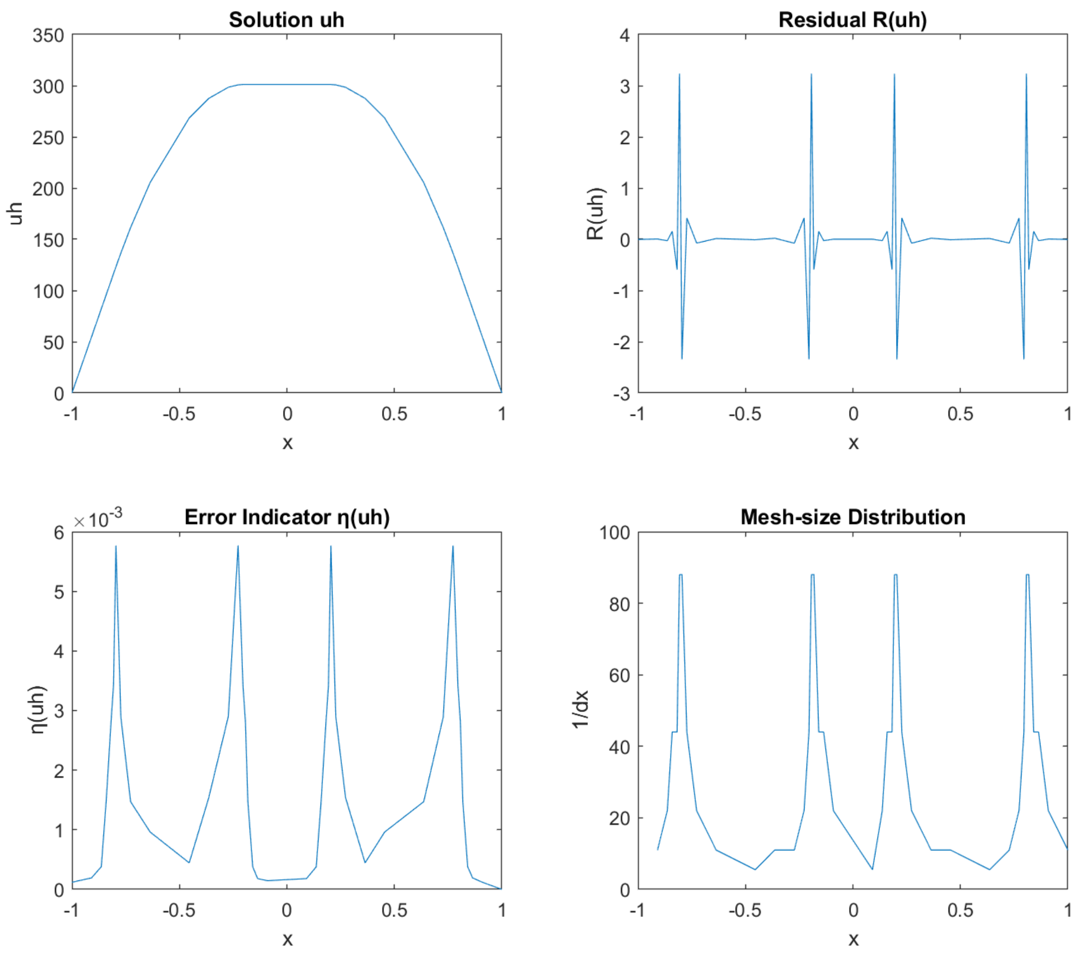


Figure 1: Adaptive Finite Element Analysis

The first sub-figure illustrates the approximation solution u_h for this problem. Through the Residual $R(u_h)$, we observe that the numerical solution is accurate in most intervals but exhibits significant error near the segmentation points of $f(x)$ at $x = \{-0.8, -0.2, 0.2, 0.8\}$. It is also reflected in the mesh-size distribution. Since the algorithm refines the elements with the biggest contribution to the error, most of the new points inserted concentrate on the intervals near the mentioned segmentation points. Ultimately, we achieved an error smaller than TOL when $N = 40$.

3 Part B

In Part B, we need to implement the piece-wise linear finite element approximation in two spatial dimensions and address the time-dependent problem. Subsequently, we will analyze its convergence.

3.1 Problem 1

We consider the following 2D stationary problem:

$$\begin{aligned} -\triangle u(\boldsymbol{x}) &= f(\boldsymbol{x}), \qquad && \boldsymbol{x} \in \mathcal{B} \\ u(\boldsymbol{x}) &= u_{exact}(\boldsymbol{x}), && \boldsymbol{x} \in \partial\mathcal{B} \end{aligned}$$

with $f(\boldsymbol{x}) = 8\pi^2 \sin(2\pi x_1)\sin(2\pi x_2)$ and $u_{exact}(\boldsymbol{x}) = \sin(2\pi x_1)\sin(2\pi x_2)$.

We first derive the variational formulation. We multiply $-\triangle u(\boldsymbol{x}) = f(\boldsymbol{x})$ by a test function v, and integrate using Green's formula:

$$\begin{aligned} \int_{\mathcal{B}} -\triangle uv\, d\boldsymbol{x} &= \int_{\mathcal{B}} fv\, d\boldsymbol{x} \\ \int_{\mathcal{B}} \nabla u \cdot \nabla v\, d\boldsymbol{x} - \int_{\partial\mathcal{B}} \partial_n uv\, ds &= \int_{\mathcal{B}} fv\, d\boldsymbol{x} \end{aligned}$$

Then we see that v and $\triangle v$ should be a square-integrable, and in addition that $v = 0$ on $\partial\mathcal{B}$. Introducing the spaces

$$\begin{aligned} \mathbf{V} &= \{v : ||v||_{L^2(\mathcal{B})} + ||\nabla v||_{L^2(\mathcal{B})} < \infty\} \\ \mathbf{V}_0 &= \{v \in \mathbf{V} : v|_{\partial\mathcal{B}} = 0\} \\ \mathbf{V}_g &= \{v \in \mathbf{V} : v|_{\partial\mathcal{B}} = u_{exact}\} \end{aligned}$$

We obtain the variational formulation:

Find $u \in \mathbf{V}_g$ such that

$$\int_{\mathcal{B}} \nabla u \cdot \nabla v d\boldsymbol{x} = \int_{\mathcal{B}} fv\, d\boldsymbol{x} \quad \forall v \in \mathbf{V}_0$$

Next, we construct the finite element spaces of the continuous piecewise linear polynomial:

$$\begin{aligned} \mathbf{V}_h &:= \{v : v \in C^0(\mathcal{B}), v|_{\mathcal{K}} \in \mathcal{P}^1(\mathcal{K}), \forall \mathcal{K} \in \tilde{\mathcal{L}}_h\}, \quad \mathbf{V}_h \in \mathbf{V} \\ \mathbf{V}_{h,0} &:= \{v \in \mathbf{V}_h : v|_{\partial\mathcal{B}} = 0\} \\ \mathbf{V}_{h,g} &:= \{v \in \mathbf{V}_h : v|_{\partial\mathcal{B}} = u_{exact}\} \end{aligned}$$

Thus we obtain (GFEM):

Find $u_h \in \mathbf{V}_{h,g}$ such that

$$\int_{\mathcal{B}} \nabla u_h \cdot \nabla v d\boldsymbol{x} = \int_{\mathcal{B}} fv\, d\boldsymbol{x} \quad \forall v \in \mathbf{V}_{h,0}$$

By implementing it in Matlab with $h_{max} = \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}\}$, we obtain the numerical convergence rates in the energy norm as $\{0.7429, 1.9856, 1.4346, 1.1710\}$. Figure 2 plots h_{max} versus the energy norm of the error and h_{max} versus h^{γ}_{max}. We can see that these two lines are almost parallel, indicating good convergence performance.

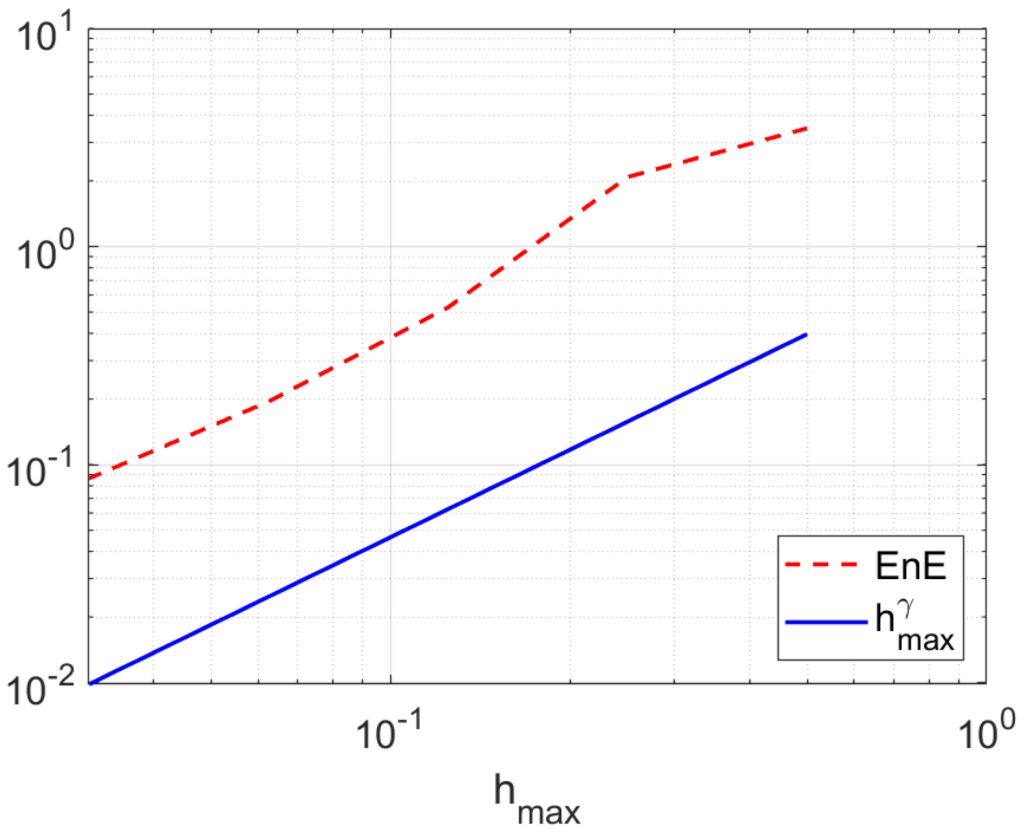


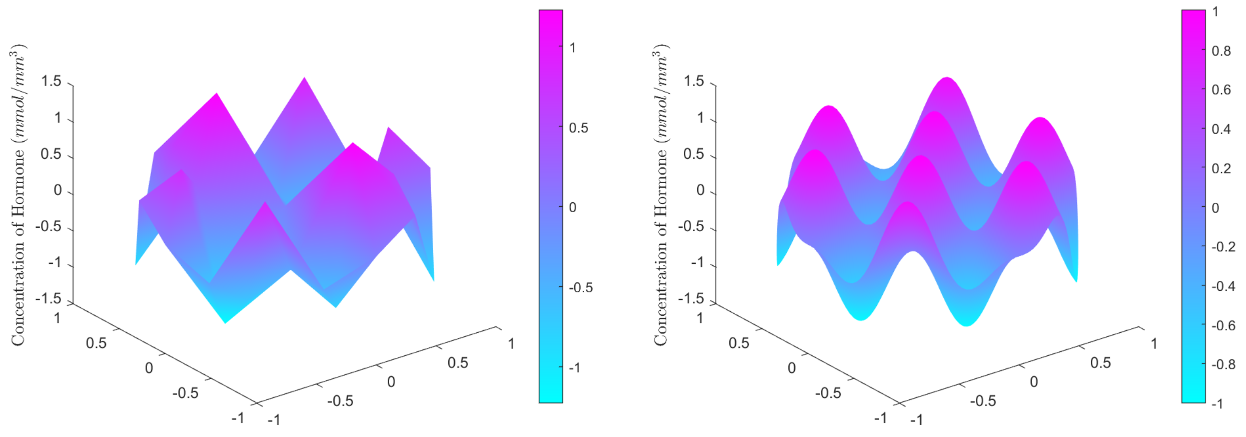
Figure 2: The Convergence Rate Analysis

The convergence rate γ is defined by

$$\gamma_i = \frac{log(\frac{e_{i+1}}{e_i})}{log(\frac{h_{i+1}}{h_i})}$$

where $e = ||u - u_h||_E = (\int_{\mathcal{B}} (\nabla u - \nabla u_h) \cdot (\nabla u - \nabla u_h)\, d\boldsymbol{x})^{\frac{1}{2}}$.

The solutions obtained using the coarsest and the finest meshes are plotted in Figure 3.



(a) Coarsest Meshes, $h_{max} = \frac{1}{2}$ (b) Finest Meshes, $h_{max} = \frac{1}{32}$

Figure 3: Solutions with Different Meshes

3.2 Problem 2

The problem is defined as following:

For $t > 0$

$$
\begin{aligned}
\partial_t u(\boldsymbol{x}, t) - \alpha \triangle u(\boldsymbol{x}, t) &= f(\boldsymbol{x}) \qquad && (\boldsymbol{x}, t) \in \mathcal{B} \times (0, T] \\
u(\boldsymbol{x}, t) &= 0, && (\boldsymbol{x}, t) \in \mathcal{B} \times (0, T] \\
u(\boldsymbol{x}, 0) &= \begin{cases} \rho \quad , & \boldsymbol{x} \in \tau \\ 0 \quad , & \boldsymbol{x} \in \mathcal{B} \setminus \tau \end{cases} && \boldsymbol{x} \in \mathcal{B}
\end{aligned}
$$

Let $\mathbf{V}_0 = \{v(\boldsymbol{x}, t) : ||v(\cdot, t)||^2 + ||\nabla v(\cdot, t)||^2 < \infty, v(\cdot, t) = 0 \ on \ \partial\mathcal{B}, \forall t \geq 0\}$.

Let $\mathbf{V}_{h,0} = \{v(\boldsymbol{x}, t) : v(\boldsymbol{x}, t) \in \mathcal{C}^0(\tilde{\mathcal{L}}_h), v(\boldsymbol{x}, t)|_{\mathcal{K}} \in \mathcal{P}^1(\mathcal{K}), v(\boldsymbol{x}, t) = 0 \ on \ \partial\mathcal{B}, \forall t \geq 0\}$.

(WF) Find $u(\boldsymbol{x}, t) \in \mathbf{V}_0$ such that

$$
\int_{\mathcal{B}} \partial_t u v \, d\boldsymbol{x} - \alpha \int_{\mathcal{B}} \triangle u v \, d\boldsymbol{x} = \int_{\mathcal{B}} f v \, d\boldsymbol{x} \qquad \forall v \in \mathbf{V}_0
$$

$$
\begin{aligned}
\int_{\mathcal{B}} f v \, d\boldsymbol{x} &= \int_{\mathcal{B}} \partial_t u v \, d\boldsymbol{x} + \alpha \int_{\mathcal{B}} \nabla u \nabla v \, d\boldsymbol{x} - \alpha \int_{\partial\mathcal{B}} \partial_n u v \, ds \\
&= \int_{\mathcal{B}} \partial_t u v \, d\boldsymbol{x} + \alpha \int_{\mathcal{B}} \nabla u \nabla v \, d\boldsymbol{x}
\end{aligned}
$$

Since $v = 0$ on $\partial\mathcal{B}$, the boundary term vanished.

(GFEM) Find $u_h(\boldsymbol{x}, t) \in \mathbf{V}_{h,0}$, such that

$$\int_{\mathcal{B}} \partial_t u_h v \, d\boldsymbol{x} + \alpha \int_{\mathcal{B}} \nabla u_h \nabla v \, d\boldsymbol{x} = \int_{\mathcal{B}} f v \, d\boldsymbol{x} \qquad \forall v \in \mathbf{V}_{h,0}$$

Take the usual tent functions $\{\varphi_j(\boldsymbol{x}\}$ as a basis of $\mathbf{V}_{h,0}$ with time dependent coefficients.

For $u_h \in \mathbf{V}_{h,0}$, $\exists\{\xi_j(t)\}$ such that

$$u_h(\boldsymbol{x}, t) = \sum_{N_j \in \mathcal{N}_h \setminus \mathcal{N}_b} \xi_j(t) \varphi_j(\boldsymbol{x})$$

Where $\mathcal{N}_h$ is the set of all nodes on $\tilde{\mathcal{L}}_h$, and $\mathcal{N}_b$ is the set of all boundary nodes on $\tilde{\mathcal{L}}_h$. Insert it into (GFEM) and obtain

$$\sum_{N_j \in \mathcal{N}_h \setminus \mathcal{N}_b} \frac{d\xi_j(t)}{dt} \int_{\mathcal{B}} \varphi_j \varphi_i \, d\boldsymbol{x} + \alpha \sum_{N_j \in \mathcal{N}_h \setminus \mathcal{N}_b} \xi_j(t) \int_{\mathcal{B}} \nabla \varphi_j \nabla \varphi_i \, d\boldsymbol{x} = \int_{\mathcal{B}} f \varphi_i \, d\boldsymbol{x} \quad \forall N_i \in \mathcal{N}_h \setminus \mathcal{N}_b$$

We obtain the following linear system of ordinary differential equations:

$$M \frac{d\boldsymbol{\xi}(t)}{dt} + \alpha A \boldsymbol{\xi}(t) = \boldsymbol{b}$$

$$\boldsymbol{\xi}(0) = \begin{cases} \rho & , \qquad \boldsymbol{x} \in \tau \\ 0 & , \qquad \boldsymbol{x} \in \mathcal{B} \setminus \tau \end{cases} \quad \boldsymbol{x} \in \mathcal{B}$$

where

$$\begin{aligned} M_{i,j} &= \int_{\mathcal{B}} \varphi_j \varphi_i \, d\boldsymbol{x} \\ A_{i,j} &= \int_{\mathcal{B}} \nabla \varphi_j \nabla \varphi_i \, d\boldsymbol{x} \\ b_i &= \int_{\mathcal{B}} f(\boldsymbol{x}) \varphi_i \, d\boldsymbol{x} \qquad N_j, N_i \in \mathcal{N}_h \setminus \mathcal{N}_b \end{aligned}$$

Then approximate $\boldsymbol{\xi}(t)$ on discrete points $0 = t_0 < t_1 < \cdots < t_n = T$ using Crank-Nicolson method, we can obtain:

$$M \frac{\boldsymbol{\xi}^{(n+1)} - \boldsymbol{\xi}^{(n)}}{t_{n+1} - t_n} + \frac{\alpha}{2} A (\boldsymbol{\xi}^{(n+1)} + \boldsymbol{\xi}^{(n)}) = \boldsymbol{b}$$

where $\boldsymbol{\xi}^{(n+1)} = \boldsymbol{\xi}(t_{n+1})$, $\boldsymbol{\xi}^{(n)} = \boldsymbol{\xi}(t_n)$.

After rearranging the terms, we obtain the following equation:

$$(\frac{1}{t_{n+1} - t_n} M + \frac{\alpha}{2} A) \boldsymbol{\xi}^{(n+1)} = (\frac{1}{t_{n+1} - t_n} M - \frac{\alpha}{2} A) \boldsymbol{\xi}^{(n)} + \boldsymbol{b}$$

3.3 Problem 3

In this problem, we implement a numerical integration in 2D to compute the mass loss.

$$M(t) = \int_{\mathcal{B}} (u_0(\boldsymbol{x}) - u(\boldsymbol{x}, t))\, d\boldsymbol{x}$$

For computing it, we use the Trapezoidal rule in each element $\mathbf{K}_i$:

$$\int_{\mathbf{K}_i} (u_0(\boldsymbol{x}) - u(\boldsymbol{x}, t)) \approx \frac{|\mathbf{K}_i|}{3} \sum_{i=1}^{3} (u_0(N_i) - u(N_i, t))$$

The control parameters are set as $f(\boldsymbol{x}) = 0$, $\rho = 10$, $R = 0.5$, $r = 0.3$, $T = 30$. So we have the following iteration equation:

$$(\frac{1}{t_{n+1} - t_n} M + \frac{\alpha}{2} A)\boldsymbol{\xi}^{(n+1)} = (\frac{1}{t_{n+1} - t_n} M - \frac{\alpha}{2} A)\boldsymbol{\xi}^{(n)}$$

The amount of emitted hormone versus time is plotted in Figure 4 with $h_{max} = \frac{1}{5}$ and $h_{max} = \frac{1}{20}$ respectively.

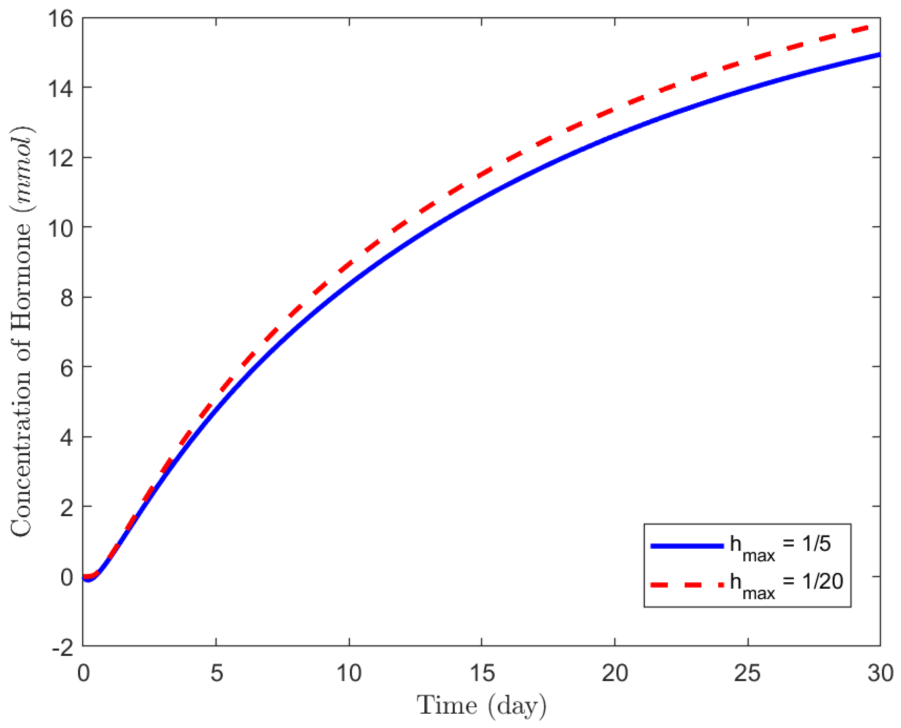
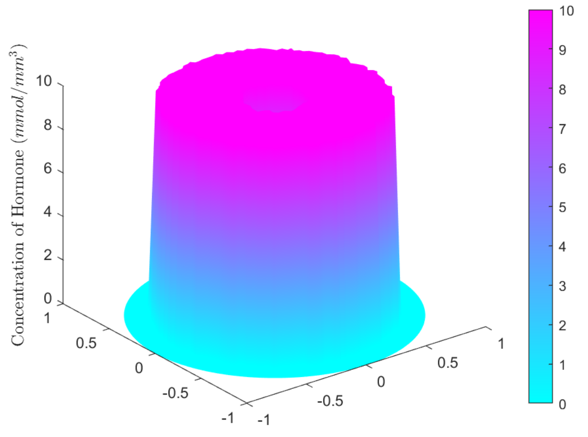
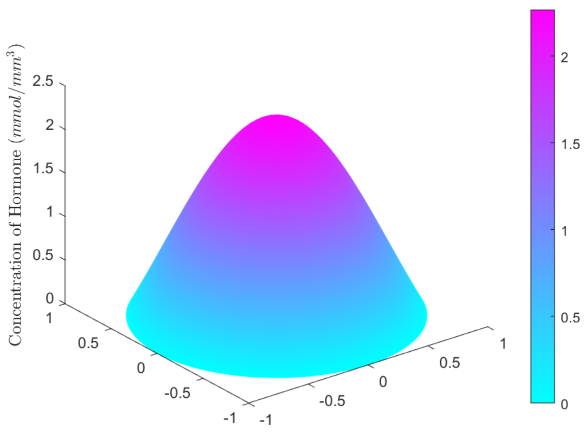


Figure 4: The Amount of Emitted Hormone with $h_{max} = \frac{1}{5}$ and $h_{max} = \frac{1}{20}$

Figure 5 displays the solutions at the initial time and the final time with $h_{max} = \frac{1}{20}$.



(a) $T = 0$



(b) $T = 30$

Figure 5: Solutions at $T = 0$ and $T = 30$ for the Fine Mesh

3.4 Problem 4

In this problem, we need to solve a heat equation with a reaction term:

$$\partial_t u(\boldsymbol{x}, t) - \alpha \triangle u(\boldsymbol{x}, t) = \beta(1-\gamma u)u, \quad (\boldsymbol{x}, t) \in \mathcal{B} \times (0, T]$$

(GFEM)

$$\int_{\mathcal{B}} \partial_t u_h v \, d\boldsymbol{x} + \alpha \int_{\mathcal{B}} \nabla u_h \nabla v \, d\boldsymbol{x} = \int_{\mathcal{B}} \beta(1-\gamma u) u v \, d\boldsymbol{x}$$

For the right-hand side, we write it as $S(u) = \beta(1-\gamma u)u$. Since $S(u)$ is nonlinear, we take a linear interpolant of it to compute it using the solution from the previous time step:

$$S(u) \approx \pi_h S = \sum_{j=1}^{\mathcal{N}} S_j \varphi_j = \sum_{j=1}^{\mathcal{N}} \beta(1-\gamma u_j) u_j \varphi_j$$

where $\mathcal{N}$ is the total number of nodes. And we know that u_h can be expressed as $u_h = \sum_{j=1}^{\mathcal{N}} \xi_j(t)\varphi_j$. After plugging them into the (GFEM), we obtain:

$$\sum_{j=1}^{\mathcal{N}} \frac{d\xi_j(t)}{dt} \int_{\mathcal{B}} \varphi_j \varphi_i \, d\boldsymbol{x} + \alpha \sum_{j=1}^{\mathcal{N}} \xi_j(t) \int_{\mathcal{B}} \nabla\varphi_j \nabla\varphi_i \, d\boldsymbol{x} = \sum_{j=1}^{\mathcal{N}} \beta(1-\gamma u_j) u_j \int_{\mathcal{B}} \varphi_j \varphi_i \, d\boldsymbol{x}$$

Following is the linear system form:

$$M \frac{d\boldsymbol{\xi}(t)}{dt} + \alpha A \boldsymbol{\xi}(t) = M(\beta \boldsymbol{\xi}^{(\boldsymbol{n})} - \beta\gamma \boldsymbol{\xi}^{(\boldsymbol{n})} \circ \boldsymbol{\xi}^{(\boldsymbol{n})})$$

where '$\circ$' represent Hadamard product.

Then approximate $\boldsymbol{\xi}(t)$ of the left-hand side using Crank-Nicolson method. For the right side hand, we use the solution from last time step:

$$M\frac{\boldsymbol{\xi}^{(n+1)} - \boldsymbol{\xi}^{(n)}}{t_{n+1} - t(n)} + \frac{\alpha}{2}A(\boldsymbol{\xi}^{(n+1)} + \boldsymbol{\xi}^{(n)}) = \beta M\boldsymbol{\xi}^{(n)} - \beta\gamma M\boldsymbol{\xi}^{(n)} \circ \boldsymbol{\xi}^{(n)}$$

After rearranging the terms, we obtain the iteration equation:

$$(\frac{M}{t_{n+1} - t_n} + \frac{\alpha}{2}A)\boldsymbol{\xi}^{(n+1)} = (\frac{M}{t_{n+1} - t_n} - \frac{\alpha}{2}A)\boldsymbol{\xi}^{(n)} + \beta M\boldsymbol{\xi}^{(n)} - \beta\gamma M\boldsymbol{\xi}^{(n)} \circ \boldsymbol{\xi}^{(n)}$$

With $\beta = 0.2$, $\gamma = 0.5$, we have following results:

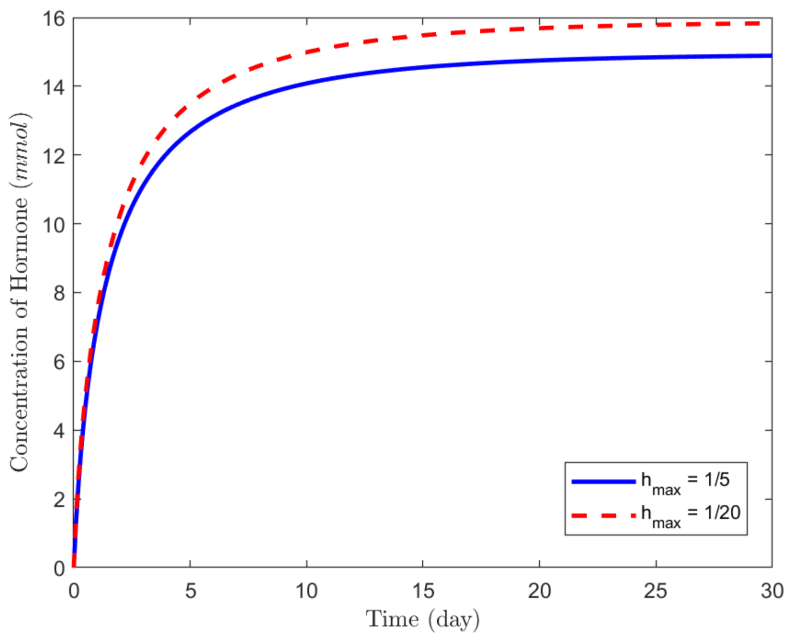
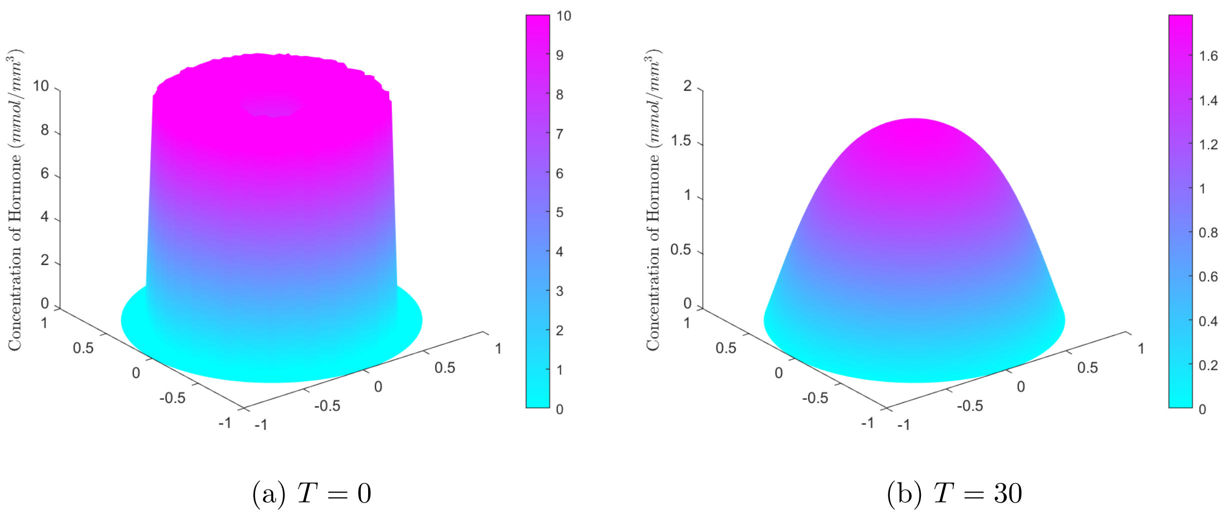


Figure 6: The Amount of Emitted Hormone with $h_{max} = \frac{1}{5}$ and $h_{max} = \frac{1}{20}$



(a) $T = 0$

(b) $T = 30$

Figure 7: Solutions at $T = 0$ and $T = 30$ for the Fine Mesh

With $\beta = 1$, $\gamma = 0.2$, we have following results:

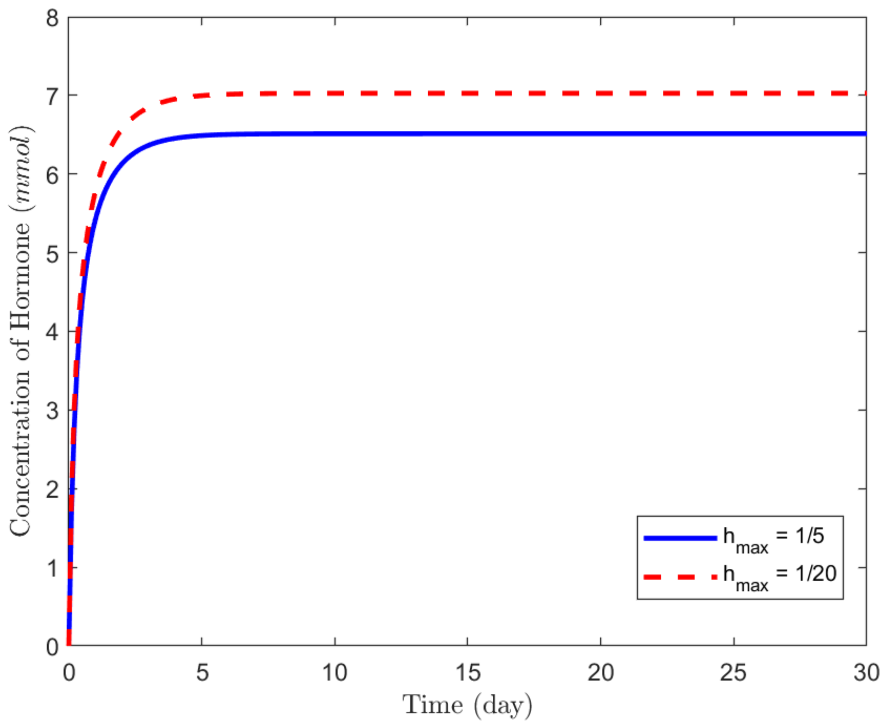
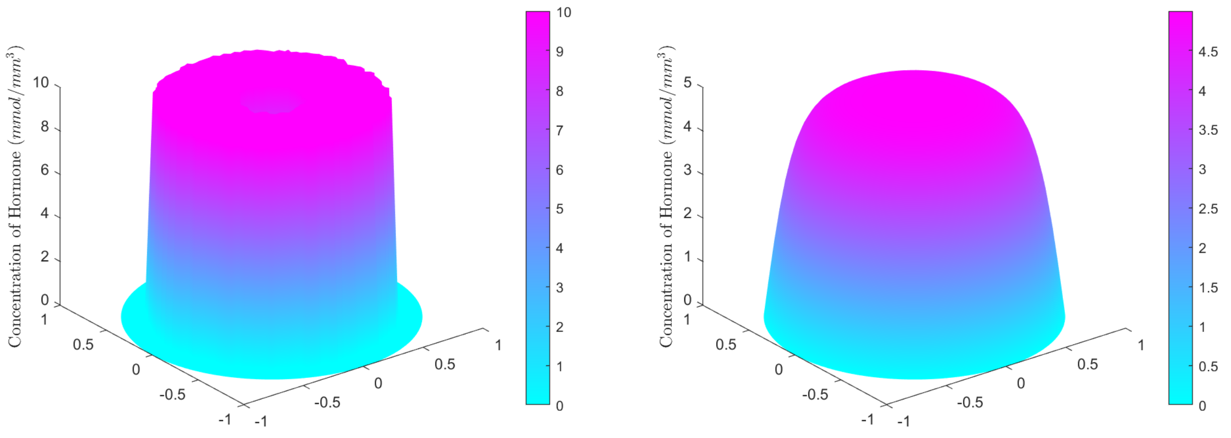


Figure 8: The Amount of Emitted Hormone with $h_{max} = \frac{1}{5}$ and $h_{max} = \frac{1}{20}$

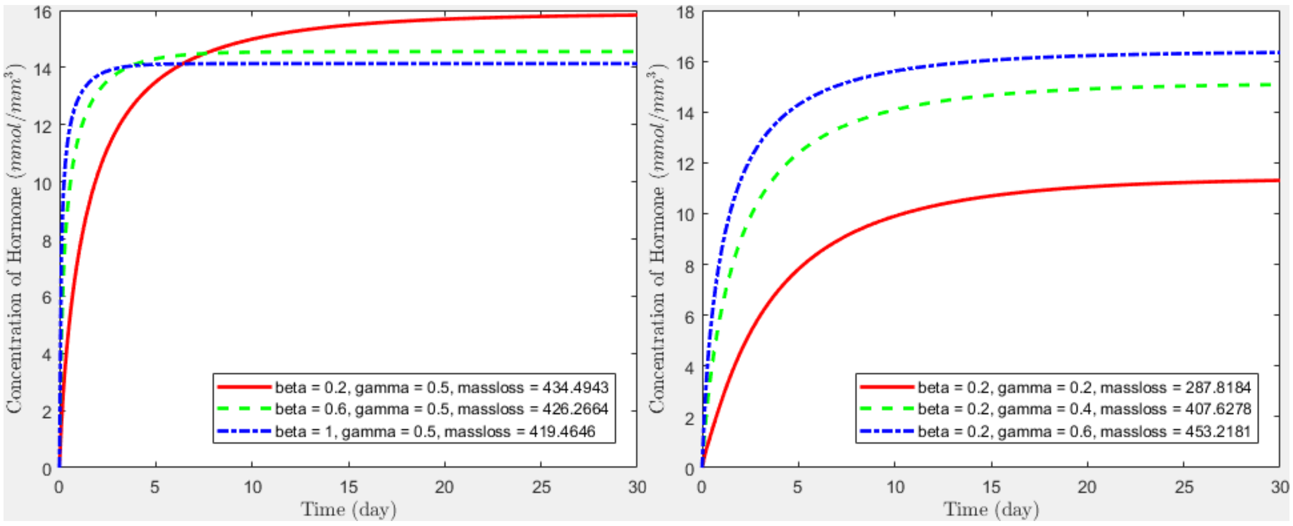


(a) $T = 0$

(b) $T = 30$

Figure 9: Solutions at $T = 0$ and $T = 30$ for the Fine Mesh

To study how changes in the β value affect the end result(Mass loss is calculated by using the trapz-function for numerical integration), we fixed γ and $h_{max} = 0.2$, then varied the β value and plotted the resulting outcomes. Similarly, by varying γ while keeping other parameters constant, we obtained the following plot:



(a) Change in β (b) Change in γ

Figure 10: How changes in β or γ affect the end result

From the figure above, we can observe that the speed of mass loss is positively correlated with the values of β and γ, meaning that the speed of mass loss increases as the values of β and γ increase. However, the total mass loss is positively correlated with the value of γ and negatively correlated with the value of β (although it is not a significant amount of change). This indicates that the total mass loss increases as the γ value increases and as the β value decreases.

4 Part C

In this part, we consider the full 3D model and solve the problems using FEniCS.

4.1 Problem 1

The problem is defined as following:

$$\begin{aligned} \partial_t u(\boldsymbol{x},t) - \alpha \triangle u(\boldsymbol{x},t) &= f(\boldsymbol{x}) \qquad && (\boldsymbol{x},t) \in \mathcal{B} \times (0,T] \\ u(\boldsymbol{x},t) &= 0, && (\boldsymbol{x},t) \in \mathcal{B} \times (0,T] \\ u(\boldsymbol{x},0) &= \begin{cases} \rho \quad , & \boldsymbol{x} \in \mathcal{T} \\ 0 \quad , & \boldsymbol{x} \in \mathcal{B} \setminus \mathcal{T} \end{cases} && \boldsymbol{x} \in \mathcal{B} \end{aligned}$$

where $f(x) = 0$. And $\mathcal{T}$ denotes the torus with the major radius R and the minor radius r, $r < R$:

$$\mathcal{T} := \begin{cases} (R - (x_1^2 + x_2^2)^{\frac{1}{2}})^2 + x_3^2 \leq r^2, \text{if } x \in \mathbf{R}^3 \\ |R - (x_1^2 + x_2^2)^{\frac{1}{2}}| \leq r, \text{if } x \in \mathbf{R}^2 \end{cases}$$

We set $\rho = 10$, $R = 0.5$, $r = 0.2$, and $T = 20$. Discrete the equation in time using the backward difference and get the following equation:

$$\int_{\mathcal{B}} (uv + \triangle t \nabla u \cdot \nabla v) dx = \int_{\mathcal{B}} u^{(n)} v dx$$

Figure 11 and Figure 12 show the solutions at T=0 and T=20 for the 2D mesh.

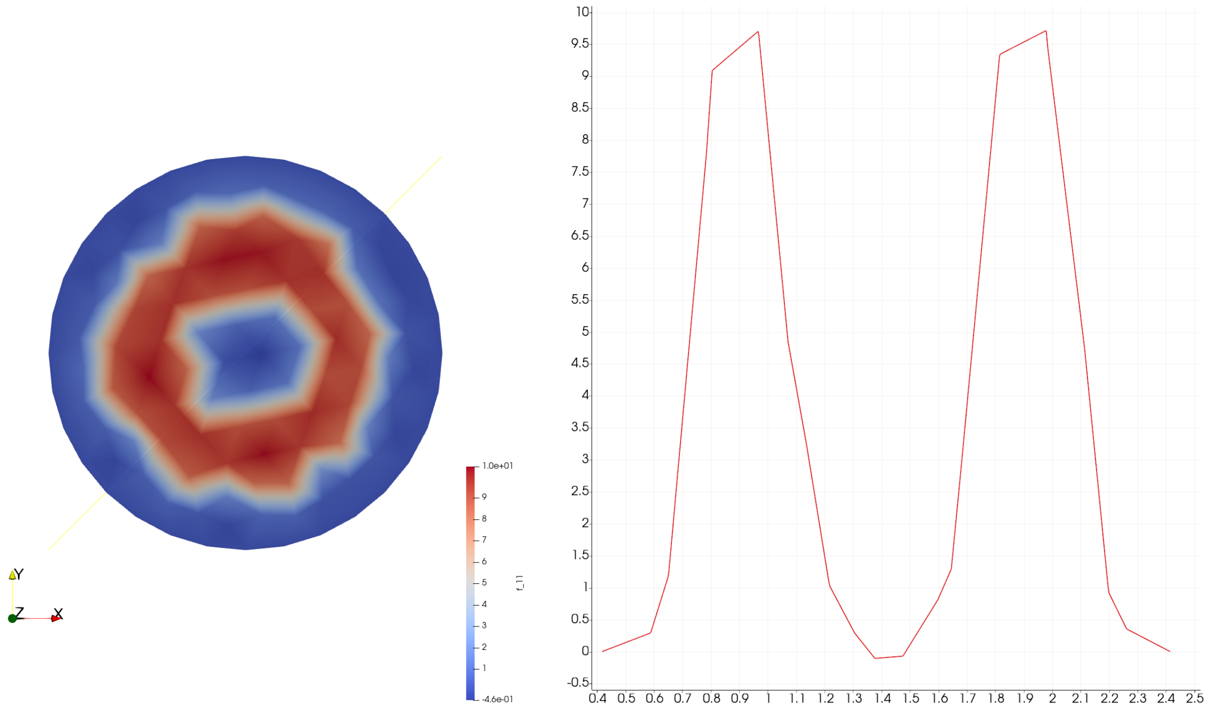


Figure 11: Solutions for the 2D mesh at $T = 0$, with a 2D image on the left and a plot along the diagonal line on the right

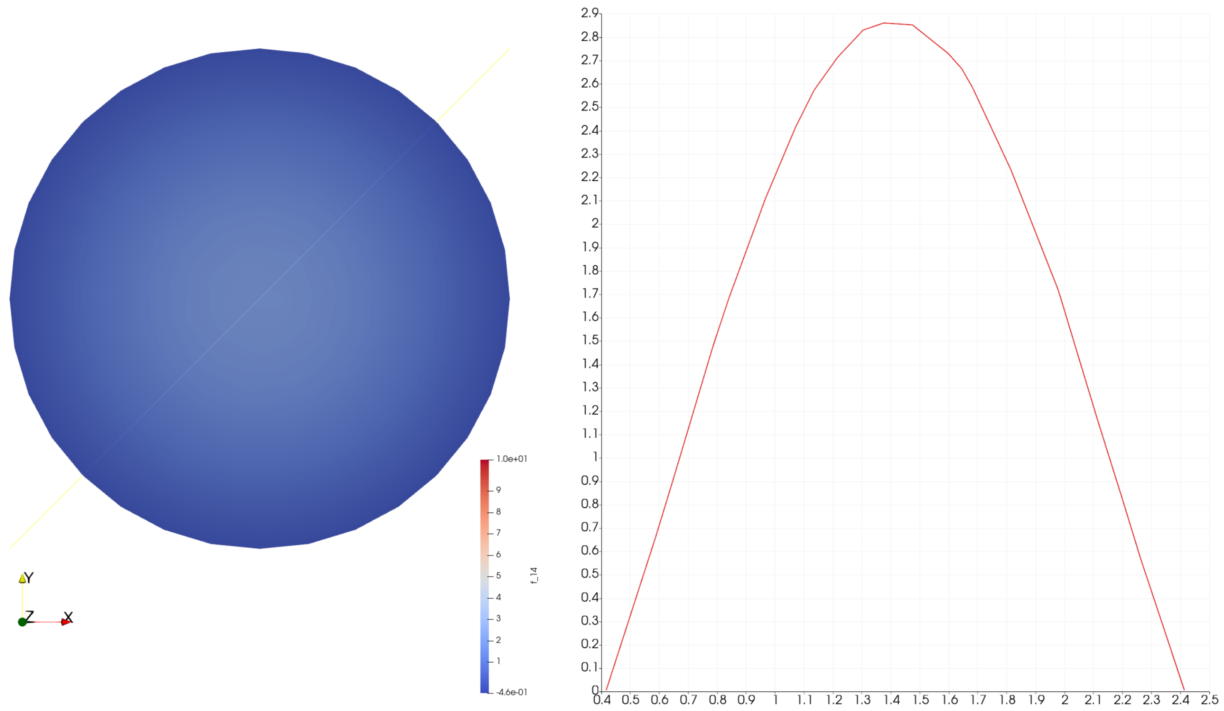


Figure 12: Solutions for the 2D mesh at $T = 20$, with a 2D image on the left and a plot along the diagonal line on the right

And Figure 13 and Figure 14 show the solution at T=0 and T=20 for the 3D mesh.

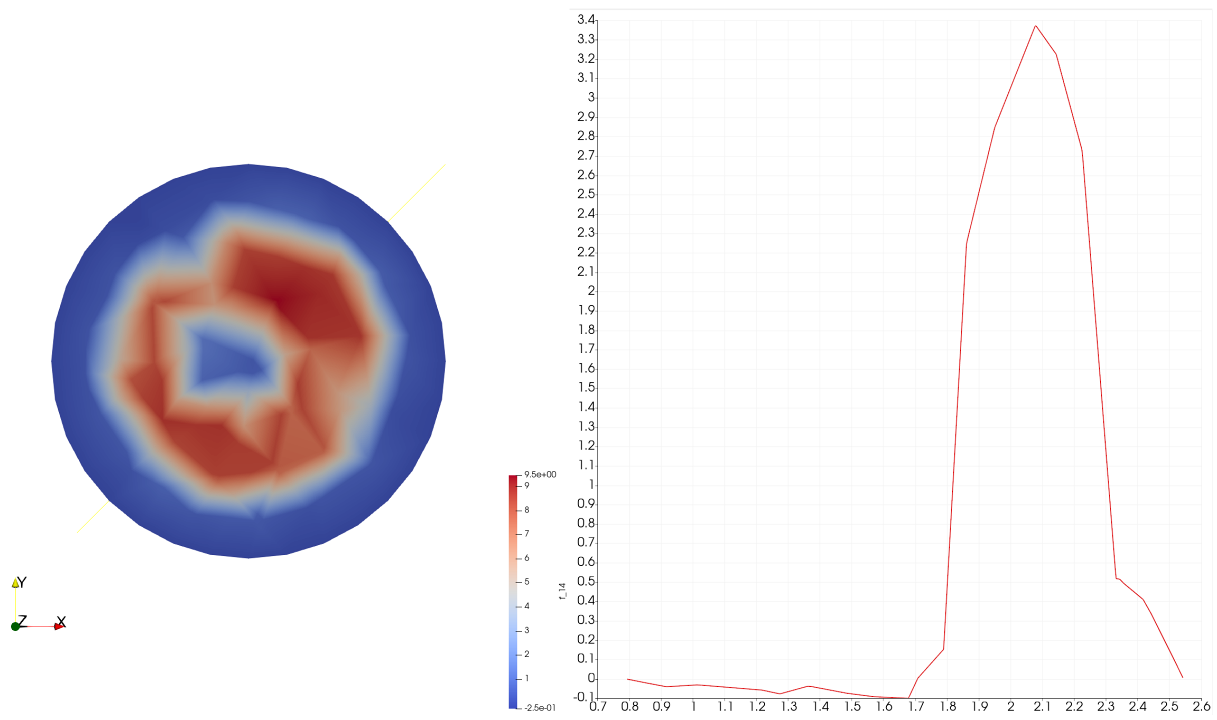


Figure 13: Solutions for the 3D mesh at $T = 0$, with a 3D image on the left and a plot along the diagonal line on the right

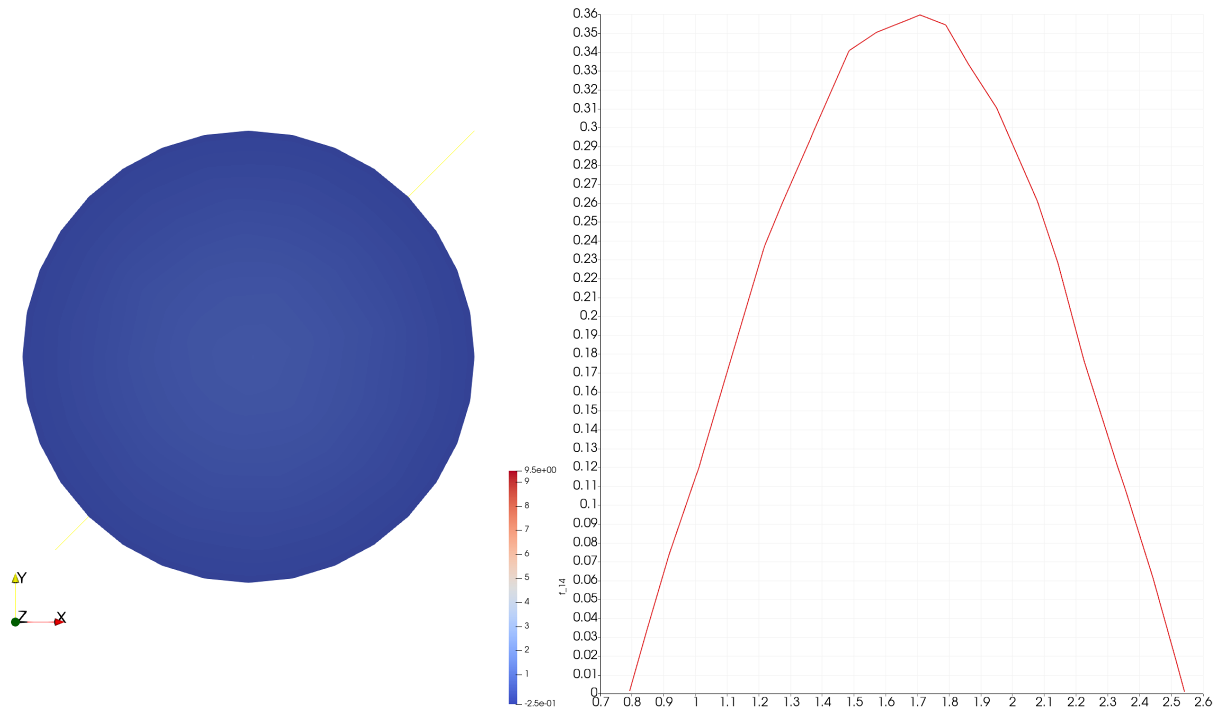


Figure 14: Solutions for the 3D mesh at $T = 20$, with a 3D image on the left and a plot along the diagonal line on the right

According to the plots above, we can see that the concentration is diffusing out from the boundary, and at the final time, the magnitude is very small.The hottest point (along the diagonal line) at the end time $T = 20$ is approximately 10% of the initial hottest point, both in 2D and 3D.

4.2 Problem 2

In this problem, we compute the mass loss with different magnitude ρ inside the torus. The results are illustrated as follows:

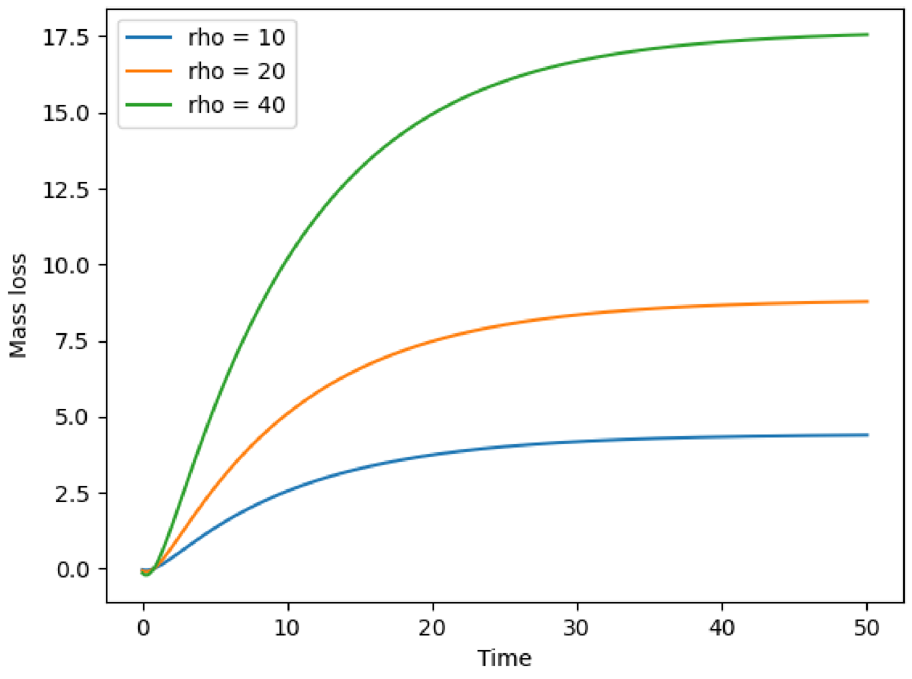


Figure 15: Mass Loss as a Function of Time for Different ρ values

The last parameter set, characterized by the highest initial density ($\rho = 40$), exhibits the fastest mass loss. Mass loss is directly proportional to the concentration gradient, and a higher initial density results in a steeper gradient. Therefore, substances diffuse more rapidly when their initial density is higher.

4.3 Problem 3

Finally, we are going to find the optimal parameters for the design of the torus. We have following objective function(least square problem with constrain):

$$\begin{cases} \min F(\rho, R, r) & = \sum_{i=0}^{2} (M(T_i) - M_i)^2 \\ 0 < r < R \end{cases}$$

where $M(T_i)$ is the numerical calculation of mass loss at time T_i, with $T_i \in [0, 30]$, and M_i is the expected mass loss, listed as $[10, 15, 30]$, corresponding to day i in the list $[5, 7, 30]$.

To optimizing the above equation we had used `'minimize'` from the Python package `'scipy.optimize'` with a tolerance of 0.001. We started with initial guess $\rho = 20, R = 0.5, r = 0.1$, after 55 iterations we obtain $\rho \approx 40.90$, $R \approx 0.50$ and $r \approx 0.30$. The corresponding plot of total mass loss as function with respect to time, using these parameters, is shown below.

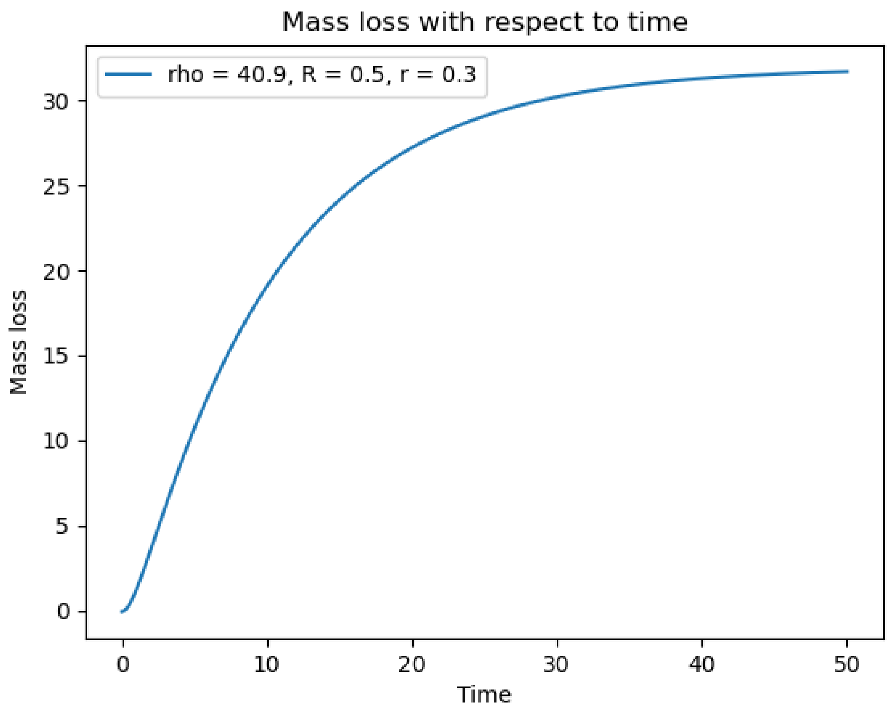


Figure 16: Mass Loss as a Function of Time that Meets the Requirements

5 Concluding Discussion

The aim of this project is to investigate the parameters ρ, β, γ, R and r affect the diffusion process and the mass loss of a hormone in a torus.

We found that the diffusion process and the mass loss are influenced by all the parameters, but in different ways. The initial density ρ has a positive effect on both the speed and the total amount of mass loss, as expected from the theory of diffusion. The parameters β and γ, which represent the nonlinear effects of the hormone on itself, have opposite effects on the total mass loss: β has a negative effect and γ has a positive effect. The shape and size of the torus, determined by R and r, also affect the diffusion process and the mass loss. These dimensions determine the injection of the total amount of hormones, given that the source term is zero.

Finally, after gaining an understanding of the key parameters, we were able to use an optimization algorithm to find the optimal values of ρ, R, and r, given constraints on the amount of mass loss at specific times.

Attached Code

Problem A.2

```
a = -1;
b = 1;
gl = 0;
gr = 0;
alpha = 0.01;

beta = 0.9;
N = 11;
TOL = 1e-3;
Nmax = 1e4;
h = (b - a) / N;
x = a:h:b;

while true
    N = length(x);
    zeta = my_discrete_Lapl(x, @fStar, gl, gr);
    eta2 = zeros(N, 1);
    for j = 1 : N-1
        h = x(j+1)- x(j);
        a = f(x(j)) + alpha*zeta(j);
        b = f(x(j+1)) + alpha*zeta(j+1);
        t = (a.^2+b.^2)*h/2;
        eta2(j) = h.^2 * t;
    end

    if sum(eta2) <= TOL || N > Nmax
        break;
    end

    for k = 1 : (length(eta2)-1)
        if eta2(k) > beta*max(eta2)
            x = [x (x(k+1)+x(k))/2];
        end
    end
    x = sort(x);
end
```

```matlab
A = my_stiffness_matrix_assemble(x);
B = my_load_vector_assemble(x, @fStar, gl, gr);
xi = A \ B;
Re = f(x) + alpha * zeta';
eta = eta2.^0.5;

figure;
subplot(2, 2, 1);
plot(x, xi);
title('Solution uh');
xlabel('x');
ylabel('uh');

% Plot the residual R(uh)
subplot(2, 2, 2);
plot(x, Re);
title('Residual R(uh)');
xlabel('x');
ylabel('R(uh)');

% Plot the error indicator eta(uh)
subplot(2, 2, 3);
plot(x, eta);
title('Error Indicator eta(uh)');
xlabel('x');
ylabel('eta(uh)');

% Plot the mesh-size distribution
subplot(2, 2, 4);
plot(x(2:end), 1./diff(x));
title('Mesh-size Distribution');
xlabel('x');
ylabel('1/dx');

disp(size(x));


function y = f(x)
    condition = ((x <= 0.8) & (x >= 0.2)) | ((x <= -0.2) & (x >=
       -0.8));
    y = zeros(size(x));
    y(condition) = 10;
end

function y = fStar(x)
    condition = ((x <= 0.8) & (x >= 0.2)) | ((x <= -0.2) & (x >=
       -0.8));
    y = zeros(size(x));
    y(condition) = 10 / 0.01;
end

function zeta = my_discrete_Lapl(x, ~, gl, gr)
    A = my_stiffness_matrix_assemble(x);
    B = my_load_vector_assemble(x, @fStar, gl, gr);
    xi = A \ B;
    M = my_mass_matrix_assemble(x);
    zeta = -inv(M) * A * xi;
end

function A = my_stiffness_matrix_assemble(x)

    N = length(x) - 1;
    A = zeros(N+1, N+1);

    for i = 1 : N
        h = x(i+1) - x(i);
        n = [i i+1];
        A(n,n) = A(n,n) + [1 -1; -1 1]/h;
    end

    % adjust for BC
    A(1,1) = 1;
    A(1,2) = 0;
    A(N+1,N) = 0;
    A(N+1,N+1) = 1;

end

function b = my_load_vector_assemble(x, f, gl, gr)
    %
    % Return assembled load vector b
    % Input vector x of node coords
    %
    N = length(x) - 1;
    b = zeros(N+1, 1);
    for i = 1 : N
        h = x(i+1) - x(i);
        n = [i i+1];
        b(n) = b(n) + [f(x(i)); f(x(i+1))]*h/2;
    end
    b(1) = gl;
    b(N+1) = gr;

end

function M = my_mass_matrix_assemble(x)
    N = length(x)-1;
    M = zeros(N+1, N+1);

    for i = 1 : N
        h = x(i+1) - x(i);
        n = [i i+1];
        M(n,n) = M(n,n) + [1/3 1/6; 1/6 1/3]*h;
    end
end
```

Problem B.1

```
clc
clear
g = @circleg;

H = [1/2 1/4 1/8 1/16 1/32];

energyNorm = zeros(length(H),1);
EnE = zeros(length(H),1);

for i = 1:length(H)
    hm = H(i);

    [p, e, t] = initmesh(g, 'hmax', hm);
    g1 = u(p(1, e(1,:)), p(2, e(1,:)));
    A = StiffnessAssembler2D(p, t, @(x, y) 1);
    b = LoadAssembler2D(p, t, @f);
    I = eye(length(p));

    A(e(1,:),:) = I(e(1,:),:);
    b(e(1,:))= g1;

    uh = A\b;

    err = u(p(1,:), p(2,:)) - uh';
    EnE(i) = sqrt(err * A * err');

    if i == 1
        figure;
        pdeplot(p, e, t, 'XYData', uh, 'ZData', uh);
        % title("Solution u with the coarsest meshes")
    elseif i == length(H)
        figure;
        pdeplot(p, e, t, 'XYData', uh, 'ZData', uh);
        % title("Solution u with the finest meshes")
    end
end

gamma = zeros((length(H)-1),1);
```

```matlab
for i=1:length(energyNorm)-1
    gamma(i) = log(EnE(i+1)/EnE(i))/log(H(i+1)/H(i));
    disp(gamma(i));
end

figure;
loglog(H, EnE, 'r--', 'LineWidth', 1.5);
hold on;
loglog(H, H.^mean(gamma), 'b-', 'LineWidth', 1.5);
xlabel('h_{max}');
grid on;
legend('EnE', ['h_{max}^{\gamma}, \gamma = ' num2str(gamma(end))
   ], 'Location', 'southeast');
set(gca, 'FontSize', 14);
set(legend, 'FontSize', 14);
hold off;

function z = f(x, y)
    z = 8 * pi^2 * sin(2 * pi * x) .* sin(2 * pi * y);
end

function z = u(x, y)
    z = sin(2 * pi * x) .* sin(2 * pi * y);
end
```

Problem B.3

```
clc
clear

alpha = 0.01;

T = 30; % final time

dt = 0.1; % time step
tn = T / dt; % number of time levels
MLoss = zeros(tn+1, 2);

g = @circleg;

H = [1/5, 1/20];

for index = 1 : length(H)
    hmax = H(index); % mesh size
    [p, e, t] = initmesh(g, 'hmax', hmax);
    nt = size(t, 2); % number of elements

    A = StiffnessAssembler2D(p, t, @(x, y) alpha);
    M = MassAssembler2D(p, t);
    I = eye(length(p));

    old_xi = u0(p(1,:), p(2,:))';
    for i = 1: tn
        % impose Dirichlet Boundary Condition
        D = ((1/dt).*M + (1/2).*A);
        D(e(1,:),:) = I(e(1,:),:);

        b = ((1/dt).*M - (1/2).*A)*old_xi;
        b(e(1,:),:) = 0;

        xi = D \ b;
        % compute the mass loss
        for K = 1:nt
            triangle = t(1:3, K);
            x = p(1, triangle);
            y = p(2, triangle);
            [area, ~, ~] = HatGradients(x, y);
            MLoss(i+1, index) = MLoss(i+1, index) + area/3 * (sum
               (u0(x,y))- sum(xi(triangle)));
            % initial = initial +  area/3 * sum(u0(x,y));
        end
        % disp(initial);
        old_xi = xi;
    end
end

figure;
% h_max = 1/5
plot(linspace(0, T, tn + 1), MLoss(:,1), 'DisplayName', 'h_{max}
   = 1/5', 'LineWidth', 2, 'Color', 'b');
hold on;
% h_max = 1/20
plot(linspace(0, T, tn + 1), MLoss(:,2), 'DisplayName', 'h_{max}
   = 1/20', 'LineWidth', 2, 'Color', 'r', 'LineStyle', '--');
hold off;
% add legend
legend('show');
xlabel('Time (day)', 'Interpreter', 'latex');
ylabel('Concentration of Hormone ($mmol/mm^3$)', 'Interpreter', '
   latex');

figure;
pdeplot(p, e, t, 'XYData', u0(p(1,:), p(2,:)), 'ZData', u0(p(1,:)
   , p(2,:)));

figure;
pdeplot(p, e, t, 'XYData', xi, 'ZData', xi);
```

Problem B.4

```
clc
clear

alpha = 0.01;
% beta = 0.2;
% gamma = 0.5;

beta = 1;
gamma = 0.2;

T = 30; % final time

dt = 0.1; % time step
tn = T / dt; % number of time levels
MLoss = zeros(tn+1, 2);

g = @circleg;

H = [1/5, 1/20];

for index = 1 : length(H)
    hmax = H(index); % mesh size
    [p, e, t] = initmesh(g, 'hmax', hmax);
    nt = size(t, 2); % number of elements

    A = StiffnessAssembler2D(p, t, @(x, y) alpha);
    M = MassAssembler2D(p, t);
    I = eye(length(p));

    A(e(1,:),:) = I(e(1,:),:);

    old_xi = u0(p(1,:), p(2,:))';
    for i = 1: tn
        D = ((1/dt).*M + (1/2).*A);
        D(e(1,:),:) = I(e(1,:),:);
        b = (((1/dt).*M - (1/2).*A)*old_xi + beta.*M*old_xi -
           beta*gamma.*M*(old_xi.*old_xi));
        b(e(1,:),:) = 0;
        xi = D \ b;
        % compute the mass loss
        for K = 1:nt
            triangle = t(1:3, K);
            x = p(1, triangle);
            y = p(2, triangle);
            [area, ~, ~] = HatGradients(x, y);
            MLoss(i+1, index) = MLoss(i+1,index) + area/3 * (sum(
               u0(x,y))- sum(xi(triangle)));
        end
        old_xi = xi;
    end
end

figure;
% h_max = 1/5
plot(linspace(0, T, tn + 1), MLoss(:,1), 'DisplayName', 'h_{max}
   = 1/5', 'LineWidth', 2, 'Color', 'b');
hold on;
% h_max = 1/20
plot(linspace(0, T, tn + 1), MLoss(:,2), 'DisplayName', 'h_{max}
   = 1/20', 'LineWidth', 2, 'Color', 'r', 'LineStyle', '--');
hold off;
% add legend
legend('show');
xlabel('Time (day)', 'Interpreter', 'latex');
ylabel('Concentration of Hormone ($mmol/mm^3$)', 'Interpreter', '
   latex');

figure;
pdeplot(p, e, t, 'XYData', u0(p(1,:), p(2,:)), 'ZData', u0(p(1,:)
   , p(2,:)));

figure;
pdeplot(p, e, t, 'XYData', xi, 'ZData', xi);
```

Functions for 2D Problems

```
function y = u0(x1, x2)
    % rho = 10; R = 0.5; r = 0.3;
    condition = ((x1.^2 + x2.^2) <= 0.64 & (x1.^2 + x2.^2) >=
       0.04);
    y = zeros(size(x1));  % Fix the size of x1, not x
    y(condition) = 10;
end

function A = StiffnessAssembler2D(p, t, a)
    np = size(p, 2);
    nt = size(t, 2);

    A = sparse(np, np);
    for K = 1:nt
        loc2glb = t(1:3, K);
        x = p(1, loc2glb);
        y = p(2, loc2glb);
        [area, b, c] = HatGradients(x, y);

        xc = mean(x);
        yc = mean(y);

        abar = a(xc, yc);

        AK = abar * ( b*b' + c*c') * area;

        A(loc2glb, loc2glb) = A(loc2glb, loc2glb) + AK;

    end
end

function M = MassAssembler2D(p,t)
    np = size(p,2); % number of nodes
    nt = size(t,2); % number of elements
    M = sparse(np,np); % allocate mass matrix

    for K = 1:nt % loop over elements
        loc2glb = t(1:3, K); % local-to-global map
        x = p(1, loc2glb); % node x-coordinates
        y = p(2, loc2glb); % node y-coordinates
        area = polyarea(x,y); % triangle area

        MK = [2 1 1;
              1 2 1;
              1 1 2]/12*area; % element mass matrix
        % add element masses to M
        M(loc2glb, loc2glb) = M(loc2glb, loc2glb) + MK;

end

function b = LoadAssembler2D(p,t,f)
    np = size(p,2);
    nt = size(t,2);
    b = zeros(np, 1);

    for K = 1:nt
        loc2glb = t(1:3, K);
        x = p(1, loc2glb);
        y = p(2, loc2glb);
        area = polyarea(x,y);

        bK = [f(x(1), y(1));
              f(x(2), y(2));
              f(x(3), y(3))]/3*area;
        b(loc2glb) = b(loc2glb) + bK;
    end
end

function [area, b, c] = HatGradients(x, y)
    area = polyarea(x, y);
    b = [y(2)-y(3); y(3)-y(1); y(1)-y(2)] / 2 / area;
    c = [x(3)-x(2); x(1)-x(3); x(2)-x(1)] / 2 / area;
end
```

Problem C.1 2D mesh

```
from dolfin import *

# Create mesh and define function space
mesh = Mesh("circle1.xml")
Q = FunctionSpace(mesh, "CG", 1)

# Define parameters
T = 20  # End time
h = mesh.hmin()
dt = h
alpha = 0.01

# Create subdomain for Dirichlet boundary
class DirichletBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary

# Set up boundary condition
g = Constant(0.0)
bc =DirichletBC(Q, g, DirichletBoundary())

# Define initial condition
indata = Expression("abs(R-sqrt(pow(x[0], 2) + pow(x[1], 2))) <= r ? rho : 0
                                        ", R=0.5, r=0.2, rho=10, degree=2)
u0 = Function(Q)
u0 = interpolate(indata, Q)


# Create bilinear and linear forms
u = TrialFunction(Q)
v = TestFunction(Q)
a = u*v*dx + alpha*dt*inner(grad(u),grad(v))*dx
L = u0*v*dx

# Set an output file
file = File("output_2D/2D_part3_Task1.pvd")

# Set initial condition
u = Function(Q)
# Time-stepping
t = 0
num_steps = int(T / dt)
for n in range(num_steps):
    t += dt

    # compute solution
    solve(a == L, u, bc)
    # save to file
    file << (u, t)

    # assign u0
    u0.assign(u)
```

Problem C.1 3D mesh

```
from dolfin import *

# Create mesh and define function space
mesh = Mesh("sphere1.xml")
Q = FunctionSpace(mesh, "CG", 1)

# Define parameters
T = 20  # End time
h = mesh.hmin()
dt = h
alpha = 0.01

# Create subdomain for Dirichlet boundary
class DirichletBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary

# Set up boundary condition
g = Constant(0.0)
bc =DirichletBC(Q, g, DirichletBoundary())

# Define initial condition
indata = Expression("pow((R-sqrt(pow(x[0], 2) + pow(x[1], 2))),2) + pow(x[2
                                       ], 2) <= pow(r,2) ? rho : 0", R=0.5, r
                                       =0.2, rho=10, degree=2)
u0 = Function(Q)
u0 = interpolate(indata, Q)

# Create bilinear and linear forms
u = TrialFunction(Q)
v = TestFunction(Q)
a = u*v*dx + alpha*dt*inner(grad(u),grad(v))*dx
L = u0*v*dx

# Set an output file
file = File("output_3D/3D_part3_Task1.pvd")

# Set initial condition
u = Function(Q)
# Time-stepping
t = 0
num_steps = int(T / dt)
for n in range(num_steps):
    t += dt

    # compute solution
    solve(a == L, u, bc)
    # save to file
    file << (u, t)

    # assign u0
    u0.assign(u)
```

Problem C.2

```
from dolfin import *

# Create mesh and define function space
mesh = Mesh("sphere1.xml")
Q = FunctionSpace(mesh, "CG", 1)

# Define parameters
T = 50  # End time
h = mesh.hmin()
dt = h
alpha = 0.01

# Create subdomain for Dirichlet boundary
class DirichletBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary

# Set up boundary condition
g = Constant(0.0)
bc =DirichletBC(Q, g, DirichletBoundary())

rho = [10, 20, 40]
for i in range(len(rho)):
    # Define initial condition
    indata = Expression("pow((R-sqrt(pow(x[0], 2) + pow(x[1], 2))),2) + pow(
                                                 x[2], 2) <= pow(r,2) ? rho : 0", R
                                                 =0.5, r=0.2, rho=rho[i], degree=2)
    u0 = Function(Q)
    u0 = interpolate(indata , Q)

    # Create bilinear and linear forms
    u = TrialFunction(Q)
    v = TestFunction(Q)
    a = u*v*dx + alpha*dt*inner(grad(u),grad(v))*dx
    L = u0*v*dx

    # Set an output file
    file = File(f"output_Task2/rho_{rho[i]}.pvd")

    # Set initial condition
    u = Function(Q)

    # Copyinitialdata
    u_initial = Function(Q)
    u_initial = interpolate(indata , Q)

    # Define an integral functional
    M = (u_initial - u) * dx
    # Create a list to store mass loss values
    mass_loss = []

    # Time-stepping
    t = 0
    num_steps = int(T / dt)
    for n in range(num_steps):
        t += dt
        solve(a==L,u,bc)

        # Compute the functional
        mass = assemble(M)

        # Append the mass loss value to the list
        mass_loss.append(mass)
        file << (u, t)

        # assign u0
        u0.assign(u)

    # Save the mass loss values to a file
    with open("mass_loss_{}.txt".format(rho[i]), "w") as f:
        for m in mass_loss:
            f.write(str(m) + "\n")

# Plot the mass loss as a function of time
import matplotlib.pyplot as plt
import numpy as np

# Create a time array
time = np.linspace(0, T, len(mass_loss))

# Plot the mass loss for different rho values
plt.figure()
for i in range(len(rho)):
    # Read the mass loss values from the file
    mass_loss = np.loadtxt("mass_loss_{}.txt".format(rho[i]))
    # Plot the mass loss curve
    plt.plot(time, mass_loss, label="rho = {}".format(rho[i]))

# Add labels and legend
plt.xlabel("Time")
plt.ylabel("Mass loss")
plt.legend()

# Save the figure
plt.savefig("mass_loss.png")
```

Problem C.3

```
import scipy.optimize as optimize
from scipy.optimize import minimize
from dolfin import *

def func(data):
    # solve the heatequationandcomputethe mass loss M(t)
    # Create mesh and define function space
    mesh = Mesh("sphere1.xml")
    Q = FunctionSpace(mesh, "CG", 1)

    # Define parameters
    T = 30  # End time
    h = mesh.hmin()
    dt = 0.1
    alpha = 0.01
    R = data[1]
    r = data[2]
    rho = data[0]

    # Create subdomain for Dirichlet boundary
    class DirichletBoundary(SubDomain):
        def inside(self, x, on_boundary):
            return on_boundary

    # Set up boundary condition
    g = Constant(0.0)
    bc =DirichletBC(Q, g, DirichletBoundary())


    # Define initial condition
    indata = Expression("pow((R-sqrt(pow(x[0], 2) + pow(x[1], 2))),2) + pow(
                                              x[2], 2) <= pow(r,2) ? rho : 0", R
                                              =R, r=r, rho=rho, degree=1)
    u0 = Function(Q)
    u0 = interpolate(indata, Q)

    # Create bilinear and linear forms
    u = TrialFunction(Q)
    v = TestFunction(Q)
    a = u*v*dx + alpha*dt*inner(grad(u),grad(v))*dx
    L = u0*v*dx

    # Set initial condition
    u = Function(Q)

    u_initial = Function(Q)
    u_initial = interpolate(indata , Q)
    # Define an integral functional
    M = (u_initial - u) * dx
    # Create a list to store mass loss values
    Mt = [0,0,0]

    # Time-stepping
    t = 0
    while t < T:
        t = round(t + dt, 2)
        solve(a==L,u,bc)
        # Compute the functional
        mass = assemble(M)
        # Append the mass loss value to the list
        if t == 5:
            Mt[0] = mass
        elif t == 7:
            Mt[1] = mass
        elif t == 30:
            Mt[2] = mass
        # assign u0
        u0.assign(u)
    # get M(5), M(10), M(30)
    # compute theminimizationfunctional f(rho , R, r)
    F = (Mt[0] - 10)**2 + (Mt[1] - 15)**2 + (Mt[2] - 30)**2

    return F

# set initialguess (rho , R, r)
data = [20, 0.5, 0.1]

res = minimize(func , data, method='nelder-mead', options={'xtol':1e-3, '
                                              disp': True})
print(res)
```