

Transient FEM Modelling

Coursework MATLAB-based modelling

Candidate #13442

University of Bath - Department of Mechanical Engineering

December 2023

Keywords: Finite Element Method, Diffusion-Reaction, MATLAB, Drug Delivery, Tissue

CONTENTS

1. Introduction	4
2. Background	5
3. Part 1	6
1. MATLAB implementation	6
2. Software verification	7
4. Part 2	14
1. Solving the drug problem	14
2. Minimum effective dose	16
3. Boundary conditions	18
4. Parameters sensitivity	18
A Appendix: Source Code	20
B Appendix: Unit Tests	39
References	55

LIST OF FIGURES

1	Diagram of human tissue (de Souza Telles, 2010). Note the discrete layers of tissue type. . . .	4
2	Numerical solution at different time stamps ($dt=0.01$, $h=0.1$).	6
3	Analytical solution comparison against space ($dt=0.01$, $h=0.1$).	9
4	Analytical solution comparison against time ($dt=0.01$, $h=0.1$).	10
5	Theta methods comparison at $x=0.8$	11
6	Basis function order comparison at $t=0.05$	12
7	L2-norm error convergence with linear mesh.	13
8	L2-norm error convergence between Crank-Nicolson (dash-dot) and Backward Euler (dash) for a linear mesh.	13
9	L2-norm error convergence between Crank-Nicolson (dash-dot) and Backward Euler (dash) for a quadratic mesh.	14
10	Schematic of finite element mesh required to represent skin tissue layers (Cookson, 2023). . .	15
11	Concentration of a drug applied on the skin.	16
12	Minimum dose of 71 becoming effective after just under 7 seconds.	17
13	Concentration of the minimum effective drug applied to the skin.	17
14	Schematic of finite element mesh required to represent skin tissue layers (Cookson, 2023). . .	18
15	Schematic of finite element mesh required to represent skin tissue layers (Cookson, 2023). . .	19
16	Global mass matrix unit tests results.	41
17	Global stiffness matrix unit tests results.	43
18	Global source vector unit tests results.	44
19	Local mass element unit tests results.	47
20	Diffusion local element matrix unit tests results.	49
21	Reaction local element matrix unit tests results.	52
22	Local source element unit tests results.	54

LISTINGS

1	Transient diffusion-reaction calculator	20
2	One dimensional mesh generator	21
3	Global mass matrix calculator	22
4	Global stiffness matrix calculator	22
5	Global source vector calculator	23
6	Neumann boundary calculator	23
7	Local mass element calculator	24
8	Diffusion local element matrix calculator	24
9	Reaction local element matrix calculator	25
10	Local source element calculator	26
11	Gaussian Quadrature scheme generator	27
12	Basis function evaluator	28
13	Dirichlet boundary calculator	28
14	Main script running the equation solver	29
15	Analytical solution calculator	31
16	L2-norm calculator	32
17	Script running the L2-norm	33
18	Main script running the skin drug diffusion analysis	34
19	Minimum effective dose calculator	37
20	Combination of different diffusion reaction coefficients	37
21	Global mass matrix unit tests	39
22	Global stiffness matrix unit tests	41
23	Global source vector unit tests	43
24	Local mass element unit tests	44
25	Diffusion local element matrix unit tests	47
26	Reaction local element matrix unit tests	49

27	Local source element unit tests	52
----	---	----

1. INTRODUCTION

The finite element method (FEM) is a numerical technique for solving a wide range of complex physical phenomena, particularly those exhibiting geometrical and material non-linearities (Dean, 2023). Diffusion-reaction problems especially benefit from this technique as they are usually applied over a non linear domain.

Section 3. of this report goes through the resolution of the transient diffusion-reaction equation in a 1D mesh using the finite element method. The resolution is accomplished by coding in MATLAB a series of functions to resolve Equation 10, all code written can be found in Appendix A.

After the code is proven to work correctly, in Section 4. the equation is applied to a real word scenario: modelling a drug delivery. Transdermal drug delivery offers a non invasive drug administration together with reduced side effects given by oral assumptions (Varga-Medveczky et al., 2021). Investigation of drug penetration across the skin can be important in topical pharmaceutical formulations.

Considering a block of human skin such as in Figure 1, the problem can be divided in discrete layers represented by a linear mesh. The diffusion reaction equation can be used to model the concentration of the drug in each part of the skin at any point in time.

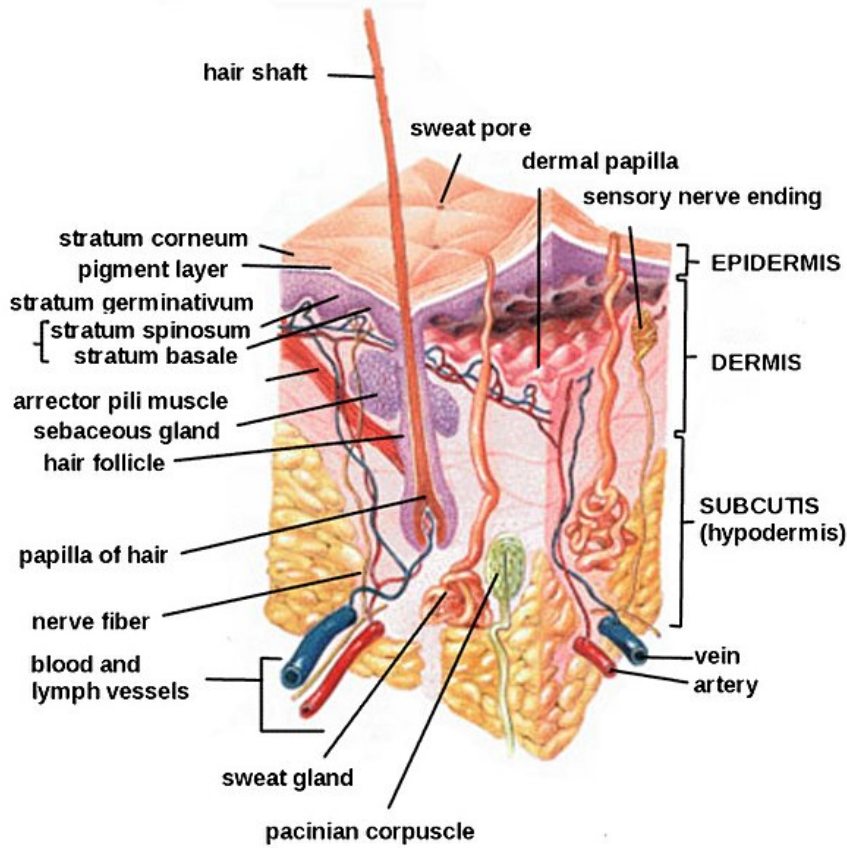


Figure 1. Diagram of human tissue (de Souza Telles, 2010). Note the discrete layers of tissue type.

This problem takes into account the diffusion of the drug and the reaction of the blood vessels and the degradation of the drug. The transient equation for this is therefore given by Equation 1 and is detailed later in the report.

$$\frac{\partial c}{\partial t} = D \frac{\partial^2 c}{\partial x^2} - \beta c - \gamma c \quad (1)$$

2. BACKGROUND

In a FEM analysis the continuous domains of a partial differential equation (PDE) are decomposed into discrete, connected regions. This technique is used to solve the transient form of the diffusion-reaction equation defined as:

$$\frac{\partial c}{\partial t} = D \frac{\partial^2 c}{\partial x^2} + \lambda c + f \quad (2)$$

Equation 2 can be fitted to the general transient equation and then integrated with on both sides with respect to time between two time points:

$$\int_{t_n}^{t_{n+1}} \frac{\partial c}{\partial t} dt = \int_{t_n}^{t_{n+1}} F(x, c, t) dt \quad (3)$$

The left hand side (LHS) is effectively the difference of the concentration at the two given time points: $c(t_{n+1}) - c(t_n)$ while the right hand side (RHS) can be solved graphically using the area of a trapezium. The resultant equation is:

$$c(t_{n+1}) - c(t_n) = \frac{1}{2} \Delta t \cdot (F^{n+1}(x, c, t) + F^n(x, c, t)) \quad (4)$$

The trapezium rule creates what is known as the Crank-Nicolson method. The value of $1/2$ is commonly known as θ , by changing this different methods can be implemented such as Forward Euler ($\theta = 0$) or Backward Euler ($\theta = 1$). Equation 4 can be generalised into the theta scheme to allow for the three resolution methods:

$$\frac{c(t_{n+1}) - c(t_n)}{\Delta t} = \theta F^{n+1}(x, c, t) + (1 - \theta) F^n(x, c, t) \quad (5)$$

There are several different flavours of finite element method, either could be used to solve the Partial Differential Equation (PDE) in Equation 2; in this instance the weighted residual method is applied. The diffusion-reaction equation can be evaluated locally over a discrete domain of finite elements (mesh) using a combination of basis functions. The global transient solution can be solved by assembling the local solutions at each time step giving the following equation:

$$\int_{x_0}^{x_1} \left(v \frac{\partial c}{\partial t} + D \frac{\partial v}{\partial x} \frac{\partial c}{\partial x} - \lambda cv \right) dx = \int_{x_0}^{x_1} v f dx + \left[v D \frac{\partial c}{\partial x} \right]_{x_0}^{x_1} \quad (6)$$

The local element of the time derivative in Equation 6 is given by the integral over the domain $x = [-1, 1]$ while $c = c_n \psi_n$ and $v = \psi_m$. the temporal derivative can be taken outside the integral and transformed to a simple delta fraction resulting in:

$$\frac{dc_n}{dt} \int_{-1}^1 \psi_n \psi_m J d\xi = \frac{c(t_{n+1}) - c(t_n)}{\Delta t} \int_{-1}^1 \psi_n \psi_m J d\xi \quad (7)$$

The integral part of the local element equation is known as the local mass element (M_{elem}) shown in Equation 8. The local stiffness element (K_{elem}) includes the diffusion and reaction terms, this is given by Equation 9.

$$M_{elem} = \int_{-1}^1 \psi_n \psi_m J d\xi \quad (8)$$

$$K_{elem} = \int_{-1}^1 D \frac{d\psi_n}{d\xi} \frac{d\xi}{dx} \frac{d\psi_m}{d\xi} \frac{d\xi}{dx} J d\xi - \int_{-1}^1 \lambda \psi_n \psi_m J d\xi \quad (9)$$

The single mass and stiffness elements can be assembled into their respective global matrices by combining them over all elements. Implementing these global matrices and using the general theta scheme, the transient FEM equation can be written as:

$$[M + \theta \Delta t K] c^{n+1} = [M - (1 - \theta) \Delta t K] c^n + \Delta t \theta [F^{n+1} + N B c^{n+1}] + \Delta t (1 - \theta) [F^n + N B c^n] \quad (10)$$

where $N B c$ is the Neumann Boundary conditions and $[M + \theta \Delta t K]$ is called the final global matrix.

3. PART 1

In this first part of the exercise code is written in MATLAB to solve the transient form of the diffusion-reaction equation. The written software is then verified using a range of techniques to confirm its working.

1. MATLAB implementation

MATLAB is used to solve the equation as it is a powerful tool for numeric computing, matrix manipulation and functions plotting. The code is structured as a series of functions in order to make the single scripts concise and generic.

The main function *TransientFEM.m*, available in Listing 1, solves the transient diffusion-reaction equation 10. The function does not set any variable locally but relies on the inputs given by the user. In this order the function initialises a mesh (Listing 2) and gets the global mass (M) and stiffness (K) by calling the functions *GlobaMassMatrix.m* and *GlobalStiffnessMatrix.m* shown in Listings 3 and 4 respectively. These together with theta (θ) allow to solve the final global matrix at the current and previous time point.

The global source vector and the Neumann boundary conditions are computed using another pair of functions: *GlobalSourceVector.m* and *NeumannBoundary.m* (Listings 5 and 6).

Finally the equation is assembled and the value of $c(x)$ at the next time step is calculated; looping through this step along the entire time domain would give the transient form $c(x, t)$.

The global functions mentioned above are complemented by their respective local element functions seen in Listings 7, 8, 9 and 10. The transient diffusion-reaction solver was ran with some initial conditions stated in the Software verification section producing the graph in Figure 2.

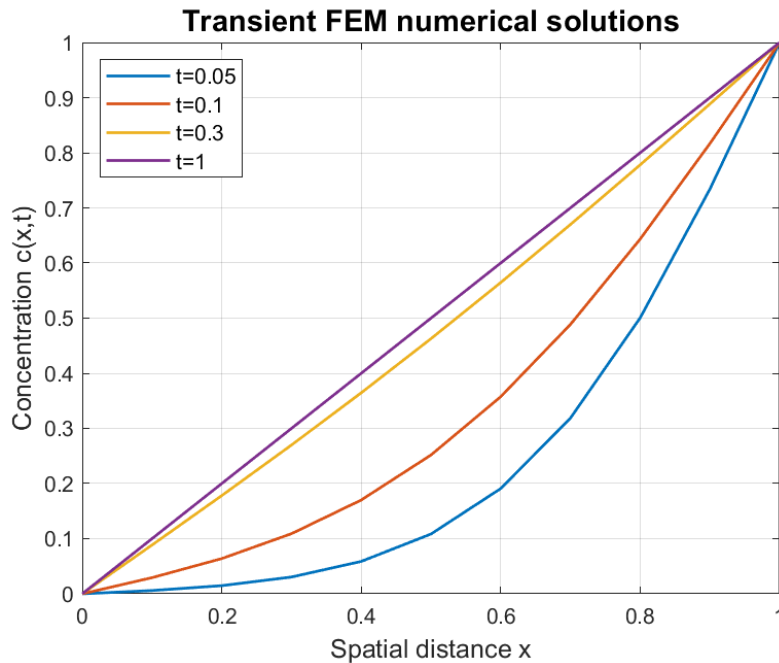


Figure 2. Numerical solution at different time stamps ($dt=0.01$, $h=0.1$).

Some additional features are implemented into the scripts to make the overall computation more generalised and to test different parameters.

One of this is the use of the Gaussian Quadrature method to evaluate the integrals throughout the equations. This removes the need for manual integration prior to implementation in code while giving a numerical integration as accurate as the analytical solution. The Gaussian-Legendre Quadrature turns any integral of polynomial into a N-point sum of Gauss weights times the function at the Gauss points according to Equation 11 (Abramowitz and Stegun, 1964).

$$\int_{-1}^1 f(x)dx \approx \sum_{i=1}^n w_i f(x_i) \quad (11)$$

The weights and points are bound to the order of the polynomial ($order = 2N - 1$), these values are given in Table 1 and are stored in software through the function *GQscheme.m* in Listing 11.

Another additional feature is the implementation of quadratic basis functions as opposed to linear ones. The software function *EvalBasis.m* found in Listing 12 evaluates a linear or quadratic basis function depending on the given input.

The last function written is *DirichletBoundary.m* in Listing 13 which enforces the Dirichlet Boundary Conditions to both the right hand side and the global matrix in Equation 10.

Table 1. First five quadrature rules over the interval $[-1, 1]$ (Abramowitz and Stegun, 1964)

N	Points (x_i)	Weights (w_i)
1	0	2
2	$\pm\sqrt{\frac{1}{3}}$	1
3	$0, \pm\sqrt{\frac{1}{3}}$	$\frac{8}{9}, \frac{5}{9}$
4	$\pm\sqrt{\frac{3}{7} \pm \frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{18 \pm \sqrt{30}}{36}$
5	$0, \pm\frac{1}{3}\sqrt{5 \pm 2\sqrt{\frac{10}{7}}}$	$\frac{128}{225}, \frac{322 \pm 13\sqrt{70}}{900}$

2. Software verification

Before applying the written software to a real word scenario it should be thoroughly tested. The verification can be done at a local level by testing the individual functions or at a global one comparing the final solution with an analytical equivalent.

At the local level the MATLAB unit testing framework is used to assert the outputs of each basic function. This allows to write test scripts which run the selected function on set parameters and assert if the results are within tolerance of the expected results. The unit tests check for either the physical shape of the matrix or for the values contained in it, all tests used are shown in Table 2.

All Unit tests scripts together with their results are found in Appendix B.

Table 2. List of tests used to assert which function.

Type	Description	Functions
Symmetry	Test that the matrix/vector is symmetric	Global Mass Matrix, Global Source Vector, Global Stiffness Matrix, Local Element Diffusion, Local Element Reaction, Local Mass Element, Local Source Element
Evaluation	Test the correct evaluation comparing with the analytical results	Global Mass Matrix, Global Source Vector, Global Stiffness Matrix, Local Element Diffusion, Local Element Reaction, Local Mass Element, Local Source Element
Size	Test that the matrix/vector is of the expected size	Global Mass Matrix, Global Source Vector, Global Stiffness Matrix
Diagonals	Test that the matrix values are only in the diagonals	Global Mass Matrix, Global Stiffness Matrix
Elements evaluation	Test that two elements in the mesh produce the same matrices	Local Element Diffusion, Local Element Reaction, Local Mass Element, Local Source Element
Gaussian Quadrature	Test that the Gaussian Quadrature solution are the same as the manual integration ones	Local Element Diffusion, Local Element Reaction, Local Mass Element, Local Source Element

The global level verification consists of comparing the numerical output solution with an analytical one. For ease of calculation of the analytical equation the original transient diffusion-reaction equation (Equation 2) is simplified to include only the diffusion term as:

$$\frac{\partial c}{\partial t} = \frac{\partial^2 c}{\partial x^2} \quad (12)$$

This was solved subject to the following initial and Dirichlet boundary conditions:

$$x = [0, 1], \quad c(x, 0) = 0, \quad c(0, t) = 0, \quad c(1, t) = 1, \quad t > 0$$

The transient diffusion equation is computed analytically using Equation 13. This solution can be compared with the numerical solution with the same boundary conditions as stated above using a range of element sizes and time steps to demonstrate spatial and temporal convergence. The MATLAB script *TransientAnalyticSoln.m* used to solve the analytical equation is shown in Listing 15.

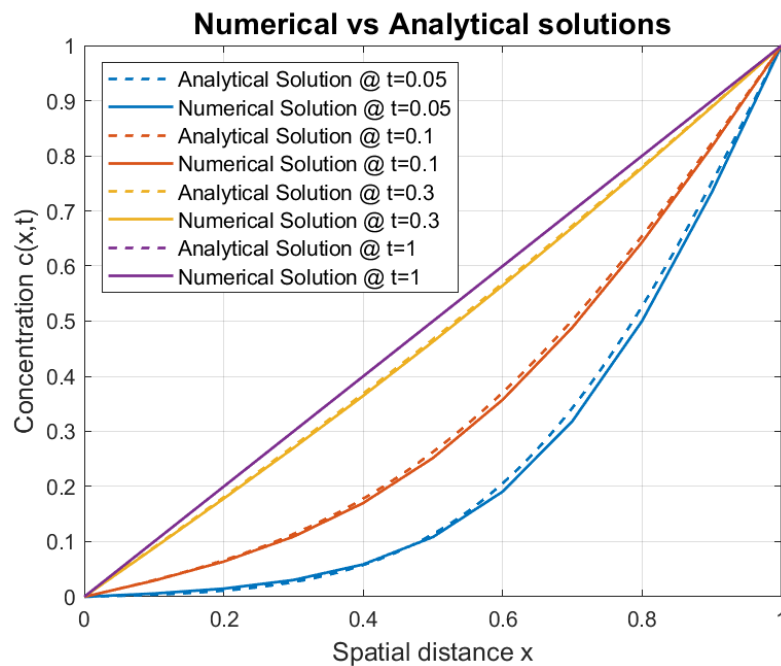
$$c(x, t) = x + \frac{2}{\pi} \sum_{n=1}^{\infty} \frac{(-1)^n}{n} e^{-n^2 \pi^2 t} \sin(n \pi x) \quad (13)$$

Figure 2 initially plotted only the numerical solution of the simplified transient diffusion equation in a ten element mesh ($N_e = 10$) and a time step $dt = 0.01$, the step was repeated showing the solutions at $t = 0.05, 0.1, 0.3, 1.0$. The full list of used parameters used to solve the transient diffusion-reaction equation are listed in Table 3.

In Figure 3, Equation 13 is plotted as dashed lines at the same time points giving a one to one comparison to the numerical solution.

Table 3. Initial parameters for the transient diffusion-reaction solution.

Parameter	Value	Unit
Diffusion coefficient	$D = 1$	
Reaction coefficient	$\lambda = 0$	
Source term	$f = 0$	
Mesh boundaries	$0 < x < 1$	meters
Mesh elements	$Ne = 10$	
Time domain	$0 < t < 1$	seconds
Time step	$dt = 0.01$	seconds
Gauss Points	$npts = 2$	

**Figure 3.** Analytical solution comparison against space ($dt=0.01$, $h=0.1$).

From this figure it can be noted that the numerical solution at $t = 0.05$ has the largest error and also discontinuities in the gradient. The gradient will be discussed later in a direct comparison with the basis function order. The error can be seen decreasing with the increase of t until it becomes close to zero when $t = 1$.

At this point the time divergence can be further tested fixing the spatial distance to a point ($x = 0.8$) and analysing the diffusion curve in the time interval from $t = 0$ to 1.0 (Figure 4).

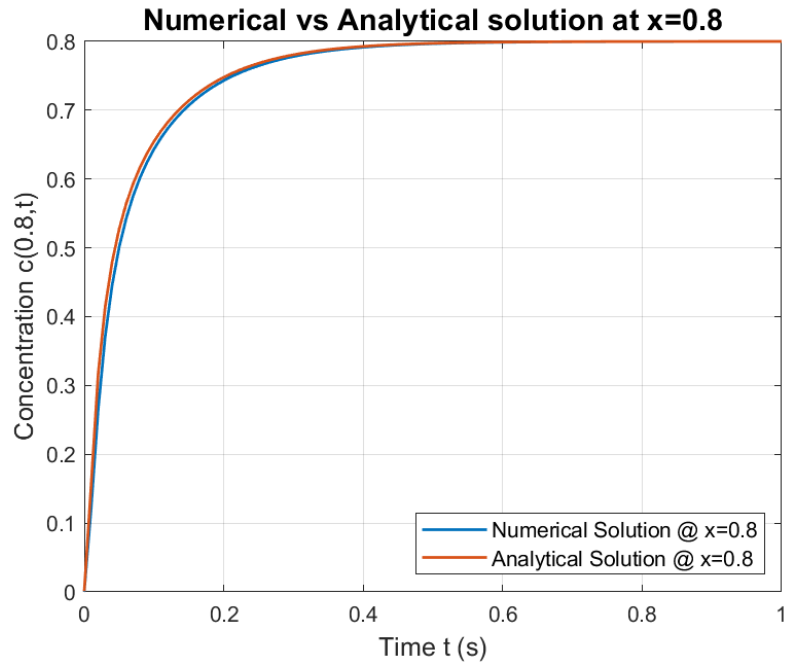


Figure 4. Analytical solution comparison against time ($dt=0.01$, $h=0.1$).

This figure confirms the findings in Figure 3 where small values of time have the largest error between the numerical and analytical solutions. For time above 0.4s the numerical solution converges with the analytical and the error is gradually reduced to zero.

The way the code was written no hard coded values were used but only inputs given by the end user. This allows to test different combination of parameters such as the value of theta (θ) to switch between forward Euler, backward Euler and Crank-Nicolson methods. These schemes are employed with the finite element method, where the spatial and temporal derivatives are replaced with discrete difference approximations.

Using the same conditions of $Ne = 10$ and $dt = 0.01$ the transient diffusion equation was solved at $x = 0.8$ for each of the theta methods in Figure 5.

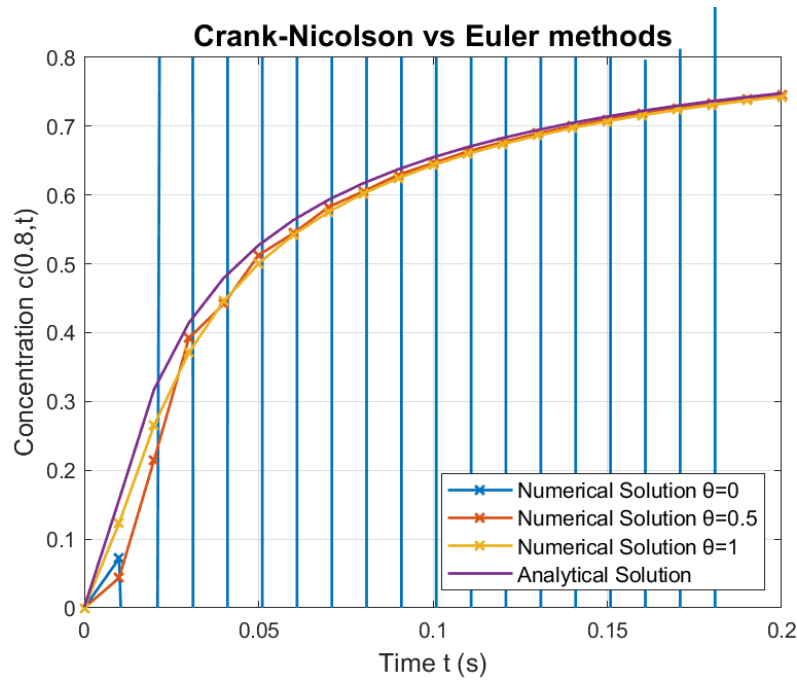


Figure 5. Theta methods comparison at $x=0.8$.

The **Forward Euler** scheme is conditionally stable with a strong dependence with the selected time step. This method has a first order accuracy in time and is generally least computational expensive however stability is only reached at extremely small time steps. The chosen time step gave great instability to the solution which is why this method was excluded.

The **Backward Euler** scheme is unconditionally stable meaning there are no conditions to its stability. This method also has a first order accuracy in time but because it requires the solution of a system of linear equations at each time step it is more computationally expensive compared to the forward Euler scheme.

The **Crank-Nicolson** scheme is also unconditionally stable but introduces a second order accuracy in time making it very accurate but computationally expensive at the same time.

Moving forward the Backward Euler scheme was chosen for the real word application task. Despite this method exhibiting lower accuracy compared to Crank-Nicolson it is less computationally expensive. In addition the time step and the size of the mesh need to be perfectly balanced to prevent oscillation in the Crank-Nicolson solution which can be observed in the figure.

Next the order of the basis function was investigated. A quadratic basis function can be obtained by introducing additional nodes within each linear element. The global matrix has more elements in case of a quadratic basis function, specifically the matrix becomes a 3-by-3 with overlapping nodes at the shared nodes between adjacent elements. Quadratic elements require higher-order numerical integration, this is where the Gaussian-Legendre quadrature comes in to numerically compute element stiffness matrices and load vectors accurately. Higher-order functions allow for a more accurate representation of the solution, especially for problems with curvature or sharp gradients. To prove the superiority of quadratic to linear basis functions both were plotted at $t = 0.05$ against the analytical solution in Figure 6

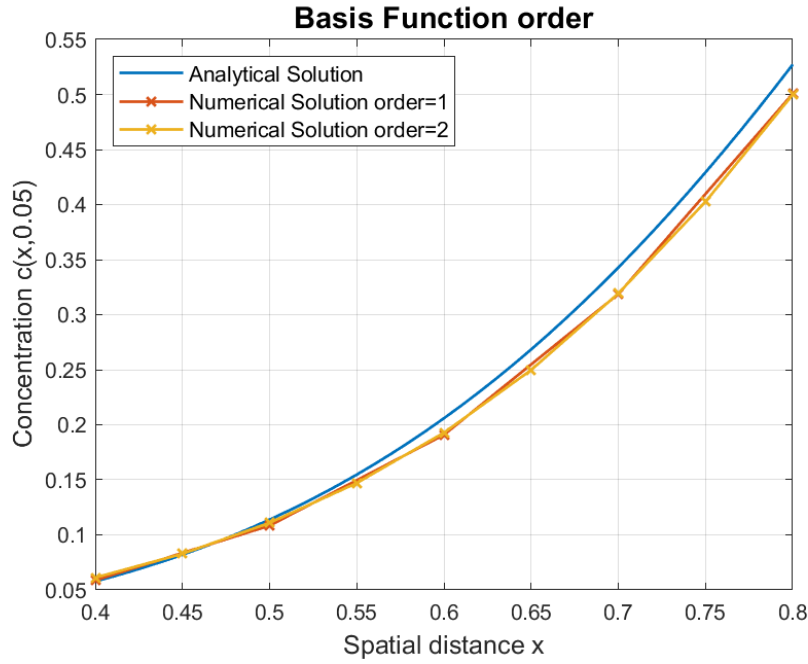


Figure 6. Basis function order comparison at $t=0.05$.

This figure is a cropped version of Figure 3 with a focus on $t = 0.05$ with the two basis functions. This was chosen because the solution here has a sharp curve with continuous gradient change. It is noticeable how a linear function introduces staggered lines due to the linear approximation between the mesh nodes. The quadratic function not only introduces extra nodes but also smoothens the curve to better approximate the analytical solution.

The better way of comparing the error obtained in the previous investigations would be to use the L_2 -norm. This is particularly useful for comparing a numerical solution to an analytical solution testing the convergence rate of the Finite Element Method. The L_2 -norm is also known as the Euclidean norm because it computes the Euclidean distance between two given point or effectively the Root Mean Square (RMS) of the error. The Gaussian-Legendre quadrature comes once again to help resolve the integral of the polynomial finally obtaining Equation 14.

$$\|E\|_{L_2} = \left[\int_{\Omega} E^2(x) dx \right]^{1/2} = \int_{-1}^1 (C_E(x) - C(x))^2 J d\xi = \sum_{i=1}^N w_i (C_E(x(\xi_i)) - C(\xi_i))^2 J \quad (14)$$

where $x(\xi_i)$ is represented by x_0, x_1 local to the element $x(\xi_i) = x_0\psi_0(\xi_i) + x_1\psi_1(\xi_i)$ and C is represented by c_0, c_1 local to the element $C(\xi_i) = c_0(1 - \xi_i/2) + c_1(1 + \xi_i/2)$.

Given the linear mesh used to represent the solution, the analytical solution (C_E) can be expressed against the numerical one (C) with the element size (h). The natural log is taken on each side resulting in a straight line of gradient 2 (Figure 7).

$$\ln(E(x)) = \ln G + 2\ln h \quad (15)$$

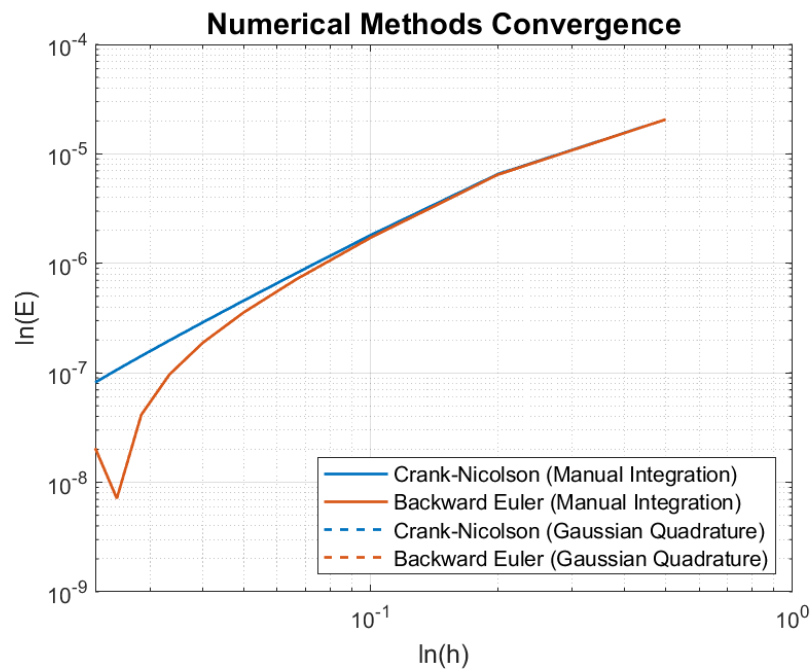


Figure 7. L2-norm error convergence with linear mesh.

In this figure the Gaussian quadrature method perfectly overlaps the manual integration because the basis function, therefore the mesh, is linear.

The L2-norm is then applied to the previous analysis such as the theta scheme and the basis function order for multiple time points. Using a Linear basis function with both the Crank-Nicolson and the Backward Euler methods show little discrepancies in the error especially with smaller space steps (dx). Figure 8 shows a divergence starting for $\ln(h) < 0.05$.

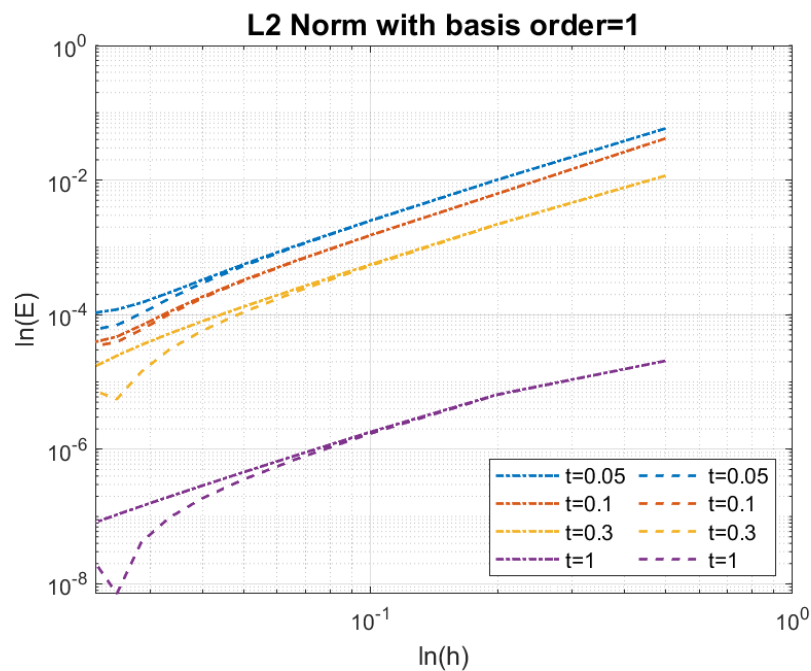


Figure 8. L2-norm error convergence between Crank-Nicolson (dash-dot) and Backward Euler (dash) for a linear mesh.

The same comparison was drawn in Figure 9 with quadratic basis functions. In this figure the differences between the two methods become much more evident and Crank-Nicolson comes out as the most accurate. This is likely due to the higher accuracy of a quadratic basis function, because Crank-Nicolson is a second-order accurate time integration scheme, it models better rapidly changing functions leading to accurate results. Quadratic basis functions are also sensible to the mesh density, this is the reason why the minimum mesh size for both elements to converge is higher.

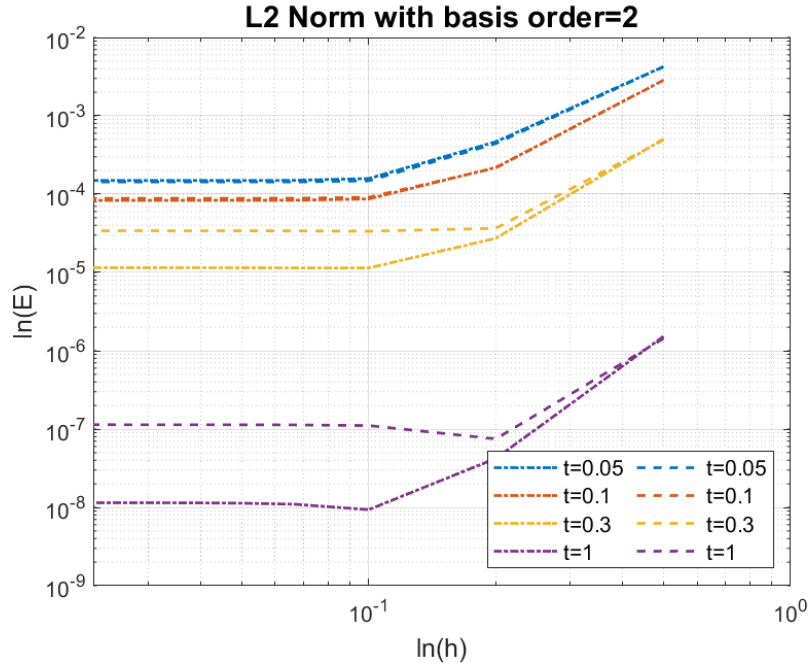


Figure 9. L2-norm error convergence between Crank-Nicolson (dash-dot) and Backward Euler (dash) for a quadratic mesh.

4. PART 2

As the code written in Part 1 is proven to work as expected it was then applied to the drug delivery problem. The drug concentration distribution in the skin tissue is modelled by Equation 1, which is a simplified version of the transient diffusion-reaction Equation 2 without the source term.

Within tissue D can be defined as the diffusion coefficient of the drug. Flow in blood vessels, instead, removes the drug from a given location therefore this sink can be modelled as a linear reaction with coefficient β . Finally, as the drug diffuses into the tissue it starts chemical reactions which reduce the overall drug potency and effectiveness on the target, this degradation is modelled as a second linear reaction term with coefficient γ . Equation 1 can be simplified to group the reaction terms as:

$$\frac{\partial c}{\partial t} = D \frac{\partial^2 c}{\partial x^2} - (\beta + \gamma)c \quad (16)$$

1. Solving the drug problem

Compared to the computation in Part 1 the skin cannot use a single uniform mesh because, as seen from Figure 1, it is made of different layers each carrying its own properties. The finite element mesh can be divided into three parts to represent the epidermis, dermis and sub-cutaneous layers like in Figure 10.

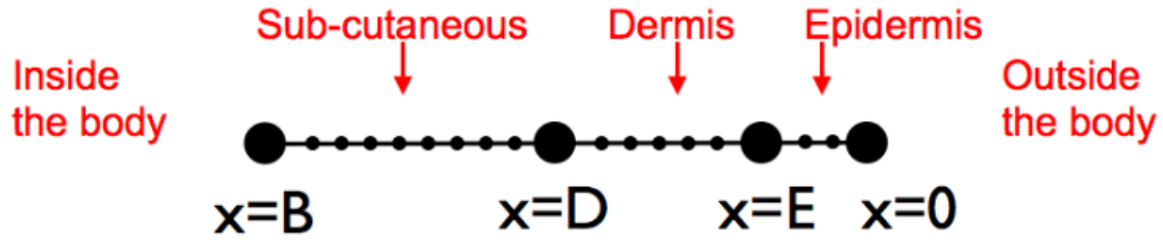


Figure 10. Schematic of finite element mesh required to represent skin tissue layers (Cookson, 2023).

These layers are defined at $x_e = 0.00166667$, $x_d = 0.005$ and $x_b = 0.01$ and have parameters listed in Table 4.

Table 4. Parameters given to solve the drug delivery problem (Cookson, 2023).

Parameter	Effect	Epidermis	Dermis	Sub-cutaneous
D	Diffusion coefficient	4×10^{-6}	5×10^{-6}	2×10^{-6}
β	Extra-vascular diffusivity	0	0.01	0.01
γ	Drug degradation rate	0.02	0.02	0.02

This variable mesh was obtained by implementing a check in the *OneDimLinearMeshGen.m* code (Listing 2) checking the position along the mesh of the current element and assigning the relevant parameters.

The final transient diffusion-reaction equation was solved subject to the following initial and Dirichlet boundary conditions and to the parameters in Table 5:

$$c(x, 0) = 0, \quad c(x = B, t) = 0, \quad c_{dose}(0, t) = 30$$

Table 5. Remaining parameters for the drug delivery problem resolution.

Parameter	Value	Unit
Mesh boundaries	$0 < x < 0.01$	meters
Mesh elements	$Ne = 40$	
Time domain	$0 < t < 30$	seconds
Time step	$dt = 0.01$	seconds
Gauss Points	$npts = 4$	

Using the script *Part2Plotter.m* in Listing 18 these parameters are initialised and the results are plotted as shown in Figure 11.

In this figure it is possible to understand how a certain drug dose applied to the surface of the skin ($x = 0$) slowly penetrates the layers while at the same time decaying. The transient solution is plotted at specific time points to show the convergence of the solutions. After approximately 10 seconds the solution reaches a steady state meaning that the drug concentration does not change with time.

Vertical lines were added to the plot to show the position of the layer boundaries explaining the sudden change in gradient of the concentration curves.

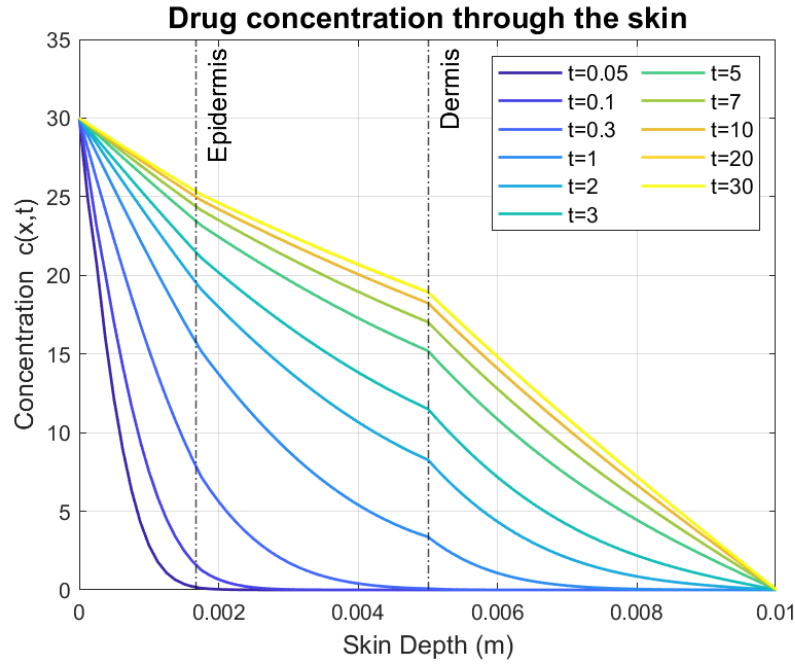


Figure 11. Concentration of a drug applied on the skin.

2. Minimum effective dose

Having resolved how a drug propagates in the skin one could find the minimum dose needed to have the desired pharmaceutical effect. At the Dermis-Sub-cutaneous boundary the drug is said to be effective when, the solution of Equation 17, K is greater than 1000.

$$K = \int_{t_{eff}}^{t=30} c dt \quad (17)$$

where t_{eff} is the point in time when the concentration is above the effectiveness threshold $c > 40$.

To compute this integral the function *MinEffectiveDose.m* in Listing 19 is written. The result of this investigation gives that a minimum dose of 71 should be applied on the outer skin to have any effect at the given depth point. In addition the drug concentration in time for this dose can be plotted (Figure 12) returning the time value necessary to reach the effectiveness.

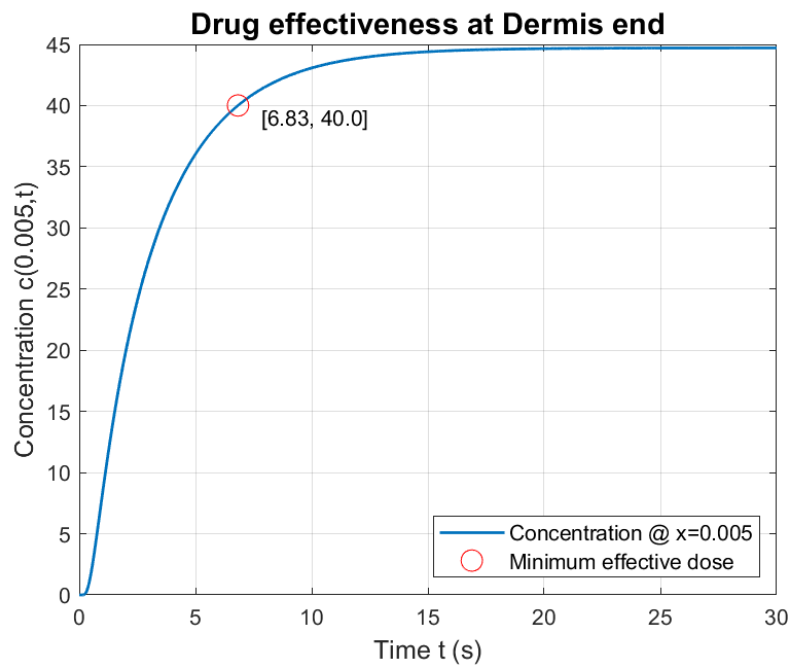


Figure 12. Minimum dose of 71 becoming effective after just under 7 seconds.

Using this minimum effective dose as the new Dirichlet boundary condition of the solver gives the effective drug transient diffusion-reaction plot in Figure 13. This plot shows in another way how the minimum dose on the skin gives a concentration value above the effectiveness concentration at $x = 0.005$.

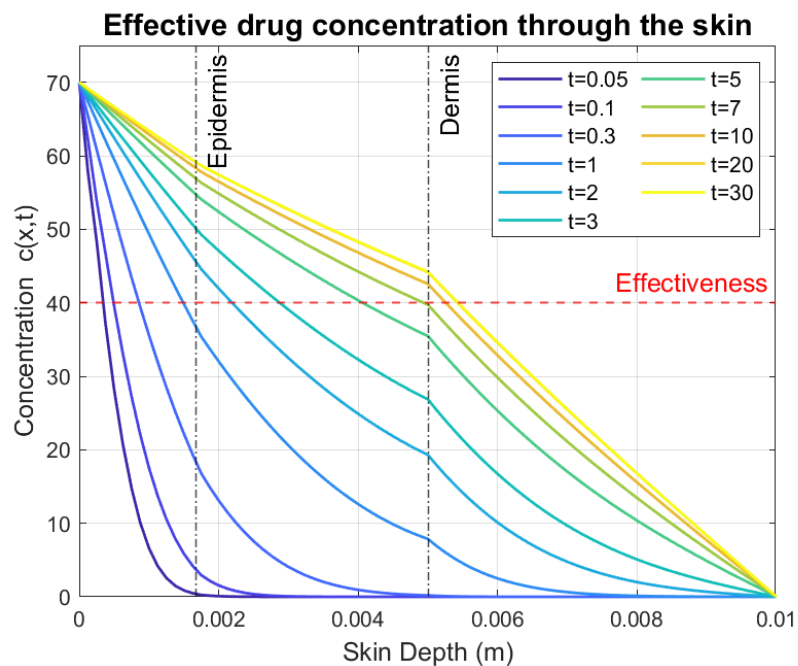


Figure 13. Concentration of the minimum effective drug applied to the skin.

3. Boundary conditions

The choice of boundary conditions plays a crucial role in determining the behaviour of the drug delivery in the skin. For this investigation the drug was applied to the outer skin therefore the most important condition was the Dirichlet lower boundary at $x = 0$ ($c(x = 0, t)$). If the drug was to be delivered to the skin via the blood flow the Dirichlet upper boundary would carry the initial drug concentration.

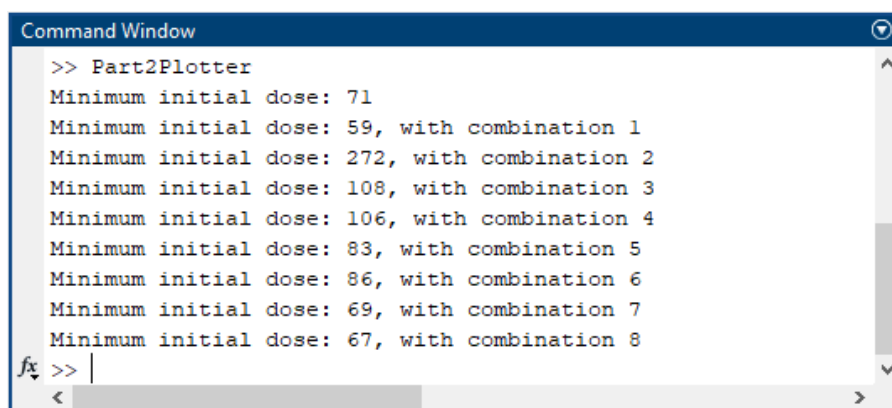
Additionally fixing the drug concentration boundary to a fixed value might not be realistic. This would mean that the drug is applied at a constant rate which is usually hard to achieve.

The upper Dirichlet boundary condition of $c(x = B, t) = 0$ is even less realistic. This condition could only be suitable if there are physical or biological barriers preventing the drug from entering the body. But realistically even the body is permeable to the drug meaning that the diffusion should continue beyond the tissue layers.

4. Parameters sensitivity

The effective delivered dose is dependent to the diffusion and reaction coefficients in different ways. To explore the effectiveness of each coefficient the function *CombinationsDR.m* was written. This contains a range of 8 combinations of diffusion and reaction coefficients, shown in Table 6.

For each combination the minimum effective dose at the skin was computed (Figure 14), and the concentration plotted in time resulting in Figure 15.



```

>> Part2Plotter
Minimum initial dose: 71
Minimum initial dose: 59, with combination 1
Minimum initial dose: 272, with combination 2
Minimum initial dose: 108, with combination 3
Minimum initial dose: 106, with combination 4
Minimum initial dose: 83, with combination 5
Minimum initial dose: 86, with combination 6
Minimum initial dose: 69, with combination 7
Minimum initial dose: 67, with combination 8
fx >>

```

Figure 14. Schematic of finite element mesh required to represent skin tissue layers (Cookson, 2023).

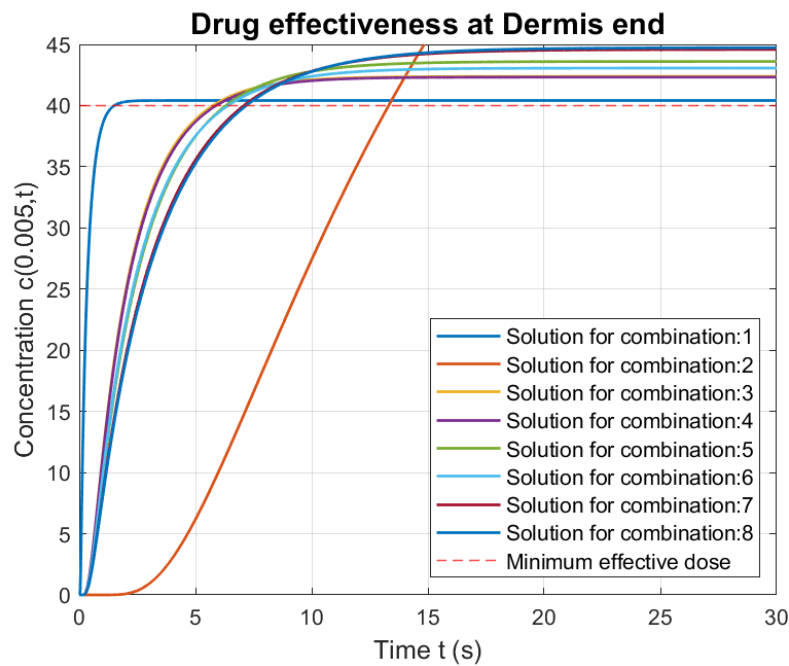


Figure 15. Schematic of finite element mesh required to represent skin tissue layers (Cookson, 2023).

During the different combinations the parameters were always changed by a factor of ten. Nevertheless, the figures above clearly show how the diffusion coefficient have a much higher impact over the analysis. A ten time increase in the diffusion coefficients results in a minimum effective dose of only 59 reaching the target point in 1.6 seconds. On the other hand a ten time decrease gives a minimum effective dose of 272 reaching the target in 13.4 seconds.

Both γ and β have unsurprisingly the same effect on the solution as they are both summed together into the single reaction coefficient λ . The reaction coefficients had a minimal impact overall only varying the initial dose from 69 to 108 with the target reached between 5.8 and 7.2 seconds.

Table 6. Combinations for parameter testing. ('=':parameter unchanged).

Combination	D	γ	β
1	$10\times$	=	=
2	$0.1\times$	=	=
3	=	$10\times$	=
4	=	$10\times$	$0.1\times$
5	=	$0.1\times$	$10\times$
6	=	=	$10\times$
7	=	=	$0.1\times$
8	=	$0.1\times$	=

The findings indicate that the efficacy of drug delivery is significantly influenced by the diffusion through the tissue layers, with other factors playing a comparatively lesser role. The diffusion coefficient exhibits a pronounced dependency on tissue thickness and characteristics. For instance, in the context of human skin, regions with thicker and tougher skin, such as the palms and soles, exhibit lower diffusion rates compared to areas with thinner and softer skin, like the back or genitalia (Varga-Medveczky et al., 2021). This underscores the importance of considering tissue-specific parameters in the design and optimisation of drug delivery systems.

A APPENDIX: SOURCE CODE

Listing 1. Transient diffusion-reaction calculator

```

1 function [c,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters)
2 %Solves the full transient form of the diffusion reaction equation.
3 %  $[M + \theta \Delta t K]c_{n+1} = [M (1 - \theta) \Delta t K]c_n + \Delta t \theta [F_{n+1} + N_{Bc_{n+1}}] + \Delta t (1 - \theta) [F_n + N_{Bc_n}]$ 
4 %
5 % Input:
6 % xmin : Lower spatial boundary
7 % xmax : Upper spatial boundary
8 % Ne : Number of elements
9 % order : weather the basis functions is linear or quadratic
10 % theta : Method selection
11 % time : All time related values combined
12 % GQ : Gaussian Quadrature parameters
13 % boundary : Dirichlet and Neumann conditions combined
14 % parameters : Parameters for current material
15 % Return:
16 % c : Solution to the full transient FEM
17 % mesh : Finite element mesh
18 % GQ : Gaussian Quadrature parameters
19 % time : All time related values combined
20 %
21 %Francesco Berteau (fb552) - November 2023
22
23 %finite element mesh between Xmin and Xmax with Ne number of elements
24 mesh = OneDimLinearMeshGen(Xmin,Xmax,Ne,order,parameters);
25
26 %get Gaussian points and weights
27 [GQ] = GQscheme(GQ);
28
29 %delta t
30 dt = time.dt;
31
32 %get global mass matrix
33 [GMmass] = GlobalMassMatrix(Ne,mesh,GQ,order);
34
35 %get global stiffness matrix
36 [GMstiffness] = GlobalStiffnessMatrix(Ne,mesh,GQ,order);
37
38 %current global matrix  $[M + \theta \Delta t K]$ 
39 GM = GMmass + theta*dt*GMstiffness;
40 %previous global matrix  $[M - (1-\theta) \Delta t K]$ 
41 prevGM = GMmass - ((1-theta)*dt*GMstiffness);
42
43 %initialise list of vector c with zeros
44 c = zeros(mesh.ngn,length(time.t));
45 %initial condition  $c(x,0)=$ initial condition
46 c(:,1) = time.ic;
47
48 %get global source vector
49 [GVsource] = GlobalSourceVector(Ne,mesh,GQ,order);
50
51 %Neumann boundary conditions, current and following
52 [NB,NBnext] = NeumannBoundary(boundary,mesh);
53
54 %compute RHS hence c at each time point
55 for n = 1:time.N
56
57     %RHS vector of equation
58     RHS = (prevGM*c(:,n)) + dt*theta*(GVsource+NBnext) + dt*(1-theta)*(GVsource + NB);
59
60     %Dirichlet boundary conditionsfor RHS and GM
61     [RHS,GM] = DirichletBoundary(boundary,RHS,GM);
62
63     %next numerical according to equation
64     c(:,n+1) = GM\RHS;
65

```

```

66     end
67 end

```

Listing 2. One dimensional mesh generator

```

1 function [mesh] = OneDimLinearMeshGen(xmin,xmax,Ne,order,parameters)
2 %%This function generates a one dimensional, equispaced, linear finite
3 %%element mesh, with Ne number of elements, between the points at x
4 %%position xmin and xmax.
5
6     mesh.ne = Ne; %set number of elements
7     mesh.ngn = Ne+1 + (Ne*(order -1)); %set number of global nodes
8     mesh.nvec = zeros(mesh.ngn,1); %allocate vector to store global node values
9     dx = (xmax - xmin)/Ne; %calculate element size
10
11     mesh.nvec = xmin:dx-(0.5*dx*(order-1)):xmax;
12
13     %loop over elements & set the element properties
14     for i=1:Ne
15         %set element Jacobian based on mapping to standard element
16         mesh.elem(i).J = 0.5*dx; %this is assuming standard element of -1 to 1
17
18         switch order
19             case 1 %linear
20                 %set spatial positions of nodes
21                 mesh.elem(i).x(1) = xmin + (i-1)*dx;
22                 mesh.elem(i).x(2) = xmin + i*dx ;
23                 %set global IDs of the nodes
24                 mesh.elem(i).n(1) = i;
25                 mesh.elem(i).n(2) = i+1;
26
27             case 2 %quadratic
28                 %set spatial positions of nodes
29                 mesh.elem(i).x(1) = xmin + (i-1)*dx;
30                 mesh.elem(i).x(2) = xmin + i*dx - dx/2;
31                 mesh.elem(i).x(3) = xmin + i*dx ;
32                 %set global IDs of the nodes
33                 mesh.elem(i).n(1) = (i*2)-1;
34                 mesh.elem(i).n(2) = (i*2);
35                 mesh.elem(i).n(3) = (i*2)+1;
36         end
37
38         %check weather using material 1 or 2
39         if parameters.selection == '1'
40
41             %diffusion coefficient of current element
42             mesh.elem(i).D = parameters.D;
43             %reaction coefficient of current element
44             mesh.elem(i).lambda = parameters.lambda;
45             %source term of current element
46             mesh.elem(i).f = parameters.f;
47
48         elseif parameters.selection == '2'
49             %position on the mesh of current element
50             position = mesh.elem(i).x(1);
51
52             % Set parameters values based on current position
53             %epidermis layer
54             if(position >= 0 && position < parameters.Xe)
55                 %diffusion coefficient and no blood flow
56                 mesh.elem(i).D = parameters.De;
57                 mesh.elem(i).beta = parameters.betaE;
58                 mesh.elem(i).gamma = parameters.gammaE;
59
60             %dermis layer
61             elseif (position >= parameters.Xe && position < parameters.Xd)
62                 %diffusion coefficient
63                 mesh.elem(i).D = parameters.Dd;
64                 mesh.elem(i).beta = parameters.betaD;
65                 mesh.elem(i).gamma = parameters.gammaD;

```

```

66         %sub-cutaneous layer
67     elseif (position >= parameters.Xd && position <= parameters.Xb)
68         %diffusion coefficient
69         mesh.elem(i).D = parameters.Db;
70         mesh.elem(i).beta = parameters.betaB;
71         mesh.elem(i).gamma = parameters.gammaB;
72     end
73
74     %sum of the reaction terms
75     mesh.elem(i).lambda = -(mesh.elem(i).beta + mesh.elem(i).gamma);
76     %no source term
77     mesh.elem(i).f = 0;
78 end
79 end
80 end
81 end

```

Listing 3. Global mass matrix calculator

```

1 function [GMmass] = GlobalMassMatrix(Ne,mesh,GQ,order)
2 %Assembles the single local mass elements into a global matrix
3 % The local mass elements are computed at each element in the finite
4 % element mesh. Based on their location they are then inserted in the
5 % global mass matrix of size mesh.ngn-by-mesh.ngn.
6 %
7 % Input:
8 % Ne : Number of elements
9 % mesh : Finite element mesh
10 % GQ : Gaussian Quadrature parameters
11 % order : weather the basis functions is linear or quadratic
12 % Return:
13 % GMmass : Global Mass matrix
14 %
15 %Francesco Berteau (fb552) - November 2023
16
17 %initialize global mass matrix with zeros
18 GMmass = zeros(mesh.ngn);
19
20 %calculate and assembly the global mass matrix
21 for eN = 1:Ne
22
23     %local matrix for mass element
24     [LMmass] = LocalMassElem(eN,mesh,GQ,order);
25
26     %location in GMatrix
27     i = order*eN - (order-1);
28
29     GMmass(i:i+order,i:i+order) = GMmass(i:i+order,i:i+order) + LMmass;
30 end
31 end

```

Listing 4. Global stiffness matrix calculator

```

1 function [GMstiffness] = GlobalStiffnessMatrix(Ne,mesh,GQ,order)
2 %Assembles the single local diffusion and reaction elements into a global matrix
3 % The local diffusion and reaction elements are computed at each element
4 % in the finite element mesh. Based on their location they are then
5 % inserted in the global stiffness matrix of size mesh.ngn-by-mesh.ngn.
6 %
7 % Input:
8 % Ne : Number of elements
9 % mesh : Finite element mesh
10 % GQ : Gaussian Quadrature parameters
11 % order : weather the basis functions is linear or quadratic
12 % Return:
13 % GMstiffness : Global Stiffness matrix
14 %
15 %Francesco Berteau (fb552) - November 2023
16
17 %initialize matrix with zeros

```

```

18 GMstiffness = zeros(mesh.ngn);
19
20 %calculate and assembly the global stiffness matrix
21 for eN = 1:Ne
22     %local Laplace Element Matrix for diffusion
23     [LMdiffusion] = LEMdiffusion(eN, mesh,GQ,order);
24     %local Laplace Element Matrix for reaction
25     [LMreaction] = LEMreaction(eN, mesh,GQ,order);
26
27     %difference between diffusion and reaction
28     diff = LMdiffusion - LMreaction;
29     %location in GMatrix
30     i = order*eN - (order-1);
31     %add local matrices into global matrix
32     GMstiffness(i:i+order,i:i+order) = GMstiffness(i:i+order,i:i+order) + diff;
33 end
34 end

```

Listing 5. Global source vector calculator

```

1 function [GVsource] = GlobalSourceVector(Ne,mesh,GQ,order)
2 %Assembles the single local source elements into a global vector
3 % The local source elements are computed at each element in the finite
4 % element mesh. Based on their location they are then inserted in the
5 % global source vector of size mesh.ngn-by-1.
6 %
7 % Input:
8 % Ne : Number of elements
9 % mesh : Finite element mesh
10 % GQ : Gaussian Quadrature parameters
11 % order : weather the basis functions is linear or quadratic
12 % Return:
13 % GVsource : Global Source vector
14 %
15 %Francesco Berteau (fb552) - November 2023
16
17 %initialize matrix with zeros
18 GVsource = zeros(mesh.ngn, 1);
19 for eN = 1:Ne
20     %local 2-by-1 vector for corresponding eN
21     [LVsource] = LocalSourceElem(eN,mesh,GQ,order);
22
23     %location in GVector
24     i = order*eN - (order-1);
25     %add local vectors into global vector
26     GVsource(i:i+order,1) = GVsource(i:i+order,1) + LVsource;
27 end
28
29 end

```

Listing 6. Neumann boundary calculator

```

1 function [NB,NBnext] = NeumannBoundary(boundary,mesh)
2 %Computes the Neumann Boundary Conditions
3 % This is done according to the lower and upper boundary values contained
4 % within boundary
5 %
6 % Input:
7 % Boundary : Data structure containing all the boundary conditions
8 % mesh : Finite element mesh
9 % Return:
10 % NB : current Neumann boundary conditions vector
11 % NBnext : next Neumann boundary conditions vector
12 %
13 %Francesco Berteau (fb552) - November 2023
14
15 %initialise boundary conditions vector with zeros
16 NB = zeros(mesh.ngn,1);
17
18 %Neumann boundaries implementation

```

```

19     if isnumeric(boundary.NeumannL)
20         %lower Neumann BC
21         NB(1) = -boundary.NeumannL;
22     elseif isnumeric(boundary.NeumannU)
23         %upper Neumann BC
24         NB(mesh.ngn) = boundary.NeumannU;
25     end
26     %store the next BC
27     NBnext = NB;
28 end

```

Listing 7. Local mass element calculator

```

1 function [LMmass] = LocalMassElem(eN,mesh,GQ,order)
2 %Calculates the Local m-by-n Element Matrix for the linear mass operator
3 % This is done for any element eN in the finite element mesh.
4 %
5 % Input:
6 %   eN : Element number
7 %   mesh : Finite element mesh
8 %   GQ : Gaussian Quadrature parameters
9 %   order : weather the basis functions is linear or quadratic
10 % Return:
11 %   LMmass : Local mass matrix
12 %
13 %Francesco Berteau (fb552) - November 2023
14
15     %Jacobian for a given element
16     J = mesh.elem(eN).J;
17     %initialise mass matrix with zeros
18     LMmass = zeros(order+1);
19     %initialise local element matrix with zeros
20     Int00 = zeros(order+1);
21
22     %if manual integration
23     if GQ.switch == '0'
24         %elements of local element matrix
25         Int00 = 2*J/3;
26         %local 2x2 Element Matrix for mass
27         LMmass = [Int00 Int00/2; Int00/2 Int00];
28
29     %if gaussian quadrature
30     elseif GQ.switch == '1'
31         %number of gauss points
32         N = GQ.npts;
33         %get Gaussian points and weights
34         [GQ] = GQscheme(GQ);
35
36         for i = 1:N
37             w = GQ.gw(i);           %Gauss weights
38             xipts = GQ.xipts(i);    %Gauss points
39             %basis function value at given point
40             [psi,~] = EvalBasis(order,xipts);
41             %local element mass matrix
42             for m = 1:order+1
43                 for n = 1:order+1
44                     Int00(m,n) = psi(m)*psi(n)*J;
45                     LMmass(m,n) = LMmass(m,n) + w*(Int00(m,n));
46                 end
47             end
48         end
49     end
50 end

```

Listing 8. Diffusion local element matrix calculator

```

1 function [DiffusionMatrix] = LEMdiffusion(eN,mesh,GQ,order)
2 %Calculates the Local m-by-n Element Matrix for the linear diffusion operator
3 % This is done for any element eN in the finite element mesh.
4 %

```



```

5 % Input:
6 % eN : Element number
7 % mesh : Finite element mesh
8 % GQ : Gaussian Quadrature parameters
9 % order : weather the basis functions is linear or quadratic
10 % Return:
11 % DiffusionMatrix : Local diffusion matrix
12 %
13 %Francesco Berteau (fb552) - November 2023
14
15 %Jacobian for a given element
16 J = mesh.elem(eN).J;
17 %initialise matrix with zeros
18 DiffusionMatrix = zeros(order+1);
19 %initialise values of matrix with zeros
20 Int00 = zeros(order+1);
21 %diffusion coefficient values
22 D = mesh.elem(eN).D;
23
24 %if manual integration
25 if GQ.switch == '0'
26     %elements of local element matrix 2J = (x1-x0)
27     Int00 = D/(2*J);
28     %local 2x2 Element Matrix for diffusion
29     DiffusionMatrix = [Int00 -Int00 ; -Int00 Int00];
30
31 %if Gaussian Quadrature
32 elseif GQ.switch == '1'
33     %number of gauss points
34     N = GQ.npts;
35     %get Gaussian points and weights
36     [GQ] = GQscheme(GQ);
37
38     for i = 1:N
39         w = GQ.gw(i); %Gauss weights
40         xipts = GQ.xipts(i); %Gauss points
41         %basis function at given point
42         [~,dpsidxiograd] = EvalBasis(order,xipts);
43         %local element diffusion matrix
44         for m = 1:order+1
45             for n = 1:order+1
46                 Int00(m,n) = D*dpsidxiograd(m)*dpsidxiograd(n)/J;
47                 DiffusionMatrix(m,n) = DiffusionMatrix(m,n) + w*(Int00(m,n));
48             end
49         end
50     end
51 end
52 end

```

Listing 9. Reaction local element matrix calculator

```

1 function [ReactionMatrix] = LEMreaction(eN,mesh,GQ,order)
2 %Calculates the Local m-by-n Element Matrix for the linear reaction operator
3 % This is done for any element eN in the finite element mesh.
4 %
5 % Input:
6 % eN : Element number
7 % mesh : Finite element mesh
8 % GQ : Gaussian Quadrature parameters
9 % order : weather the basis functions is linear or quadratic
10 % Return:
11 % ReactionMatrix : Local reaction matrix
12 %
13 %Francesco Berteau (fb552) - November 2023
14
15 %Jacobian of given element
16 J = mesh.elem(eN).J;
17 %initialise matrix with zeros
18 ReactionMatrix = zeros(order+1);
19 %initialise values of matrix with zeros

```

```

20 Int00 = zeros(order +1);
21 %reaction coefficient values
22 lambda = mesh.elem(eN).lambda;
23
24 %if manual integration
25 if GQ.switch == '0'
26     %elements of local element matrix
27     Int00 = 2*(lambda*J)/3;
28     %local 2x2 Element Matrix for reaction
29     ReactionMatrix = [Int00 Int00/2 ; Int00/2 Int00];
30
31 %if gaussian quadrature
32 elseif GQ.switch == '1'
33     %number of gauss points
34     N = GQ.npts;
35     %get Gaussian points and weights
36     [GQ] = GQscheme(GQ);
37
38     for i = 1:N
39         w = GQ.gw(i);           %Gauss weights
40         xipts = GQ.xipts(i);     %Gauss points
41         %basis function value at given point
42         [psi,~] = EvalBasis(order,xipts);
43         %local element reaction matrix
44         for m = 1:order+1
45             for n = 1:order+1
46                 Int00(m,n) = lambda*psi(m)*psi(n)*J;
47                 ReactionMatrix(m,n) = ReactionMatrix(m,n) + w*(Int00(m,n));
48             end
49         end
50     end
51 end
52 end

```

Listing 10. Local source element calculator

```

1 function [LVsource] = LocalSourceElem(eN,mesh,GQ,order)
2 %Calculates the Local m-by-1 Element vector for the source term
3 % This is done for any element eN in the finite element mesh.
4 %
5 % Input:
6 % eN : Element number
7 % mesh : Finite element mesh
8 % GQ : Gaussian Quadrature parameters
9 % order : whether the basis functions is linear or quadratic
10 % Return:
11 % LVsource : Local element source vector
12 %
13 %Francesco Berteau (fb552) - November 2023
14
15 %Jacobian for a given element
16 J = mesh.elem(eN).J;
17 %initialise source vector with zeros
18 LVsource = zeros(order+1,1);
19 %initialise local element vector with zeros
20 Int00 = zeros(order+1);
21 %source term coefficient values
22 f = mesh.elem(eN).f;
23
24 %if manual integration
25 if GQ.switch == '0'
26     %value of elements in matrix
27     Int00 = f*J;
28     %local 2-by-1 vector for corresponding eN
29     LVsource = [Int00;Int00];
30
31 %if gaussian quadrature
32 elseif GQ.switch == '1'
33     %number of gauss points
34     N = GQ.npts;

```

```

35     %get Gaussian points and weights
36     [GQ] = GQscheme(GQ);
37
38     for i = 1:N
39         w = GQ.gw(i);           %Gauss weights
40         xipts = GQ.xipts(i);    %Gauss points
41         %basis function value at given point
42         [psi,~] = EvalBasis(order,xipts);
43         %local element source vector
44         for m = 1:order+1
45             Int00(m) = f*psi(m)*J;
46             LVsource(m) = LVsource(m) + w*(Int00(m));
47         end
48     end
49 end
50 end

```

Listing 11. Gaussian Quadrature scheme generator

```

1 function [gq] = GQscheme(gq)
2 %Creates a Gauss-Legendre Quadrature scheme data structure of order N
3 % The scheme stores both the quadrature weights and the Legendre points.
4 % Values found @
5 % https://en.wikipedia.org/wiki/Gauss-Legendre\_quadrature#Definition
6 %
7 % Input:
8 % GQ : Gaussian Quadrature initial parameters
9 % Return:
10 % GQ : Gaussian Quadrature with weights and points
11 %
12 %Francesco Berteau (fb552) - November 2023
13
14 %order of quadrature rule
15 N = gq.npts;
16
17 if (N>0) && (N<6)
18     %initialise zero array of size equal gauss quadrature order
19     gq.gw = zeros(N,1); %Gauss weights
20     gq.xipts = zeros(N,1); %Gauss points
21
22     switch N
23     case 1
24         gq.gw(1) = 2;
25         gq.xipts(1) = 0;
26     case 2
27         gq.gw(1) = 1;
28         gq.gw(2) = 1;
29         gq.xipts(1) = -sqrt(1/3);
30         gq.xipts(2) = sqrt(1/3);
31     case 3
32         gq.gw(1) = 5/9;
33         gq.gw(2) = 8/9;
34         gq.gw(3) = 5/9;
35         gq.xipts(1) = -sqrt(3/5);
36         gq.xipts(2) = 0;
37         gq.xipts(3) = sqrt(3/5);
38     case 4
39         gq.gw(1) = (18+sqrt(30))/36;
40         gq.gw(2) = (18+sqrt(30))/36;
41         gq.gw(3) = (18-sqrt(30))/36;
42         gq.gw(4) = (18-sqrt(30))/36;
43         gq.xipts(1) = -sqrt((3/7)-(2/7)*sqrt(6/5));
44         gq.xipts(2) = sqrt((3/7)-(2/7)*sqrt(6/5));
45         gq.xipts(3) = -sqrt((3/7)+(2/7)*sqrt(6/5));
46         gq.xipts(4) = sqrt((3/7)+(2/7)*sqrt(6/5));
47     case 5
48         gq.gw(1) = 128/225;
49         gq.gw(2) = (322+13*sqrt(70))/900;
50         gq.gw(3) = (322+13*sqrt(70))/900;
51         gq.gw(4) = (322-13*sqrt(70))/900;

```

```

52         gq.gw(5) = (322-13*sqrt(70))/900;
53         gq.xipts(1) = 0.0;
54         gq.xipts(2) = -(1/3)*sqrt(5-2*sqrt(10/7));
55         gq.xipts(3) = (1/3)*sqrt(5-2*sqrt(10/7));
56         gq.xipts(4) = -(1/3)*sqrt(5+2*sqrt(10/7));
57         gq.xipts(5) = (1/3)*sqrt(5+2*sqrt(10/7));
58     end
59 else
60     fprintf('Invalid number of Gauss points (N)');
61 end
62 end

```

Listing 12. Basis function evaluator

```

1 function [psi, dpsidxigrad] = EvalBasis(order,xipts)
2 %Evaluates the basis functions used in the model
3 % The basis functions are evaluated by calculating their values and gradients
4 % for a given local node (lnid) at a xipts.
5 %
6 % Input:
7 % order : weather the basis functions is linear or quadratic
8 % xipts : x points between [-1,1]
9 % Return:
10 % psi : the value of the basis function
11 % nvecnext : the gradient of the basis function
12 %
13 %Francesco Berteau (fb552) - November 2023
14
15 %initialise psi as zeros array, with size order + 1
16 psi = zeros(order+1,1);
17 dpsidxigrad = zeros(order+1,1);
18
19 %linear or quadratic basis functions
20 switch order
21     %linear
22     case 1
23         %function values
24         for lnid = 0:1
25             sign = (-1)^(lnid+1);
26             psi(lnid+1) = 0.5*(1+((sign*xipts)));
27         end
28         %function gradients
29         for lnid = 1:order+1
30             sign1 = (-1)^(lnid);
31             dpsidxigrad(lnid) = 0.5 * sign1;
32         end
33     %quadratic
34     case 2
35         %function values
36         psi(1) = xipts*(xipts - 1)*0.5;
37         psi(2) = (1 - xipts^2);
38         psi(3) = xipts*(xipts + 1)*0.5;
39         %function gradients
40         dpsidxigrad(order-1) = xipts-0.5;
41         dpsidxigrad(order) = -2*xipts;
42         dpsidxigrad(order+1) = xipts+0.5;
43     end
44 end

```

Listing 13. Dirichlet boundary calculator

```

1 function [RHS,GlobalMatrix] = DirichletBoundary(boundary,RHS,GlobalMatrix)
2 %Applies Dirichlet Boundary Conditions to transient diffusion equation.
3 % Sets the global matrix rows corresponding to boundary nodes to zero and
4 % the RHS vector to the boundaries values.
5 %
6 % Input:
7 % Boundary : Data structure containing all the boundary conditions
8 % RHS : RHS vector
9 % GlobalMatrix : Global Matrix

```

```

10 % Return:
11 % RHS : RHS vector with DBC applied
12 % GlobalMatrix : Global Matrix with DBC applied
13 %
14 %Francesco Berteau (fb552) - November 2023
15
16 %upper Dirichlet BC
17 GlobalMatrix(end, :) = 0;
18 GlobalMatrix(end) = 1;
19 RHS(end) = boundary.DirichletU;
20
21 %lower Dirichlet BC
22 GlobalMatrix(1, :) = 0;
23 GlobalMatrix(1, 1) = 1;
24 RHS(1) = boundary.DirichletL;
25
26 end

```

Listing 14. Main script running the equation solver

```

1 %This script runs the transient FEM solver and plots some graphs used to
2 % prove the code functioning for Part 1 of ME40064, Transient MATLAB-Based
3 % FEM Modelling, CourseWork.
4 %
5 %Francesco Berteau (fb552) - November 2023
6 close all
7
8 %% ----- Input parameters -----%
9
10 % Parameters of current material
11 parameters.selection = '1'; %material for Part 1
12 parameters.D = 1; %Diffusion coefficient
13 parameters.lambda = 0; %Reaction coefficient
14 parameters.f = 0; %Source term
15
16 Xmin = 0; %lower spatial boundary
17 Xmax = 1; %upper spatial boundary
18 Ne = 10; %number of elements
19 order = 1; %order (linear(1) or quadratic(2)) of basis function
20 theta = 1; %0:Forward Euler, 1:Backward Euler, 0.5:Crank-Nicolson
21 Xpoints = Xmin:0.01:Xmax; %vector with x points for analytical plotting
22
23 % Time related values
24 time.tmin = 0; %start time
25 time.tmax = 1; %end time
26 time.ic = 0; %initial condition time
27 time.dt = 0.01; %delta t
28 time.t = time.tmin:time.dt:time.tmax; %entire time vector
29 time.N = (time.tmax-time.tmin)/time.dt; %number of time steps
30 tpoints = [0.05 0.1 0.3 1.0]; %given time points
31
32 GQ.switch = '1'; %Gaussian Quadrature yes or no
33 GQ.npts = 2; %number of gauss points (2N-1)
34 [GQ] = GQscheme(GQ); %create Gaussian Quadrature
35
36 % All boundaries combined
37 boundary.DirichletL = 0; %Lower Dirichlet Boundary Condition
38 boundary.DirichletU = 1; %Upper Dirichlet Boundary Condition
39 boundary.NeumannL = 'Na'; %Lower Neumann Boundary Condition
40 boundary.NeumannU = 'Na'; %Upper Neumann Boundary Condition
41
42 %get numerical solution for the transient FEM
43 [Cnum, mesh, GQ, time] = TransientFEM(Xmin, Xmax, Ne, order, theta, time, GQ, boundary, parameters);
44
45
46 %% ----- Graphs for 1.1 -----%
47 %first figure for computed solution alone
48 figure('Name', 'Part1 numerical')
49 for i = 1:length(tpoints)
50

```

```

51     %Plot c(x) against x
52     element = time.t == tpoints(i);
53     plot(mesh.nvec,Cnum(:,element),'DisplayName',(strcat('t=',num2str(tpoints(i)))), 'LineWidth',1.3)
54     ;
55     hold on
56 end
57 grid on %use grid lines
58 title('Transient FEM numerical solutions','FontSize',14)
59 xlabel('Spatial distance x','FontSize',12)
60 ylabel('Concentration c(x,t)','FontSize',12)
61 legend('Location','NorthWest','FontSize',10)
62 % save plot as picture
63 saveas(gcf,'TransientFEM','png')
64
65 %second figure for analytical vs numerical comparison
66 figure('Name', 'Part1 analytical')
67 %Defines line colours using MATLAB RGB triplet
68 colours = {[0.00 0.45 0.74],[0.85 0.33 0.10], [0.93 0.69 0.13], [0.49 0.18 0.56]};
69 Canalytical = zeros(length(Xpoints),length(tpoints));
70
71 for i = 1:length(tpoints)
72     for m = 1:length(Xpoints)
73         %Calculates analytical solution
74         Canalytical(m,i) = TransientAnalyticSoln(Xpoints(m),tpoints(i));
75     end
76
77     %plot analytical solution against x
78     plot(Xpoints, Canalytical(:,i),'DisplayName',strcat('Analytical Solution @ t=',num2str(tpoints(i)
79     )), 'LineStyle','--', 'LineWidth',1.3, 'color', colours{i});
80     hold on
81
82     %plot c(x) against x
83     element = time.t == tpoints(i);
84     plot(mesh.nvec,Cnum(:,element),'DisplayName',(strcat('Numerical Solution @ t=',num2str(tpoints(i)
85     )), 'LineWidth',1.3, 'color', colours{i});
86     hold on
87 end
88 grid on %use grid lines
89 title('Numerical vs Analytical solutions','FontSize',14)
90 xlabel('Spatial distance x','FontSize',12)
91 ylabel('Concentration c(x,t)','FontSize',12)
92 legend('Location','NorthWest','FontSize',10)
93 % save plot as picture
94 saveas(gcf,'TransientFEM-analytical','png')
95
96 %% ----- Graphs for 1.2 ----- %
97 %figure for analytical vs numerical comparison @ x = 0.8
98 figure('Name','Part1 analytical @ 0.8')
99
100 element = mesh.nvec == 0.8;
101 plot(time.t,Cnum(element,:), 'DisplayName','Numerical Solution @ x=0.8', 'LineWidth',1.3);
102 hold on
103
104 Canalytical = TransientAnalyticSoln(0.8,time.t);
105 plot(time.t,Canalytical,'DisplayName','Analytical Solution @ x=0.8', 'LineWidth',1.3);
106 hold on
107
108 grid on %use grid lines
109 title('Numerical vs Analytical solution at x=0.8','FontSize',14)
110 xlabel('Time t (s)','FontSize',12)
111 ylabel('Concentration c(0.8,t)','FontSize',12)
112 legend('Location','SouthEast','FontSize',10)
113 % save plot as picture
114 saveas(gcf,'TransientFEM-analytical@08','png')
115
116 %% ----- Backward Euler vs Crank-Nicolson ----- %
117 %figure for different theta scheme comparison @ x = 0.8
118 figure('Name','Crank-Nicolson, Backward and Forward @ x = 0.8')

```

```

117 for theta = [0 0.5 1]
118     %get numerical solution for given theta
119     [Cnum,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);
120
121     element = mesh.nvec == 0.8;
122     plot(time.t,Cnum(element,:), '-x', 'DisplayName', strcat('Numerical Solution   ', num2str(theta)), '
        LineWidth', 1.3);
123     hold on
124 end
125
126 %Plots numerical solution at x = 0.8 for the established time interval
127 Canalytical = TransientAnalyticSoln(0.8,time.t);
128 plot(time.t,Canalytical, 'DisplayName', 'Analytical Solution', 'LineWidth', 1.3);
129
130 grid on %use grid lines
131 title('Crank-Nicolson vs Euler methods', 'FontSize', 14)
132 xlabel('Time t (s)', 'FontSize', 12)
133 ylabel('Concentration c(0.8,t)', 'FontSize', 12)
134 legend('Location', 'SouthEast', 'FontSize', 10)
135 ylim([0,0.8])
136 xlim([0,0.2])
137 % save plot as picture
138 saveas(gcf, 'Crank-Nicolson_vs_Euler', 'png')
139
140 %% ----- Linear vs Quadratic basis function ----- %
141 %figure for different basis function order
142 figure('Name', 'Basis function order')
143
144 Canalytical = zeros(length(Xpoints),1);
145 for m = 1:length(Xpoints)
146     %Calculates analytical solution
147     Canalytical(m,1) = TransientAnalyticSoln(Xpoints(m),0.05);
148 end
149 %plot analytical solution against x
150 plot(Xpoints, Canalytical(:,1), 'DisplayName', 'Analytical Solution', 'LineWidth', 1.3);
151 hold on
152
153 for order = [1 2]
154     [Cnum,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);
155     %plot c(x) against x
156     element = time.t == 0.05;
157     plot(mesh.nvec,Cnum(:,element), '-x', 'DisplayName', (strcat('Numerical Solution order=', num2str(
        order))), 'LineWidth', 1.3);
158     hold on
159 end
160
161 grid on %use grid lines
162 title('Basis Function order', 'FontSize', 14)
163 xlabel('Spatial distance x', 'FontSize', 12)
164 ylabel('Concentration c(x,0.05)', 'FontSize', 12)
165 legend('Location', 'NorthWest', 'FontSize', 10)
166 xlim([0.4,0.8])
167 % save plot as picture
168 saveas(gcf, 'BasisFunctionOrder', 'png')

```

Listing 15. Analytical solution calculator

```

1 function [ c ] = TransientAnalyticSoln(x,t)
2 %TransientAnalyticSoln Analytical solution to transient diffusion equation
3 % Computes the analytical solution to the transient diffusion equation for
4 % the domain x=[0,1], subject to initial condition: c(x,0) = 0, and Dirichlet
5 % boundary conditions: c(0,t) = 0, and c(1,t) = 1.
6 % Input Arguments:
7 % x is the point in space to evaluate the solution at
8 % t is the point in time to evaluate the solution at
9 % Output Argument:
10 % c is the value of concentration at point x and time t, i.e. c(x,t)
11
12 trans = 0.0;
13

```

```

14 for k=1:1000
15     trans = trans + (((-1)^k)/k) * exp(-k^2*pi^2*t)*sin(k*pi*x);
16 end
17
18 c = x + (2/pi)*trans;
19
20 end

```

Listing 16. L2-norm calculator

```

1 function [L2Norm,h] = L2Norm(Xmin,Xmax,elements,order,theta,time,GQ,boundary,parameters)
2 %Computes the L2 Norm of the transient diffusion reaction equation.
3 % The L2 Norm is the error between the numerical and the analytical
4 % solution. This is done to test convergence of the TransientFEM calculator
5 % over a range of elements.
6 %
7 % Input:
8 %   Xmin : Lower spatial boundary
9 %   Xmax : Upper spatial boundary
10 %   elements : given elements
11 %   order : weather the basis functions is linear or quadratic
12 %   theta : Method selection
13 %   time : All time related values combined
14 %   GQ : Gaussian Quadrature parameters
15 %   boundary : Dirichlet and Neumann conditions combined
16 %   parameters : Parameters for current material
17 % Return:
18 %   L2Norm : L2 Norm error
19 %   h : Characteristic length
20 %
21 %Francesco Berteau (fb552) - November 2023
22
23     %number of gauss points
24     N = GQ.npts;
25     %preallocations for fast processing
26     h = zeros(1,length(elements));
27     L2Norm = zeros(1,length(elements));
28     E = zeros(1,N);
29
30     for b = 1:length(elements)
31         %Defines number of elements
32         Ne = elements(b);
33         Etot = zeros(1,Ne); %preallocation for fast processing
34         %length for each element
35         h(b) = (Xmax-Xmin)/Ne;
36         %solve transient FEM
37         [Cnum,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);
38         %Returns the solution column at the selected time
39         cnumerical = Cnum(:,time.t == time.range);
40         for eN = 1:Ne
41             %Jacobian for a given element
42             J = mesh.elem(eN).J;
43             %rows of the local nodes at given eN
44             Cnoderows = cnumerical(mesh.elem(eN).n(1):mesh.elem(eN).n(end),1);
45             for i = 1:N
46                 w = GQ.gw(i); %Gauss weights
47                 xipts = GQ.xipts(i); %Gauss points
48                 %basis function at given point
49                 [psi,~] = EvalBasis(order,xipts);
50                 %interpolation between nodes
51                 CNum = psi'*Cnoderows;
52                 %Finds x position
53                 x = mesh.elem(eN).x*psi;
54                 %analytical solution
55                 CAnalyt = TransientAnalyticSoln(x,time.range);
56                 %error squared
57                 E(i) = w*J*(CAnalyt-CNum)^2;
58             end
59             %total error
60             Etot(eN) = sum(E);

```



```

61     end
62     %L2 Norm is RMS of total error
63     L2Norm(b) = sqrt(sum(Etot));
64 end
65 %coefficients of polynomial to fit the data
66 y = polyfit(log(h),log(L2Norm),1);
67 %error gradient
68 gradient = y(1);
69 end

```

Listing 17. Script running the L2-norm

```

1 %This script runs the transient FEM solver and plots the L2 norm used to
2 % test the convergence rate of the finite element method function.
3 %
4 %Francesco Berteau (fb552) - November 2023
5 close all
6
7 %----- Input parameters for L2 norm -----%
8
9 % Parameters of current material
10 parameters.selection = '1'; %material for Part 1
11 parameters.D = 1; %Diffusion coefficient
12 parameters.lambda = 0; %Reaction coefficient
13 parameters.f = 0; %Source term
14
15 Xmin = 0; %lower spatial boundary
16 Xmax = 1; %upper spatial boundary
17 Ne = 10; %number of elements
18 elements = [2 5 10 15 20 25 30 35 40 45]; %chosen elements
19
20 % Time related values
21 time.tmin = 0; %start time
22 time.tmax = 1; %end time
23 time.ic = 0; %initial condition time
24 time.dt = 0.0001; %delta t
25 time.t = time.tmin:time.dt:time.tmax; %entire time vector
26 time.N = (time.tmax-time.tmin)/time.dt; %number of time steps
27 tpoints = [0.05 0.1 0.3 1.0]; %given time points
28
29 GQ.switch = '1'; %Gaussian Quadrature yes or no
30 GQ.npts = 3; %number of gauss points (2N-1)
31 [GQ] = GQscheme(GQ); %create Gaussian Quadrature
32
33 % All boundaries combined
34 boundary.DirichletL = 0; %Lower Dirichlet Boundary Condition
35 boundary.DirichletU = 1; %Upper Dirichlet Boundary Condition
36 boundary.NeumannL = 'Na'; %Lower Neumann Boundary Condition
37 boundary.NeumannU = 'Na'; %Upper Neumann Boundary Condition
38
39
40 %----- Linear Vs Quadratic Basis Function -----%
41 %Loops through selected elements range switching between linear and quadratic basis functions
42 for order = [1 2]
43     figure('Name',strcat('L2 Norm test, basis order = ',num2str(order)))
44     %predefined line colours [rgb]
45     linecolor = {[0.00 0.45 0.75],[0.85 0.35 0.10], [0.95 0.70 0.15], [0.50 0.20 0.55]};
46     %Crank-Nicolson or Backward Euler methods
47     for theta = [0.5 1]
48
49         for i = 1:length(tpoints)
50             %get L2 Norm error
51             time.range = tpoints(i);
52             [L2N, h] = L2Norm(Xmin,Xmax,elements,order,theta,time,GQ,boundary,parameters);
53             %Crank-Nicolson or Backward Euler methods
54             switch theta
55                 case 0.5
56                     %plot L2 Norm error against characteristic length using log-log plot
57                     loglog(h,L2N,'DisplayName',strcat('t=',num2str(time.range)), 'LineStyle','-.', '
LineWidth',1.3, 'color',linecolor{i});

```

```

58         case 1
59             %plot L2 Norm error against characteristic lenght using log-log plot
60             loglog(h,L2N,'DisplayName',strcat('t=',num2str(time.range)),'LineStyle','--','
        LineWidth',1.3,'color',linecolor{i});
61         end
62         hold on
63     end
64 end
65 grid on %use grid lines
66 title(strcat('L2 Norm with basis order=',num2str(order)),'FontSize',14)
67 xlabel('ln(h)','FontSize',12);
68 ylabel('ln(E)','FontSize',12);
69 legend('Location','SouthEast','FontSize',10,'NumColumns',2)
70 % save plot as picture
71 saveas(gcf,strcat('L2Norm_order=',num2str(order)),'png')
72 end
73
74
75 %----- Manual integration Vs. Gaussian Quadrature -----%
76 order = 1;
77 time.range = time.tmax;
78 figure('Name','Numerical Methods Convergence')
79 %manual or gaussian quadrature integration methods
80 for j = [0 1]
81     GQ.Switch = num2str(j);
82     %Crank-Nicolson or Backward Euler methods
83     for theta = [0.5 1]
84         %get L2 Norm error
85         [L2N, h] = L2Norm(Xmin,Xmax,elements,order,theta,time,GQ,boundary,parameters);
86         %two line colours
87         if theta == 0.5
88             linecolor = [0.00 0.45 0.75];
89         elseif theta == 1
90             linecolor = [0.85 0.35 0.10];
91         end
92         %manual or gaussian quadrature
93         switch j
94             case 0
95                 %plot L2 Norm error against characteristic lenght using log-log plot
96                 loglog(h,L2N,'LineWidth',1.3,'color',linecolor);
97             case 1
98                 %plot L2 Norm error against characteristic lenght using log-log plot
99                 loglog(h,L2N,'--','LineWidth',1.3,'color',linecolor);
100         end
101         hold on
102     end
103 end
104 grid on %use grid lines
105 title('Numerical Methods Convergence','FontSize',14)
106 xlabel('ln(h)','FontSize',12);
107 ylabel('ln(E)','FontSize',12)
108 legend('Crank-Nicolson (Manual Integration)','Backward Euler (Manual Integration)',...
109 'Crank-Nicolson (Gaussian Quadrature)','Backward Euler (Gaussian Quadrature)','Location','SouthEast'
110 , 'FontSize',10)
111 % save plot as picture
112 saveas(gcf,'NumericalMethodsConvergence','png')

```

Listing 18. Main script running the skin drug diffusion analysis

```

1 %This script solves the drug delivery through uman skin problem.
2 % Various graphs are used to show the results and to do some analysis for
3 % Part 2 of ME40064, Transient MATLAB-Based FEM Modelling, CourseWork.
4 %
5 %Francesco Berteau (fb552) - November 2023
6 close all
7
8 %% ----- Input parameters -----%
9
10 % Parameters of current material
11 parameters.selection = '2'; %material for Part 2

```

```

12 parameters.De = 4e-6;           %Diffusion coefficient epidermis
13 parameters.Dd = 5e-6;           %Diffusion coefficient dermis
14 parameters.Db = 2e-6;           %Diffusion coefficient sub-cutaneous
15 parameters.betaE = 0;           %Reaction coefficient blood epidermis
16 parameters.betaD = 0.01;        %Reaction coefficient blood dermis
17 parameters.betaB = 0.01;        %Reaction coefficient blood sub-cutaneous
18 parameters.gammaE = 0.02;       %Reaction coefficient degradation epidermis
19 parameters.gammaD = 0.02;       %Reaction coefficient degradation dermis
20 parameters.gammaB = 0.02;       %Reaction coefficient degradation sub-cutaneous
21 parameters.Xe = 0.00166667;     %Epidermis x coordinate
22 parameters.Xd = 0.005;         %Dermis x coordinate
23 parameters.Xb = 0.01;          %Sub-cutaneous x coordinate
24
25 % Space related values
26 Xmin = 0;    %lower spatial boundary
27 Xmax = 0.01; %upper spatial boundary
28 Ne = 40;     %number of elements
29 order = 2;   %order (linear(1) or quadratic(2)) of basis function
30 theta = 0.5; %0:Forward Euler, 1:Backward Euler, 0.5:Crank-Nicolson
31
32 % Time related values
33 time.tmin = 0; %start time
34 time.tmax = 30; %end time
35 time.ic = 0;   %initial condition time
36 time.dt = 0.01; %delta t
37 time.t = time.tmin:time.dt:time.tmax; %entire time vector
38 time.N = (time.tmax-time.tmin)/time.dt; %number of time steps
39 tpoints = [0.05 0.1 0.3 1 2 3 5 7 10 20 30]; %given time points
40
41 GQ.switch = '1'; %Gaussian Quadrature yes or no
42 GQ.npts = 4;    %number of gauss points (2N-1)
43 [GQ] = GQscheme(GQ); %create Gaussian Quadrature
44
45 % All boundaries combined
46 boundary.DirichletL = 30; %Lower Dirichlet Boundary Condition
47 boundary.DirichletU = 0; %Upper Dirichlet Boundary Condition
48 boundary.NeumannL = 'Na'; %Lower Neumann Boundary Condition
49 boundary.NeumannU = 'Na'; %Upper Neumann Boundary Condition
50
51 %colours for plotting (parula colormap)
52 colours = parula(length(tpoints));
53
54 %Calculates numerical solution for the transient problem
55 [c,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);
56
57 %% ----- Drug concentration -----%%
58 figure('Name', 'Concentration')
59
60 %Loop through all the selected time points
61 for i = 1:length(tpoints)
62
63     %Plots temperature distribution through the tissue for a range of
64     %time points
65     element = time.t == tpoints(i);
66     plot(mesh.nvec,c(:,element),'DisplayName',(strcat('t=',num2str(tpoints(i)))), 'LineWidth',1.3, '
67     color',colours(i,:));
68     hold on
69 end
70
71 %show where the epidermis and dermis end
72 xline(parameters.Xe,'-k','Epidermis','FontSize',12,'LineWidth',0.75,'HandleVisibility','off');
73 xline(parameters.Xd,'-k','Dermis','FontSize',12,'LineWidth',0.75,'HandleVisibility','off');
74
75 grid on %use grid lines
76 title('Drug concentration through the skin','FontSize',14)
77 xlabel('Skin Depth (m)','FontSize',12);
78 ylabel('Concentration c(x,t)','FontSize',12);
79 legend('Location','NorthEast','FontSize',10,'NumColumns',2)
80 ylim([0,35])

```

```

80 % save plot as picture
81 saveas(gcf,'DrugConcentration','png')
82
83
84 %% ----- Drug effectiveness -----
85 figure('Name', 'Effectiveness')
86
87 % concentration for effectiveness at given position
88 ceff = 40;
89 Xpos = 0.005;
90
91 % increase boundary dose untill integral of c is less that 1000
92 K = 0;
93 while K < 1000
94     [c,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);
95
96     [K,teff,position,element] = MinEffectiveDose(mesh,time,order,c,Xpos,ceff);
97
98     % increase dose applied to skin
99     boundary.DirichletL = boundary.DirichletL + 1;
100 end
101
102 plot(time.t,c(element,:), 'DisplayName', 'Concentration @ x=0.005', 'LineWidth', 1.3);
103 hold on
104 plot(teff,c(element,position), 'ro', 'MarkerSize', 10, 'DisplayName', 'Minimum effective dose')
105 % Add label with coordinates
106 label = sprintf(['%.2f, %.1f'], teff, c(element,position));
107 text(teff + 1, c(element,position) - 1, label, 'FontSize', 10);
108
109 grid on %use grid lines
110 title('Drug effectiveness at Dermis end', 'FontSize', 14)
111 xlabel('Time t (s)', 'FontSize', 12);
112 ylabel('Concentration c(0.005,t)', 'FontSize', 12);
113 legend('Location', 'SouthEast', 'FontSize', 10)
114 % save plot as picture
115 saveas(gcf,'DrugEffect','png')
116
117 fprintf("Minimum initial dose: %d \n", boundary.DirichletL);
118
119 %% ----- Effective drug concentration -----
120 figure('Name', 'Concentration')
121
122 %Loop through all the selected time points
123 for i = 1:length(tpoints)
124
125     %Plots temperature distribution through the tissue for a range of
126     %time points
127     element = time.t == tpoints(i);
128     plot(mesh.nvec,c(:,element), 'DisplayName', (strcat('t=', num2str(tpoints(i)))), 'LineWidth', 1.3, 'color', colours(i,:));
129     hold on
130 end
131
132 %show where the epidermis and dermis end
133 xline(parameters.Xe, '-.k', 'Epidermis', 'FontSize', 12, 'LineWidth', 0.75, 'HandleVisibility', 'off');
134 xline(parameters.Xd, '-.k', 'Dermis', 'FontSize', 12, 'LineWidth', 0.75, 'HandleVisibility', 'off');
135 %show the minimum concentration for effectiveness
136 yline(ceff, '--r', 'Effectiveness', 'FontSize', 12, 'LineWidth', 1, 'HandleVisibility', 'off')
137
138 grid on %use grid lines
139 title('Effective drug concentration through the skin', 'FontSize', 14)
140 xlabel('Skin Depth (m)', 'FontSize', 12);
141 ylabel('Concentration c(x,t)', 'FontSize', 12);
142 legend('Location', 'NorthEast', 'FontSize', 10, 'NumColumns', 2)
143 ylim([0,75])
144 % save plot as picture
145 saveas(gcf,'EffectiveDrugConcentration','png')
146
147 %% ----- Parameters influence -----

```

```

148 figure('Name', 'Parameters')
149
150 for i = [1 2 3 4 5 6 7 8]
151     parameters = CombinationsDR(i,parameters);
152     boundary.DirichletL = 30;
153     % increase boundary dose untill integral of c is less than 1000
154     K = 0;
155     while K < 1000
156         [c,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);
157
158         [K,teff,position,element] = MinEffectiveDose(mesh,time,order,c,Xpos,ceff);
159
160         % increase dose applied to skin
161         boundary.DirichletL = boundary.DirichletL + 1;
162     end
163
164     fprintf("Minimum initial dose: %d, with combination %d \n",boundary.DirichletL,i);
165     plot(time.t,c(element,:), 'DisplayName', strcat('Solution for combination: ',num2str(i)), '
166         LineWidth',1.3);
167     hold on
168
169     yline(ceff,'--r','LineWidth',0.75,'DisplayName','Minimum effective dose')
170     grid on %use grid lines
171     title('Drug effectiveness with different combinations','FontSize',14)
172     xlabel('Time t (s)','FontSize',12);
173     ylabel('Concentration c(0.005,t)','FontSize',12);
174     legend('Location','SouthEast','FontSize',10)
175     ylim([0,45])
176     % save plot as picture
177     saveas(gcf,'ParameterComparison','png')

```

Listing 19. Minimum effective dose calculator

```

1 function [K,teff,position,element] = MinEffectiveDose(mesh,time,order,c,Xpos,ceff)
2 %Computes the Minimum effective drug dose
3 % Given a position in the mesh and a value for the concentration for
4 % effectiveness this returns the integral of the curve between the
5 % effectiveness point and the end.
6 %
7 % Input:
8 % mesh : Finite element mesh
9 % time : All time related values combined
10 % order : whether the basis functions is linear or quadratic
11 % c : Solution to the full transient FEM
12 % Xpos : Position along the mesh
13 % ceff : effectiveness concentration
14 % Return:
15 % K : current Neumann boundary conditions vector
16 % teff : time of effectiveness
17 % position : index of given Xpos
18 % element : index of effective time
19 %
20 %Francesco Berteau (fb552) - November 2023
21
22 % find in c(Xpos,:) where the effectiveness concentration is reached
23 element = round(order*Xpos/(mesh.nvec(end)/mesh.ne));
24 position = find(c(element,:) > ceff,1,'first');
25 % use the index to return a time point
26 teff = time.t(position);
27
28 % compute integral between current time and final of c in dt
29 K = trapz(c(element,position:end))*time.dt;
30 end

```

Listing 20. Combination of different diffusion reaction coefficients

```

1 function [parameters] = CombinationsDR(I,parameters)
2 %Stores a series of parameters combinations for diffusion and reaction coefficients.
3 % Changes the parameters data structure to have different coefficients for

```

```

4 % diffusion, beta and gamma. Entera a combination value in range [1:1:8]
5 %
6 % Input:
7 % I : combination number
8 % parameters : current parameters data structure
9 % Return:
10 % parameters : new parameters data structure
11 %
12 %Francesco Berteau (fb552) - November 2023
13
14 % this function should only be called in material 2 testing
15 parameters.selection = '2'; %material for Part 2
16
17 switch I
18     case 1 % High diffusion
19         parameters.De = 4e-5; %Diffusion coefficient epidermis
20         parameters.Dd = 5e-5; %Diffusion coefficient dermis
21         parameters.Db = 2e-5; %Diffusion coefficient sub-cutaneous
22         parameters.betaE = 0; %Reaction coefficient blood epidermis
23         parameters.betaD = 0.01; %Reaction coefficient blood dermis
24         parameters.betaB = 0.01; %Reaction coefficient blood sub-cutaneous
25         parameters.gammaE = 0.02; %Reaction coefficient degradation epidermis
26         parameters.gammaD = 0.02; %Reaction coefficient degradation dermis
27         parameters.gammaB = 0.02; %Reaction coefficient degradation sub-cutaneous
28     case 2 % Low diffusion
29         parameters.De = 4e-7; %Diffusion coefficient epidermis
30         parameters.Dd = 5e-7; %Diffusion coefficient dermis
31         parameters.Db = 2e-7; %Diffusion coefficient sub-cutaneous
32         parameters.betaE = 0; %Reaction coefficient blood epidermis
33         parameters.betaD = 0.01; %Reaction coefficient blood dermis
34         parameters.betaB = 0.01; %Reaction coefficient blood sub-cutaneous
35         parameters.gammaE = 0.02; %Reaction coefficient degradation epidermis
36         parameters.gammaD = 0.02; %Reaction coefficient degradation dermis
37         parameters.gammaB = 0.02; %Reaction coefficient degradation sub-cutaneous
38     case 3 % high degradation
39         parameters.De = 4e-6; %Diffusion coefficient epidermis
40         parameters.Dd = 5e-6; %Diffusion coefficient dermis
41         parameters.Db = 2e-6; %Diffusion coefficient sub-cutaneous
42         parameters.betaE = 0; %Reaction coefficient blood epidermis
43         parameters.betaD = 0.01; %Reaction coefficient blood dermis
44         parameters.betaB = 0.01; %Reaction coefficient blood sub-cutaneous
45         parameters.gammaE = 0.2; %Reaction coefficient degradation epidermis
46         parameters.gammaD = 0.2; %Reaction coefficient degradation dermis
47         parameters.gammaB = 0.2; %Reaction coefficient degradation sub-cutaneous
48     case 4 % high degradation, low blood
49         parameters.De = 4e-6; %Diffusion coefficient epidermis
50         parameters.Dd = 5e-6; %Diffusion coefficient dermis
51         parameters.Db = 2e-6; %Diffusion coefficient sub-cutaneous
52         parameters.betaE = 0; %Reaction coefficient blood epidermis
53         parameters.betaD = 0.001; %Reaction coefficient blood dermis
54         parameters.betaB = 0.001; %Reaction coefficient blood sub-cutaneous
55         parameters.gammaE = 0.2; %Reaction coefficient degradation epidermis
56         parameters.gammaD = 0.2; %Reaction coefficient degradation dermis
57         parameters.gammaB = 0.2; %Reaction coefficient degradation sub-cutaneous
58     case 5 % low degradation, high blood
59         parameters.De = 4e-6; %Diffusion coefficient epidermis
60         parameters.Dd = 5e-6; %Diffusion coefficient dermis
61         parameters.Db = 2e-6; %Diffusion coefficient sub-cutaneous
62         parameters.betaE = 0; %Reaction coefficient blood epidermis
63         parameters.betaD = 0.1; %Reaction coefficient blood dermis
64         parameters.betaB = 0.1; %Reaction coefficient blood sub-cutaneous
65         parameters.gammaE = 0.002; %Reaction coefficient degradation epidermis
66         parameters.gammaD = 0.002; %Reaction coefficient degradation dermis
67         parameters.gammaB = 0.002; %Reaction coefficient degradation sub-cutaneous
68     case 6 % high blood
69         parameters.De = 4e-6; %Diffusion coefficient epidermis
70         parameters.Dd = 5e-6; %Diffusion coefficient dermis
71         parameters.Db = 2e-6; %Diffusion coefficient sub-cutaneous
72         parameters.betaE = 0; %Reaction coefficient blood epidermis

```

```

73     parameters.betaD = 0.1; %Reaction coefficient blood dermis
74     parameters.betaB = 0.1; %Reaction coefficient blood sub-cutaneous
75     parameters.gammaE = 0.02; %Reaction coefficient degradation epidermis
76     parameters.gammaD = 0.02; %Reaction coefficient degradation dermis
77     parameters.gammaB = 0.02; %Reaction coefficient degradation sub-cutaneous
78     case 7 % low blood
79         parameters.De = 4e-6; %Diffusion coefficient epidermis
80         parameters.Dd = 5e-6; %Diffusion coefficient dermis
81         parameters.Db = 2e-6; %Diffusion coefficient sub-cutaneous
82         parameters.betaE = 0; %Reaction coefficient blood epidermis
83         parameters.betaD = 0.001; %Reaction coefficient blood dermis
84         parameters.betaB = 0.001; %Reaction coefficient blood sub-cutaneous
85         parameters.gammaE = 0.02; %Reaction coefficient degradation epidermis
86         parameters.gammaD = 0.02; %Reaction coefficient degradation dermis
87         parameters.gammaB = 0.02; %Reaction coefficient degradation sub-cutaneous
88     case 8 % low degradation
89         parameters.De = 4e-6; %Diffusion coefficient epidermis
90         parameters.Dd = 5e-6; %Diffusion coefficient dermis
91         parameters.Db = 2e-6; %Diffusion coefficient sub-cutaneous
92         parameters.betaE = 0; %Reaction coefficient blood epidermis
93         parameters.betaD = 0.01; %Reaction coefficient blood dermis
94         parameters.betaB = 0.01; %Reaction coefficient blood sub-cutaneous
95         parameters.gammaE = 0.002; %Reaction coefficient degradation epidermis
96         parameters.gammaD = 0.002; %Reaction coefficient degradation dermis
97         parameters.gammaB = 0.002; %Reaction coefficient degradation sub-cutaneous
98
99     end
100 end

```

B APPENDIX: UNIT TESTS

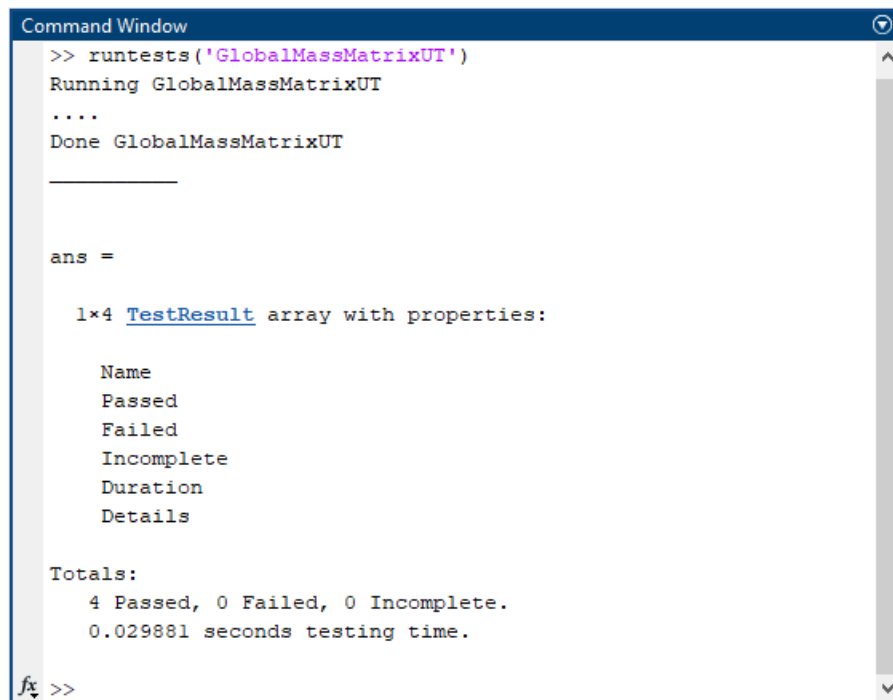
Listing 21. Global mass matrix unit tests

```

1 %% Test 1: test symmetry of the matrix
2 %% Test that this matrix is symmetric
3 tol = 1e-14;
4 Ne=4;
5 parameters.selection = '1';
6 parameters.D = 2;
7 parameters.lambda = 2;
8 parameters.f = 1;
9 GQ.switch = '0';
10 GQ.npts = 2;
11 order = 1;
12
13 msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);
14
15 %global matrix and transpose with values above
16 GM = GlobalMassMatrix(Ne,msh,GQ,order);
17 transposedGM = transpose(GM);
18
19 diff = GM - transposedGM;
20 diffnorm = sum(sum(diff.*diff));
21 assert(abs(diffnorm) <= tol)
22
23 %% Test 2: test that the Global Matrix is evaluated correctly
24 tol = 1e-14;
25 Ne=4;
26 parameters.selection = '1';
27 parameters.D = 1;
28 parameters.lambda = 1;
29 parameters.f = 1;
30 GQ.switch = '0';
31 GQ.npts = 2;
32 order = 1;
33
34 msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);
35
36 %global matrix with values above

```

```
37 GM = GlobalMassMatrix(Ne,msh,GQ,order);
38
39 %analytical global matrix
40 analyticalGM =
    [1/12,1/24,0,0,0;1/24,1/6,1/24,0,0;0,1/24,1/6,1/24,0;0,0,1/24,1/6,1/24;0,0,0,1/24,1/12];
41
42 diff = GM - analyticalGM;
43 diffnorm = sum(sum(diff.*diff));
44 assert(abs(diffnorm) <= tol)
45
46 %% Test 3: test Global Matrix Size
47 % Test that the size is equal to that of the nodes in the mesh
48 Ne=4;
49 parameters.selection = '1';
50 parameters.D = 5;
51 parameters.lambda = 5;
52 parameters.f = 1;
53 GQ.switch = '0';
54 GQ.npts = 2;
55 order = 1;
56
57 msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);
58
59 %global matrix with values above and size
60 GM = GlobalMassMatrix(Ne,msh,GQ,order);
61 sizeGM = size(GM);
62
63 assert(sizeGM(1)==(msh.ngn)); %columns are same as node number
64 assert(sizeGM(2)==(msh.ngn)); %rows are same as node number
65
66 %% Test 4: test that values are only in the diagonals
67 % Tests that all values are only in the diagonals, and the rest are zeros.
68 tol = 1e-14;
69 Ne=4;
70 parameters.selection = '1';
71 parameters.D = 1;
72 parameters.lambda = 1;
73 parameters.f = 1;
74 GQ.switch = '0';
75 GQ.npts = 2;
76 order = 1;
77
78 msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);
79
80 %global matrix with values above
81 GM = GlobalMassMatrix(Ne,msh,GQ,order);
82 %zeros same size as GM
83 zeroMatrix = zeros(msh.ngn);
84
85 mainGMDiagonal = diag(diag(GM)); %main diagonal
86 upperGMDiagonal = diag(diag(GM, 1), 1); %upper diagonal
87 lowerGMDiagonal = diag(diag(GM, -1), -1); %lower diagonal
88
89 %remove above diagonals from GM
90 emptyGM = GM - mainGMDiagonal - lowerGMDiagonal - upperGMDiagonal;
91
92 diff = emptyGM - zeroMatrix;
93 diffnorm = sum(sum(diff.*diff));
94 assert(abs(diffnorm) <= tol)
```

```

Command Window
>> runtests('GlobalMassMatrixUT')
Running GlobalMassMatrixUT
....
Done GlobalMassMatrixUT

-----

ans =

1x4 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
    4 Passed, 0 Failed, 0 Incomplete.
    0.029881 seconds testing time.

fx >>

```

Figure 16. Global mass matrix unit tests results.

Listing 22. Global stiffness matrix unit tests

```

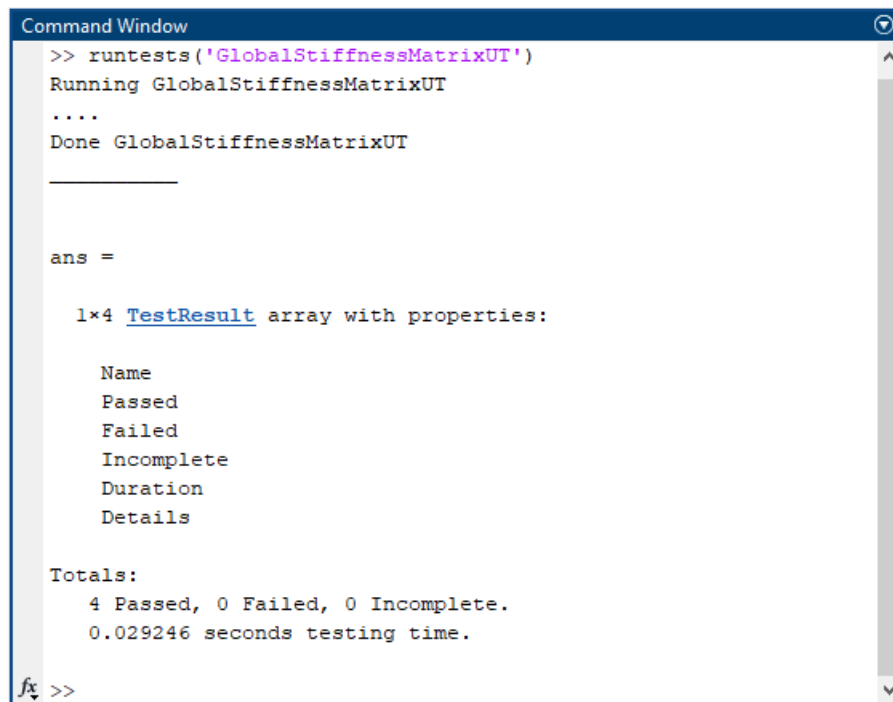
1 %% Test 1: test symmetry of the matrix
2 % % Test that this matrix is symmetric
3 tol = 1e-14;
4 Ne=4;
5 parameters.selection = '1';
6 parameters.D = 1;
7 parameters.lambda = 1;
8 parameters.f = 0;
9 GQ.switch = '0';
10 GQ.npts = 2;
11 order = 1;
12
13 msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);
14
15 %global matrix and transpose with values above
16 GM = GlobalStiffnessMatrix(Ne,msh,GQ,order);
17 transposedGM = transpose(GM);
18
19 diff = GM - transposedGM;
20 diffnorm = sum(sum(diff.*diff));
21 assert(abs(diffnorm) <= tol)
22
23 %% Test 2: test that the Global Matrix is evaluated correctly
24 tol = 1e-14;
25 Ne=4;
26 parameters.selection = '1';
27 parameters.D = 5;
28 parameters.lambda = 0;
29 parameters.f = 0;
30 GQ.switch = '0';
31 GQ.npts = 2;
32 order = 1;
33
34 msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);
35
36 %global matrix with values above

```

```

37 GM = GlobalStiffnessMatrix(Ne,msh,GQ,order);
38
39 %analytical global matrix
40 analyticalGM = [20,-20,0,0,0;-20,40,-20,0,0;0,-20,40,-20,0;0,0,-20,40,-20;0,0,0,-20,20];
41
42 diff = GM - analyticalGM;
43 diffnorm = sum(sum(diff.*diff));
44 assert(abs(diffnorm) <= tol)
45
46 %% Test 3: test Global Matrix Size
47 % Test that the size is equal to that of the nodes in the mesh
48 Ne=4;
49 parameters.selection = '1';
50 parameters.D = 5;
51 parameters.lambda = 5;
52 parameters.f = 0;
53 GQ.switch = '0';
54 GQ.npts = 2;
55 order = 1;
56
57 msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);
58
59 %global matrix with values above and size
60 GM = GlobalStiffnessMatrix(Ne,msh,GQ,order);
61 sizeGM = size(GM);
62
63 assert(sizeGM(1)==(msh.ngn)); %columns are same as node number
64 assert(sizeGM(2)==(msh.ngn)); %rows are same as node number
65
66 %% Test 4: test that values are only in the diagonals
67 % Tests that all values are only in the diagonals, and the rest are zeros.
68 tol = 1e-14;
69 Ne=4;
70 parameters.selection = '1';
71 parameters.D = 1;
72 parameters.lambda = 1;
73 parameters.f = 0;
74 GQ.switch = '0';
75 GQ.npts = 2;
76 order = 1;
77
78 msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);
79
80 %global matrix with values above
81 GM = GlobalStiffnessMatrix(Ne,msh,GQ,order);
82 %zeros same size as GM
83 zeroMatrix = zeros(msh.ngn);
84
85 mainGMDiagonal = diag(diag(GM)); %main diagonal
86 upperGMDiagonal = diag(diag(GM, 1), 1); %upper diagonal
87 lowerGMDiagonal = diag(diag(GM, -1), -1); %lower diagonal
88
89 %remove above diagonals from GM
90 emptyGM = GM - mainGMDiagonal - lowerGMDiagonal - upperGMDiagonal;
91
92 diff = emptyGM - zeroMatrix;
93 diffnorm = sum(sum(diff.*diff));
94 assert(abs(diffnorm) <= tol)

```



```

Command Window
>> runtests('GlobalStiffnessMatrixUT')
Running GlobalStiffnessMatrixUT
....
Done GlobalStiffnessMatrixUT

-----

ans =

1x4 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
    4 Passed, 0 Failed, 0 Incomplete.
    0.029246 seconds testing time.

fx >>

```

Figure 17. Global stiffness matrix unit tests results.

Listing 23. Global source vector unit tests

```

1 %% Test 1: test symmetry of the vector
2 % % Test that this vector is symmetric
3 tol = 1e-14;
4 Ne=4;
5 parameters.selection = '1';
6 parameters.D = 0;
7 parameters.lambda = 0;
8 parameters.f = 1;
9 GQ.switch = '0';
10 GQ.npts = 2;
11 order = 1;
12
13 msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);
14
15 %global vector and flipped with values above
16 GV = GlobalSourceVector(Ne,msh,GQ,order);
17 flippedGV = flip(GV);
18
19 diff = GV - flippedGV;
20 diffnorm = sum(sum(diff.*diff));
21 assert(abs(diffnorm) <= tol)
22
23 %% Test 2: test that the Global Vector is evaluated correctly.
24 tol = 1e-14;
25 Ne=4;
26 parameters.selection = '1';
27 parameters.D = 0;
28 parameters.lambda = 0;
29 parameters.f = 5;
30 GQ.switch = '0';
31 GQ.npts = 2;
32 order = 1;
33
34 msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);
35
36 %global vector with values above

```

```

37 GV = GlobalSourceVector(Ne,msh,GQ,order);
38
39 %analytical global vector
40 analyticalGV = [0.625; 1.25; 1.25; 1.25; 0.625];
41
42 diff = GV - analyticalGV;
43 diffnorm = sum(sum(diff.*diff));
44 assert(abs(diffnorm) <= tol)
45
46 %% Test 3: test Global Vector Size
47 % Test that the size is equal to that of the nodes in the mesh
48 Ne=4;
49 parameters.selection = '1';
50 parameters.D = 0;
51 parameters.lambda = 0;
52 parameters.f = 5;
53 GQ.switch = '0';
54 GQ.npts = 2;
55 order = 1;
56
57 msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);
58
59 %global vector with values above and size
60 GV = GlobalSourceVector(Ne,msh,GQ,order);
61 GVsize = size(GV);
62
63 assert(GVsize(1)==(msh.ngn)); %columns are same as node number
64 assert(GVsize(2)==1); %rows are 1

```

The screenshot shows a MATLAB Command Window titled "Command Window". The user has entered the command `>> runtests('GlobalSourceVectorUT')`. The output shows the test suite "GlobalSourceVectorUT" is running and has completed successfully. Below this, a table of results is displayed for 3 tests. All tests passed. The total testing time is 0.024703 seconds.

```

>> runtests('GlobalSourceVectorUT')
Running GlobalSourceVectorUT
...
Done GlobalSourceVectorUT

-----

ans =

1x3 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
    3 Passed, 0 Failed, 0 Incomplete.
    0.024703 seconds testing time.
fx >>

```

Figure 18. Global source vector unit tests results.

Listing 24. Local mass element unit tests

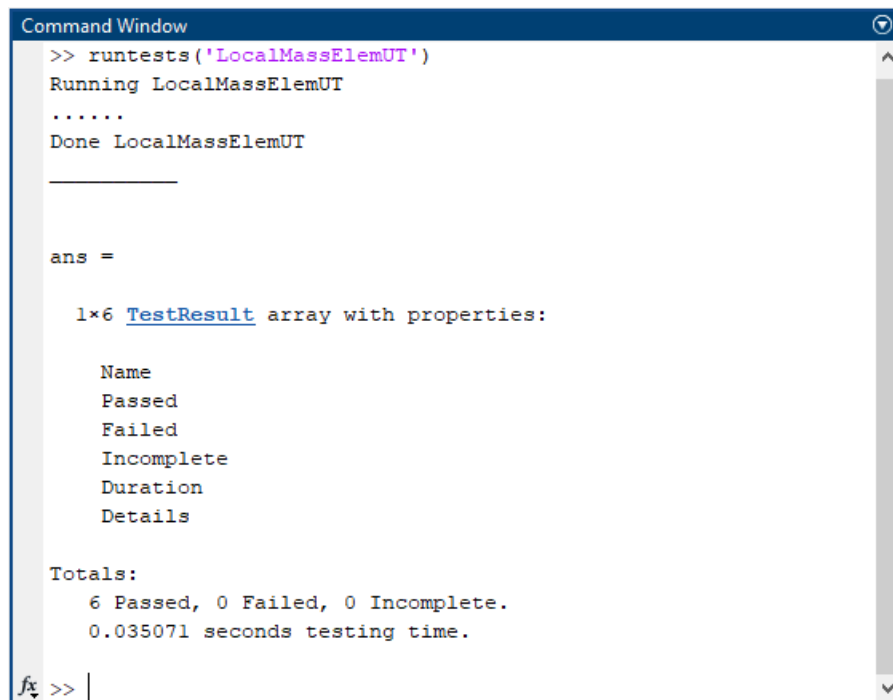
```

1 %% Test 1: test symmetry of the matrix
2 % % Test that this matrix is symmetric using manual integration
3 tol = 1e-14;
4 eN=1;
5 parameters.selection = '1';

```

```
6 parameters.D = 2;
7 parameters.lambda = 2;
8 parameters.f = 1;
9 GQ.switch = '0';
10 GQ.npts = 2;
11 order = 1;
12
13 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
14
15 elemat = LocalMassElem(eN,msh,GQ,order);
16
17 assert(abs(elemat(1,2) - elemat(2,1)) <= tol)
18
19 %% Test 2: test 2 different elements of the same size produce same matrix
20 % % Test that for two elements of an equispaced mesh, the element matrices
21 % % calculated are the same using manual integration
22 tol = 1e-14;
23 eN=1;
24 parameters.selection = '1';
25 parameters.D = 5;
26 parameters.lambda = 5;
27 parameters.f = 1;
28 GQ.switch = '0';
29 GQ.npts = 2;
30 order = 1;
31
32 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
33
34 elemat1 = LocalMassElem(eN,msh,GQ,order);
35
36 eN=2;
37
38 elemat2 = LocalMassElem(eN,msh,GQ,order);
39
40 diff = elemat1 - elemat2;
41 diffnorm = sum(sum(diff.*diff));
42 assert(abs(diffnorm) <= tol)
43
44 %% Test 3: test that one matrix is evaluated correctly
45 % % Test that the element matrix is evaluated correctly using manual integration
46 tol = 1e-14;
47 eN=1;
48 parameters.selection = '1';
49 parameters.D = 5;
50 parameters.lambda = 5;
51 parameters.f = 1;
52 GQ.switch = '0';
53 GQ.npts = 2;
54 order = 1;
55
56 msh = OneDimLinearMeshGen(0,1,2,order,parameters);
57
58 elemat1 = LocalMassElem(eN,msh,GQ,order);
59
60 elemat2 = [ 1/6 1/12; 1/12 1/6];
61 diff = elemat1 - elemat2; %calculate the difference between the two matrices
62 diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
63 assert(abs(diffnorm) <= tol)
64
65 %% Test 4: test symmetry of the matrix
66 % % Test that this matrix is symmetric using Gaussian Quadrature
67 tol = 1e-14;
68 eN=1;
69 parameters.selection = '1';
70 parameters.D = 2;
71 parameters.lambda = 2;
72 parameters.f = 1;
73 GQ.switch = '1';
74 GQ.npts = 2;
```

```
75 order = 1;
76
77 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
78
79 elemat = LocalMassElem(eN,msh,GQ,order);
80
81 assert(abs(elemat(1,2) - elemat(2,1)) <= tol)
82
83 %% Test 5: test 2 different elements of the same size produce same matrix
84 % % Test that for two elements of an equispaced mesh, the element matrices
85 % % calculated are the same using Gaussian Quadrature
86 tol = 1e-14;
87 eN=1;
88 parameters.selection = '1';
89 parameters.D = 5;
90 parameters.lambda = 5;
91 parameters.f = 1;
92 GQ.switch = '1';
93 GQ.npts = 2;
94 order = 1;
95
96 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
97
98 elemat1 = LocalMassElem(eN,msh,GQ,order);
99
100 eN=2;
101
102 elemat2 = LocalMassElem(eN,msh,GQ,order);
103
104 diff = elemat1 - elemat2;
105 diffnorm = sum(sum(diff.*diff));
106 assert(abs(diffnorm) <= tol)
107
108 %% Test 6: test that one matrix is evaluated correctly
109 % % Test that the element matrix is evaluated correctly using Gaussian Quadrature
110 tol = 1e-14;
111 eN=1;
112 parameters.selection = '1';
113 parameters.D = 5;
114 parameters.lambda = 5;
115 parameters.f = 1;
116 GQ.switch = '1';
117 GQ.npts = 2;
118 order = 1;
119
120 msh = OneDimLinearMeshGen(0,1,2,order,parameters);
121
122 elemat1 = LocalMassElem(eN,msh,GQ,order);
123
124 elemat2 = [ 1/6 1/12; 1/12 1/6];
125 diff = elemat1 - elemat2; %calculate the difference between the two matrices
126 diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
127 assert(abs(diffnorm) <= tol)
```



```

Command Window
>> runtests('LocalMassElemUT')
Running LocalMassElemUT
.....
Done LocalMassElemUT

-----

ans =

1x6 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
    6 Passed, 0 Failed, 0 Incomplete.
    0.035071 seconds testing time.

fx >>

```

Figure 19. Local mass element unit tests results.

Listing 25. Diffusion local element matrix unit tests

```

1 %% Test 1: test symmetry of the matrix
2 %% Test that this matrix is symmetric using manual integration
3 tol = 1e-14;
4 eN=1;
5 parameters.selection = '1';
6 parameters.D = 2;
7 parameters.lambda = 0;
8 parameters.f = 0;
9 GQ.switch = '0';
10 GQ.npts = 2;
11 order = 1;
12
13 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
14
15 elemat = LEMdiffusion(eN,msh,GQ,order);
16
17 assert(abs(elemat(1,2) - elemat(2,1)) <= tol)
18
19 %% Test 2: test 2 different elements of the same size produce same matrix
20 %% Test that for two elements of an equispaced mesh, the element matrices
21 %% calculated are the same using manual integration
22 tol = 1e-14;
23 eN=1;
24 parameters.selection = '1';
25 parameters.D = 5;
26 parameters.lambda = 0;
27 parameters.f = 0;
28 GQ.switch = '0';
29 GQ.npts = 2;
30 order = 1;
31 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
32
33 elemat1 = LEMdiffusion(eN,msh,GQ,order);
34
35 eN=2;
36

```

```
37 elemat2 = LEMdiffusion(eN,msh,GQ,order);
38
39 diff = elemat1 - elemat2;
40 diffnorm = sum(sum(diff.*diff));
41 assert(abs(diffnorm) <= tol)
42
43 %% Test 3: test that one matrix is evaluted correctly
44 % % Test that the element matrix is evaluated correctly using manual integration
45 tol = 1e-14;
46 eN=1;
47 parameters.selection = '1';
48 parameters.D = 2.5;
49 parameters.lambda = 0;
50 parameters.f = 0;
51 GQ.switch = '0';
52 GQ.npts = 2;
53 order = 1;
54 msh = OneDimLinearMeshGen(0,1,3,order,parameters);
55
56 elemat1 = LEMdiffusion(eN,msh,GQ,order);
57
58 elemat2 = [ 7.5 -7.5; -7.5 7.5];
59 diff = elemat1 - elemat2; %calculate the difference between the two matrices
60 diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
61 assert(abs(diffnorm) <= tol)
62
63 %% Test 4: test symmetry of the matrix
64 % % Test that this matrix is symmetric using Gaussian Quadrature
65 tol = 1e-14;
66 eN=1;
67 parameters.selection = '1';
68 parameters.D = 2;
69 parameters.lambda = 0;
70 parameters.f = 0;
71 GQ.switch = '1';
72 GQ.npts = 2;
73 order = 1;
74
75 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
76
77 elemat = LEMdiffusion(eN,msh,GQ,order);
78
79 assert(abs(elemat(1,2) - elemat(2,1)) <= tol)
80
81 %% Test 5: test 2 different elements of the same size produce same matrix
82 % % Test that for two elements of an equispaced mesh, the element matrices
83 % % calculated are the same using Gaussian Quadrature
84 tol = 1e-14;
85 eN=1;
86 parameters.selection = '1';
87 parameters.D = 5;
88 parameters.lambda = 0;
89 parameters.f = 0;
90 GQ.switch = '1';
91 GQ.npts = 2;
92 order = 1;
93 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
94
95 elemat1 = LEMdiffusion(eN,msh,GQ,order);
96
97 eN=2;
98
99 elemat2 = LEMdiffusion(eN,msh,GQ,order);
100
101 diff = elemat1 - elemat2;
102 diffnorm = sum(sum(diff.*diff));
103 assert(abs(diffnorm) <= tol)
104
105 %% Test 6: test that one matrix is evaluted correctly
```



```

106 % % Test that the element matrix is evaluated correctly using Gaussian
107 % % Quadrature
108 tol = 1e-14;
109 eN=1;
110 parameters.selection = '1';
111 parameters.D = 2.5;
112 parameters.lambda = 0;
113 parameters.f = 0;
114 GQ.switch = '1';
115 GQ.npts = 2;
116 order = 1;
117 msh = OneDimLinearMeshGen(0,1,3,order,parameters);
118
119 elemat1 = LEMdiffusion(eN,msh,GQ,order);
120
121 elemat2 = [ 7.5 -7.5; -7.5 7.5];
122 diff = elemat1 - elemat2; %calculate the difference between the two matrices
123 diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
124 assert(abs(diffnorm) <= tol)

```

```

Command Window
>> runtests('LEMdiffusionUT')
Running LEMdiffusionUT
.....
Done LEMdiffusionUT

_____

ans =

    1×6 TestResult array with properties:

        Name
        Passed
        Failed
        Incomplete
        Duration
        Details

Totals:
    6 Passed, 0 Failed, 0 Incomplete.
    0.036294 seconds testing time.

fx >>

```

Figure 20. Diffusion local element matrix unit tests results.

Listing 26. Reaction local element matrix unit tests

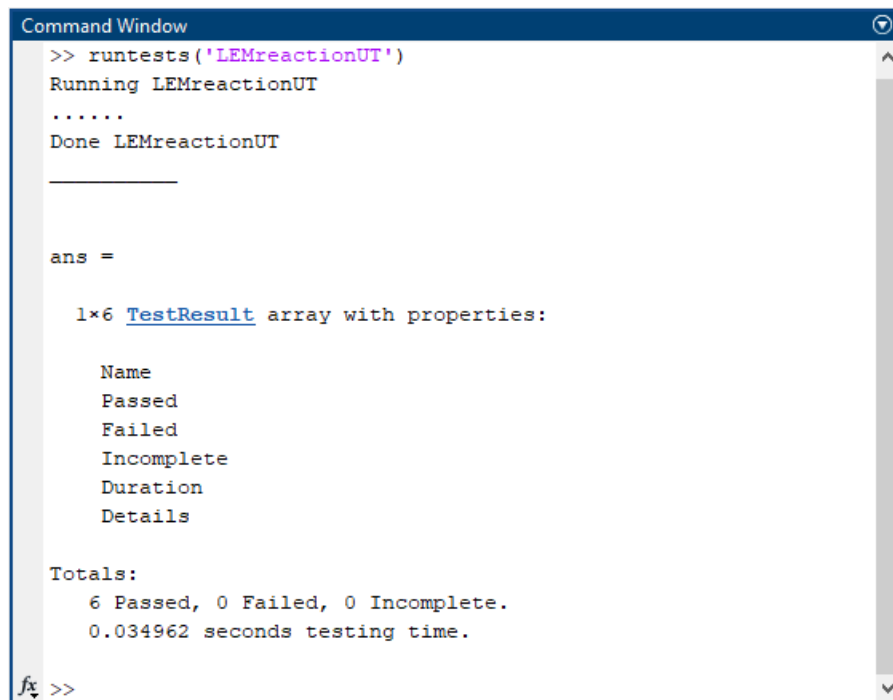
```

1 %% Test 1: test symmetry of the matrix
2 % % Test that this matrix is symmetric using manual integration
3 tol = 1e-14;
4 eN=1;
5 parameters.selection = '1';
6 parameters.D = 0;
7 parameters.lambda = 2;
8 parameters.f = 0;
9 GQ.switch = '0';
10 GQ.npts = 2;
11 order = 1;
12
13 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
14

```

```
15 elemat = LEMreaction(eN,msh,GQ,order);
16
17 assert(abs(elemat(1,2) - elemat(2,1)) <= tol)
18
19 %% Test 2: test 2 different elements of the same size produce same matrix
20 % % Test that for two elements of an equispaced mesh, the element matrices
21 % % calculated are the same using manual integration
22 tol = 1e-14;
23 eN=1;
24 parameters.selection = '1';
25 parameters.D = 0;
26 parameters.lambda = 5;
27 parameters.f = 0;
28 GQ.switch = '0';
29 GQ.npts = 2;
30 order = 1;
31
32 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
33
34 elemat1 = LEMreaction(eN,msh,GQ,order);
35
36 eN=2;
37
38 elemat2 = LEMreaction(eN,msh,GQ,order);
39
40 diff = elemat1 - elemat2;
41 diffnorm = sum(sum(diff.*diff));
42 assert(abs(diffnorm) <= tol)
43
44 %% Test 3: test that one matrix is evaluated correctly
45 % % Test that the element matrix is evaluated correctly using manual integration
46 tol = 1e-14;
47 eN=1;
48 parameters.selection = '1';
49 parameters.D = 0;
50 parameters.lambda = 5;
51 parameters.f = 0;
52 GQ.switch = '0';
53 GQ.npts = 2;
54 order = 1;
55
56 msh = OneDimLinearMeshGen(0,1,2,order,parameters);
57
58 elemat1 = LEMreaction(eN,msh,GQ,order);
59
60 elemat2 = [ 5/6 5/12; 5/12 5/6];
61 diff = elemat1 - elemat2; %calculate the difference between the two matrices
62 diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
63 assert(abs(diffnorm) <= tol)
64
65 %% Test 4: test symmetry of the matrix
66 % % Test that this matrix is symmetric using Gaussian Quadrature
67 tol = 1e-14;
68 eN=1;
69 parameters.selection = '1';
70 parameters.D = 0;
71 parameters.lambda = 2;
72 parameters.f = 0;
73 GQ.switch = '1';
74 GQ.npts = 2;
75 order = 1;
76
77 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
78
79 elemat = LEMreaction(eN,msh,GQ,order);
80
81 assert(abs(elemat(1,2) - elemat(2,1)) <= tol)
82
83 %% Test 5: test 2 different elements of the same size produce same matrix
```

```
84 %% Test that for two elements of an equispaced mesh, the element matrices
85 %% calculated are the same using Gaussian Quadrature
86 tol = 1e-14;
87 eN=1;
88 parameters.selection = '1';
89 parameters.D = 0;
90 parameters.lambda = 5;
91 parameters.f = 0;
92 GQ.switch = '1';
93 GQ.npts = 2;
94 order = 1;
95
96 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
97
98 elemat1 = LEMreaction(eN,msh,GQ,order);
99
100 eN=2;
101
102 elemat2 = LEMreaction(eN,msh,GQ,order);
103
104 diff = elemat1 - elemat2;
105 diffnorm = sum(sum(diff.*diff));
106 assert(abs(diffnorm) <= tol)
107
108 %% Test 6: test that one matrix is evaluated correctly
109 %% Test that the element matrix is evaluated correctly using Gaussian Quadrature
110 tol = 1e-14;
111 eN=1;
112 parameters.selection = '1';
113 parameters.D = 0;
114 parameters.lambda = 5;
115 parameters.f = 0;
116 GQ.switch = '1';
117 GQ.npts = 2;
118 order = 1;
119
120 msh = OneDimLinearMeshGen(0,1,2,order,parameters);
121
122 elemat1 = LEMreaction(eN,msh,GQ,order);
123
124 elemat2 = [ 5/6 5/12; 5/12 5/6];
125 diff = elemat1 - elemat2; %calculate the difference between the two matrices
126 diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
127 assert(abs(diffnorm) <= tol)
```



```

Command Window
>> runtests('LEMreactionUT')
Running LEMreactionUT
.....
Done LEMreactionUT

_____

ans =

1×6 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
    6 Passed, 0 Failed, 0 Incomplete.
    0.034962 seconds testing time.

fx >>

```

Figure 21. Reaction local element matrix unit tests results.

Listing 27. Local source element unit tests

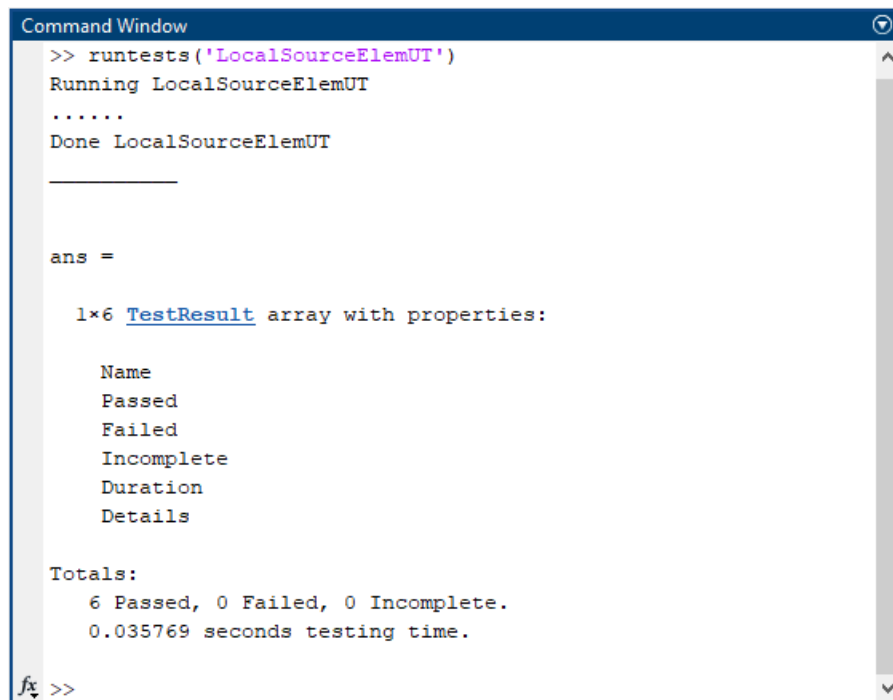
```

1 %% Test 1: test symmetry of the vector
2 %% Test that this vector is symmetric using manual integration
3 tol = 1e-14;
4 eN=1;
5 parameters.selection = '1';
6 parameters.D = 0;
7 parameters.lambda = 0;
8 parameters.f = 1;
9 GQ.switch = '0';
10 GQ.npts = 2;
11 order = 1;
12
13 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
14
15 elemat = LocalSourceElem(eN,msh,GQ,order);
16
17 assert(abs(elemat(1) - elemat(2)) <= tol)
18
19 %% Test 2: test 2 different elements of the same size produce same vector
20 %% Test that for two elements of an equispaced mesh, the element matrices
21 %% calculated are the same using manual integration
22 tol = 1e-14;
23 eN=1;
24 parameters.selection = '1';
25 parameters.D = 0;
26 parameters.lambda = 0;
27 parameters.f = 1;
28 GQ.switch = '0';
29 GQ.npts = 2;
30 order = 1;
31
32 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
33
34 elemat1 = LocalSourceElem(eN,msh,GQ,order);
35
36 eN=2;

```

```
37
38 elemat2 = LocalSourceElem(eN,msh,GQ,order);
39
40 diff = elemat1 - elemat2;
41 diffnorm = sum(sum(diff.*diff));
42 assert(abs(diffnorm) <= tol)
43
44 %% Test 3: test that one vector is evaluated correctly
45 % % Test that the element vector is evaluated correctly using manual integration
46 tol = 1e-14;
47 eN=1;
48 parameters.selection = '1';
49 parameters.D = 0;
50 parameters.lambda = 0;
51 parameters.f = 1;
52 GQ.switch = '0';
53 GQ.npts = 2;
54 order = 1;
55
56 msh = OneDimLinearMeshGen(0,1,4,order,parameters);
57
58 elemat1 = LocalSourceElem(eN,msh,GQ,order);
59
60 elemat2 = [ 0.1250; 0.1250 ];
61 diff = elemat1 - elemat2; %calculate the difference between the two matrices
62 diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
63 assert(abs(diffnorm) <= tol)
64
65 %% Test 4: test symmetry of the vector
66 % % Test that this vector is symmetric using Gaussian Quadrature
67 tol = 1e-14;
68 eN=1;
69 parameters.selection = '1';
70 parameters.D = 0;
71 parameters.lambda = 0;
72 parameters.f = 1;
73 GQ.switch = '1';
74 GQ.npts = 2;
75 order = 1;
76
77 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
78
79 elemat = LocalSourceElem(eN,msh,GQ,order);
80
81 assert(abs(elemat(1) - elemat(2)) <= tol)
82
83 %% Test 5: test 2 different elements of the same size produce same vector
84 % % Test that for two elements of an equispaced mesh, the element matrices
85 % % calculated are the same using Gaussian Quadrature
86 tol = 1e-14;
87 eN=1;
88 parameters.selection = '1';
89 parameters.D = 0;
90 parameters.lambda = 0;
91 parameters.f = 1;
92 GQ.switch = '1';
93 GQ.npts = 2;
94 order = 1;
95
96 msh = OneDimLinearMeshGen(0,1,10,order,parameters);
97
98 elemat1 = LocalSourceElem(eN,msh,GQ,order);
99
100 eN=2;
101
102 elemat2 = LocalSourceElem(eN,msh,GQ,order);
103
104 diff = elemat1 - elemat2;
105 diffnorm = sum(sum(diff.*diff));
```

```
106 assert(abs(diffnorm) <= tol)
107
108 %% Test 6: test that one vector is evaluated correctly
109 % % Test that the element vector is evaluated correctly using Gaussian Quadrature
110 tol = 1e-14;
111 eN=1;
112 parameters.selection = '1';
113 parameters.D = 0;
114 parameters.lambda = 0;
115 parameters.f = 1;
116 GQ.switch = '1';
117 GQ.npts = 2;
118 order = 1;
119
120 msh = OneDimLinearMeshGen(0,1,4,order,parameters);
121
122 elemat1 = LocalSourceElem(eN,msh,GQ,order);
123
124 elemat2 = [ 0.1250; 0.1250 ];
125 diff = elemat1 - elemat2; %calculate the difference between the two matrices
126 diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
127 assert(abs(diffnorm) <= tol)
```



```
Command Window
>> runtests('LocalSourceElemUT')
Running LocalSourceElemUT
.....
Done LocalSourceElemUT

-----

ans =

    1x6 TestResult array with properties:

        Name
        Passed
        Failed
        Incomplete
        Duration
        Details

Totals:
    6 Passed, 0 Failed, 0 Incomplete.
    0.035769 seconds testing time.

fx >>
```

Figure 22. Local source element unit tests results.

REFERENCES

- Abramowitz, M. and Stegun, I.A., 1964. *Handbook of mathematical functions with formulas, graphs and mathematical tables*. Washington, D.C.: National Bureau of Standards, p.887. Applied mathematics series ; 55.
- Cookson, A., 2023. *Assignment: Transient MATLAB-Based FEM Modelling*. University of Bath. Unpublished.
- Dean, J., 2023. *Introduction to the Finite Element Method (FEM)* [Online]. University of Cambridge. Unpublished. Available from: https://www.ccg.msm.cam.ac.uk/system/files/documents/FEMOR_Lecture_1.pdf [Accessed 2023-12-01].
- Souza Telles, D. de, 2010. Human skin diagram [Online]. [Online; accessed April 27, 2013]. Available from: <https://commons.wikimedia.org/wiki/File:HumanSkinDiagram.jpg> [Accessed 2023-12-01].
- Varga-Medveczky, Z., Kocsis, D., Naszlady, M.B., Fónagy, K. and Erdő, F., 2021. Skin-on-a-chip technology for testing transdermal drug delivery—starting points and recent developments. *Pharmaceutics* [Online], 13(11), p.1852. Available from: <https://doi.org/10.3390/pharmaceutics13111852>.