Pages 1–55
2023

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

LIST OF FIGURES

LISTINGS

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 suck 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.

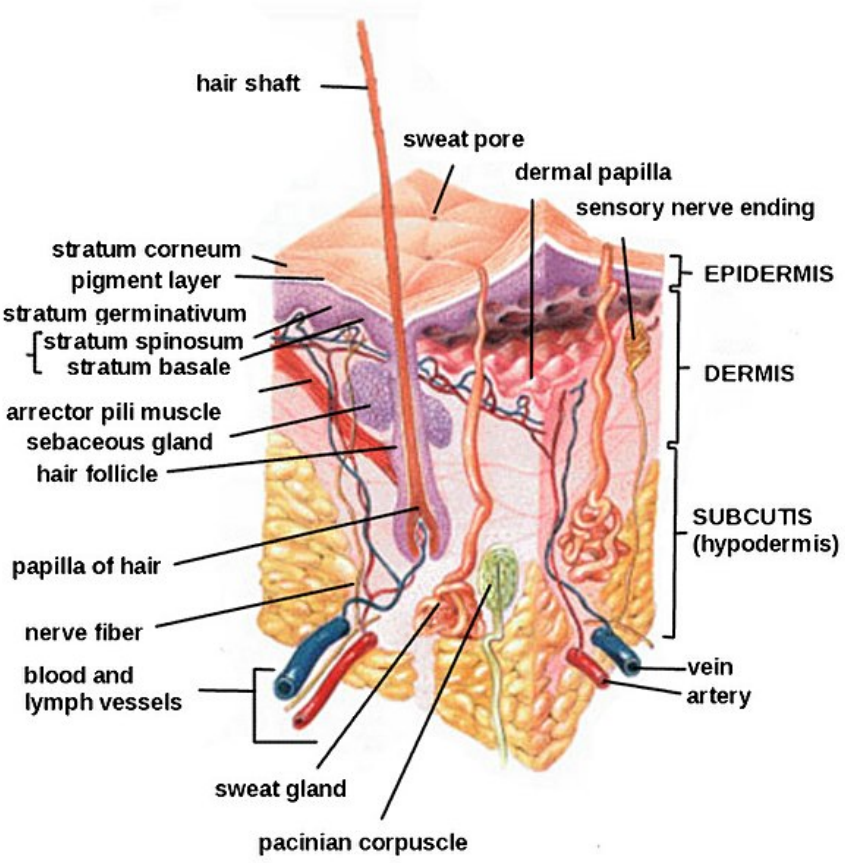

Figure 1. Diagram of human tissue (de Souza Telles, 2010). Note the discrete layers of tissue type.

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.

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

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:

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

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

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

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:

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

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:

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

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:

$$\int_{x_0}^{x_1} \left(v\frac{\partial c}{\partial t} + D\frac{\partial v}{\partial x}\frac{\partial c}{\partial x} - \lambda cv \right) \mathrm{d}x = \int_{x_0}^{x_1} vf\,\mathrm{d}x + \left[vD\frac{\partial c}{\partial x} \right]_{x_0}^{x_1} \tag{6}$$

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:

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

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.

$$M_{elem} = \int_{-1}^{1} \psi_n\psi_m J\mathrm{d}\xi \tag{8}$$

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

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:

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

where NBc is the Neumann Boundary conditions and $[M + \theta\Delta tK]$ 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.

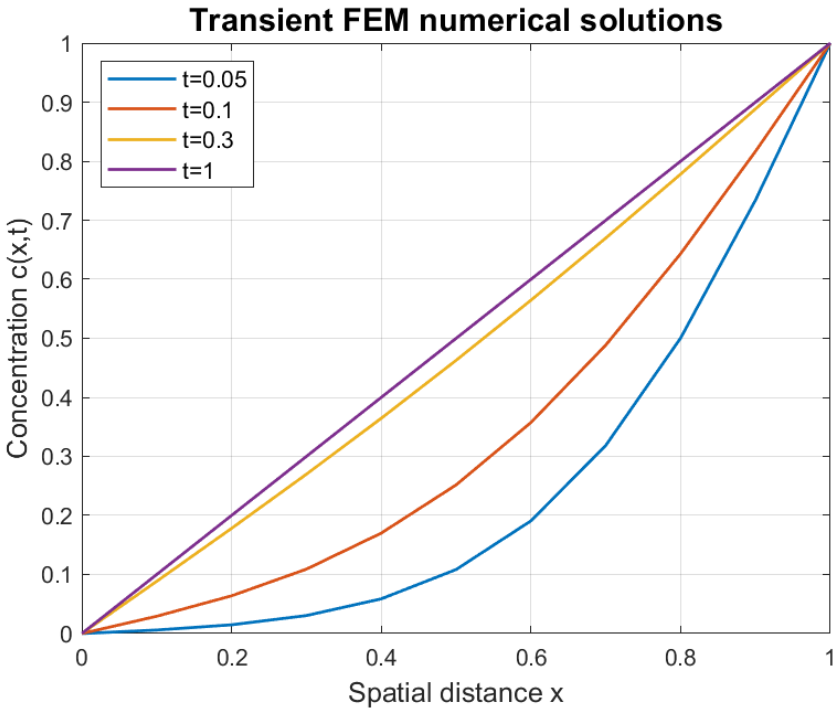

Figure 2. Numerical solution at different time stamps (dt=0.01, h=0.1).

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).

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

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 withe 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:

$$\frac{\partial c}{\partial t} = \frac{\partial^2 c}{\partial x^2} \tag{12}$$

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

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

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.

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

Figure 2 initially plotted only the numerical solution of the simplified transient diffusion equation in a ten element mesh ($Ne = 10$) 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	$0 < x < 1$	metres
Mesh elements	$Ne = 10$	
Time domain	$0 < t < 1$	seconds
Time step	$dt = 0.01$	seconds
Gauss Points	$npts = 2$	

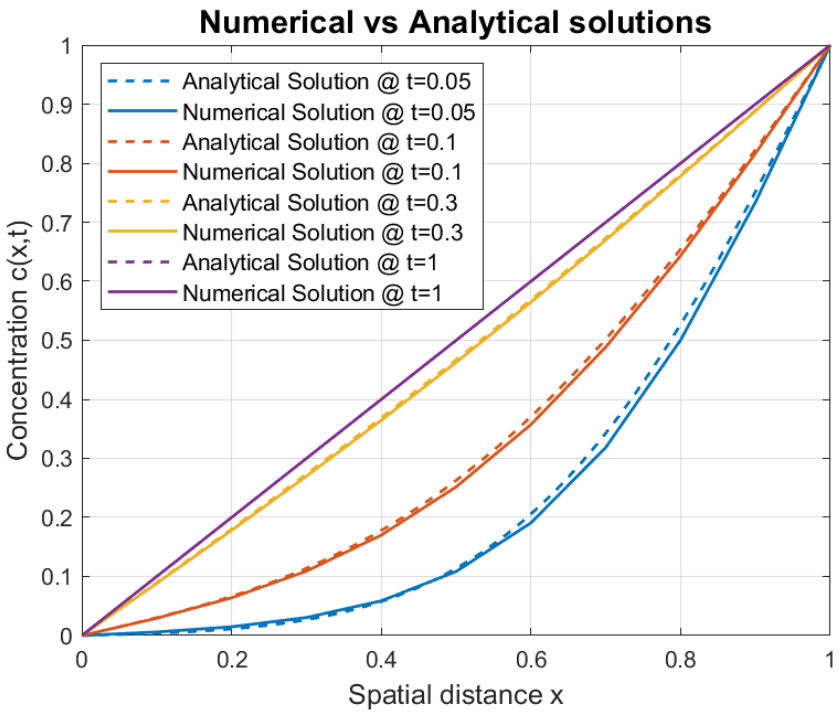

Figure 3. Analytical solution comparison against space (dt=0.01, h=0.1).

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).

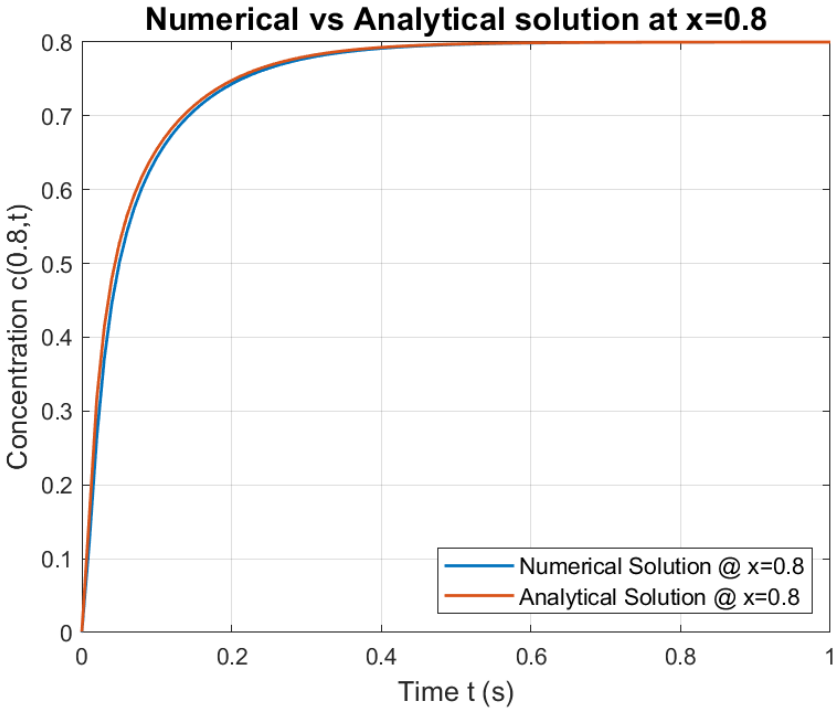

Figure 4. Analytical solution comparison against time (dt=0.01, h=0.1).

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 where 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.

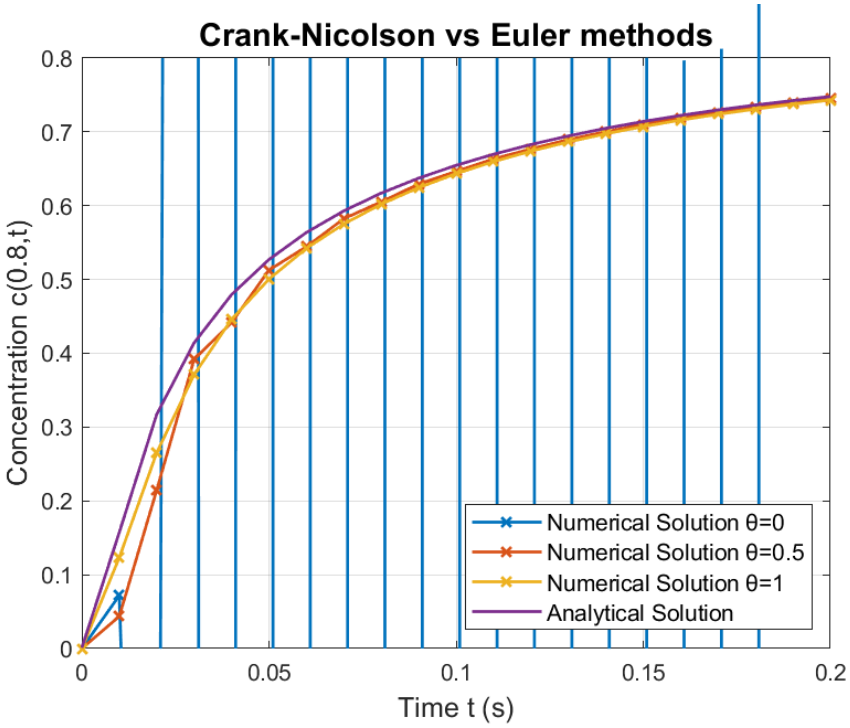

Figure 5. Theta methods comparison at x=0.8.

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 where plotted at $t = 0.05$ against the analytical solution in Figure 6

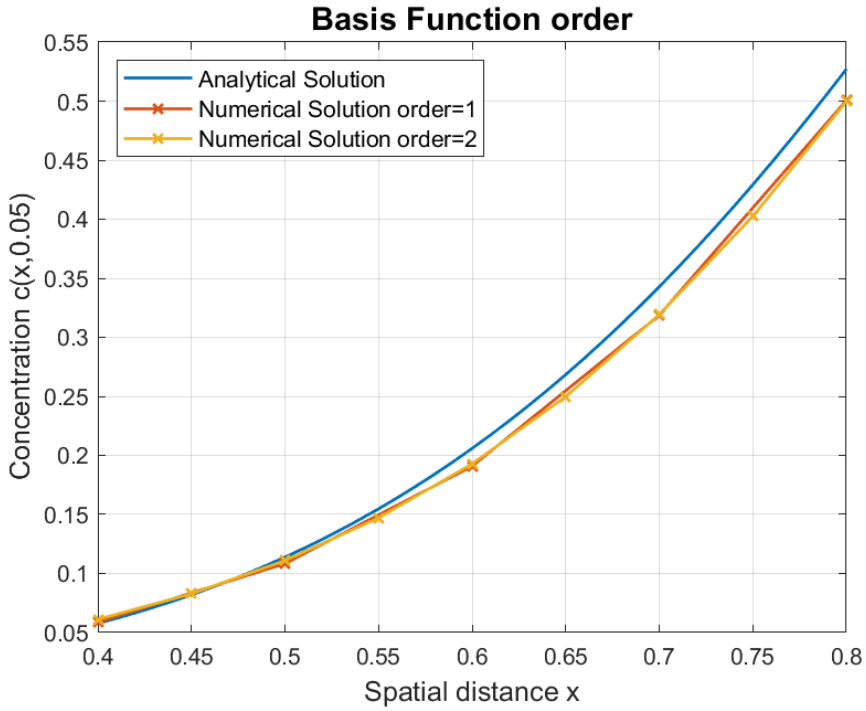

Figure 6. Basis function order comparison at t=0.05.

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 L2-norm. This is particularly useful for comparing a numerical solution to an analytical solution testing the convergence rate of the Finite Element Method. The L2-norn 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.

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

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 logarithm is taken on each side resulting in a straight line of gradient 2 (Figure 7).

$$\ln(E(x)) = \ln G + 2\ln h \tag{15}$$

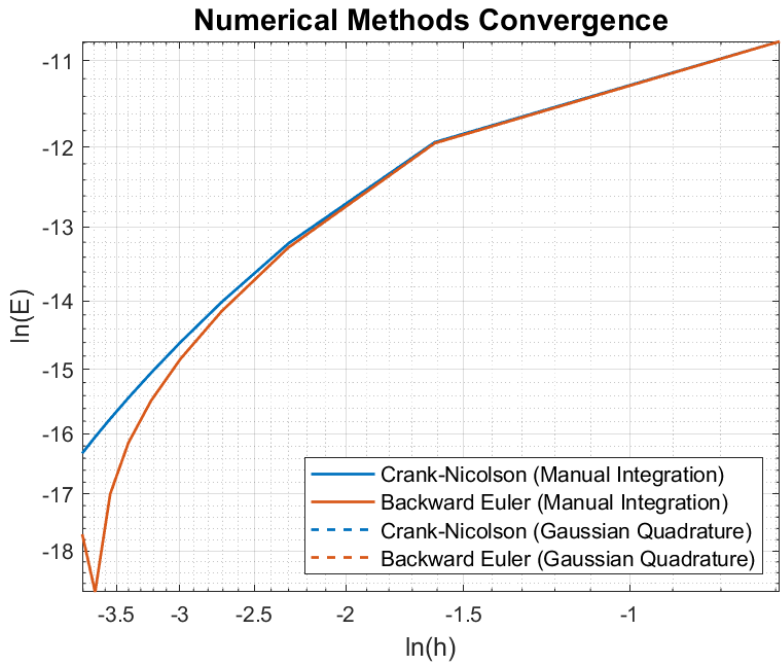

Figure 7. L2-norm error convergence with linear mesh.

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 $ln(h) < 0.05$ and the gradient of 1.91 demonstrate the validity of the plot compared to the predicted gradient of 2.

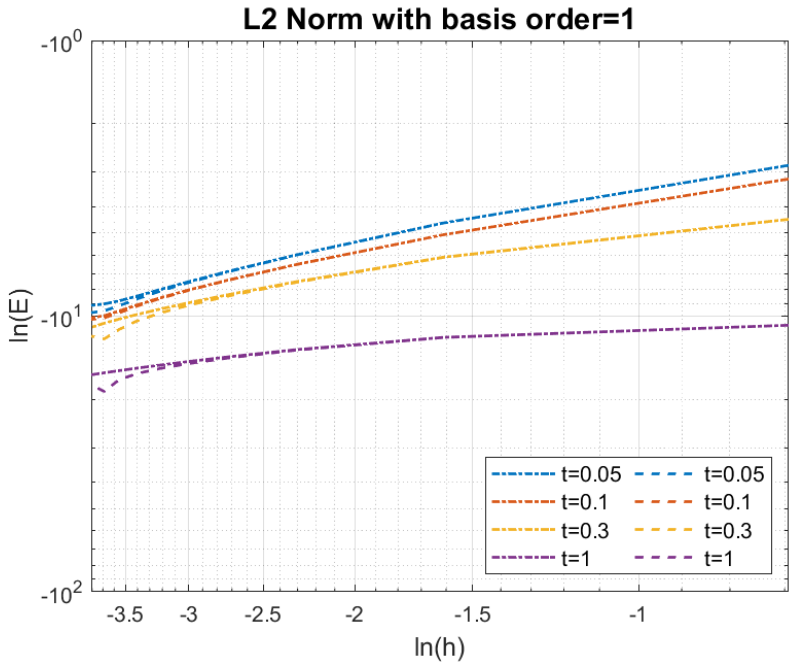

Figure 8. L2-norm error convergence between Crank-Nicolson (dash-dot) and Backward Euler (dash) for a linear mesh.

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.
From the theory the gradient for a quadratic basis function should be 3 and the following plot has a gradient of 3.22 demonstrating once again the accuracy of the plot.

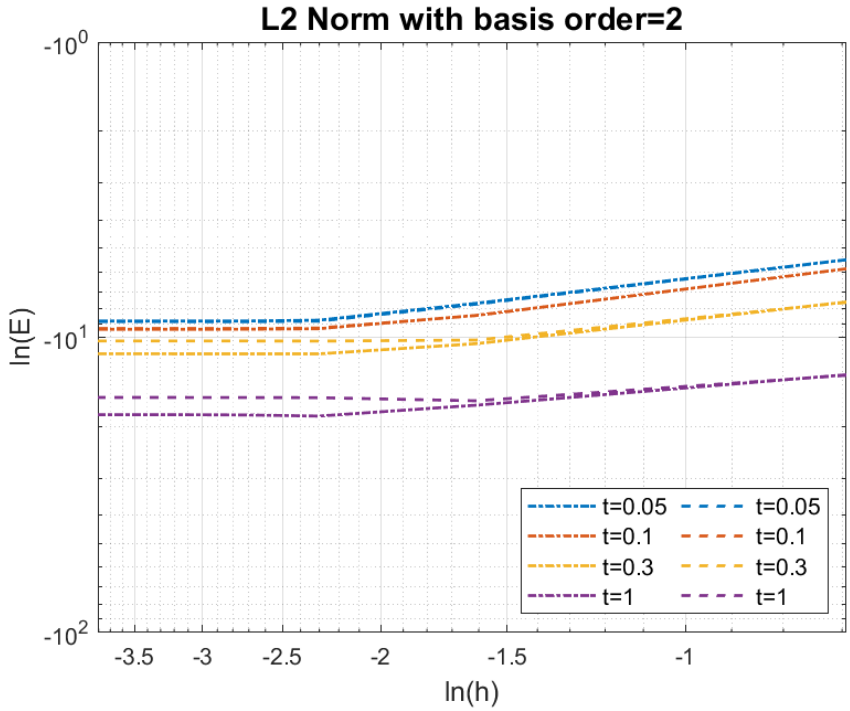

Figure 9. L2-norm error convergence between Crank-Nicolson (dash-dot) and Backward Euler (dash) for a quadratic mesh.

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:

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

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.

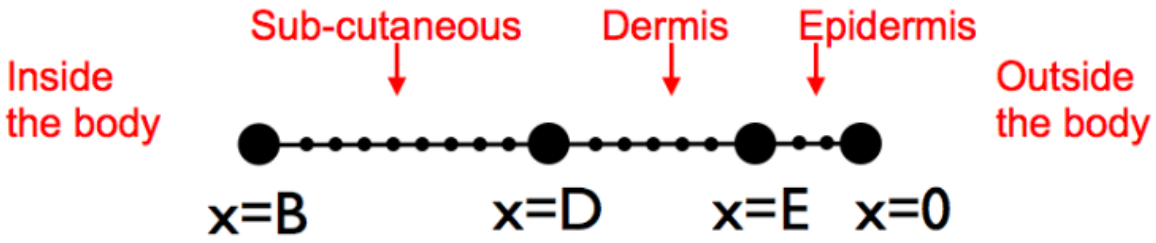

Figure 10. Schematic of finite element mesh required to represent skin tissue layers (Cookson, 2023).

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:

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

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

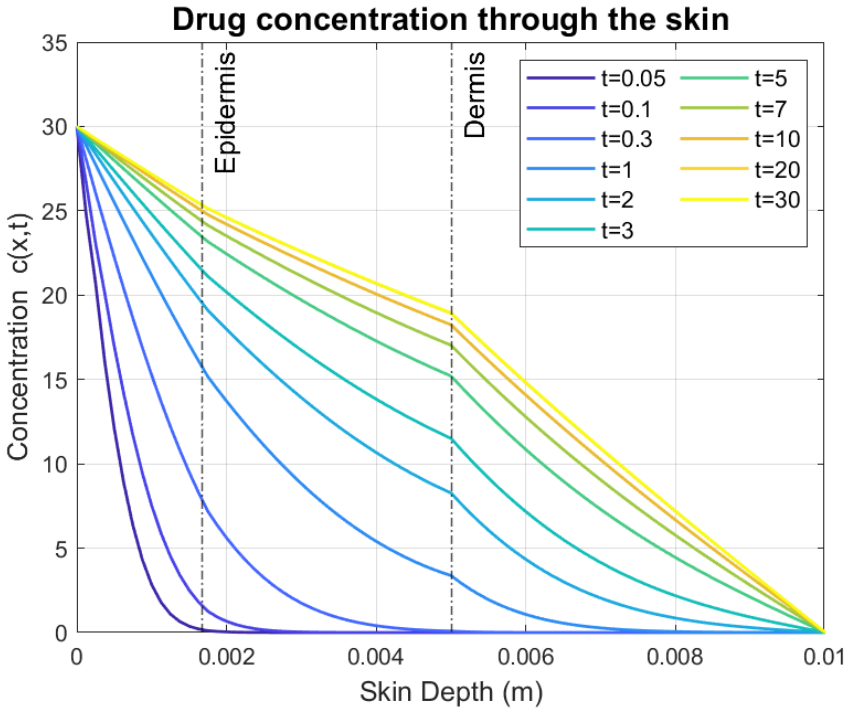

Figure 11. Concentration of a drug applied on the skin.

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.

$$K = \int_{t_{eff}}^{t=30} c\mathrm{d}t \tag{17}$$

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.

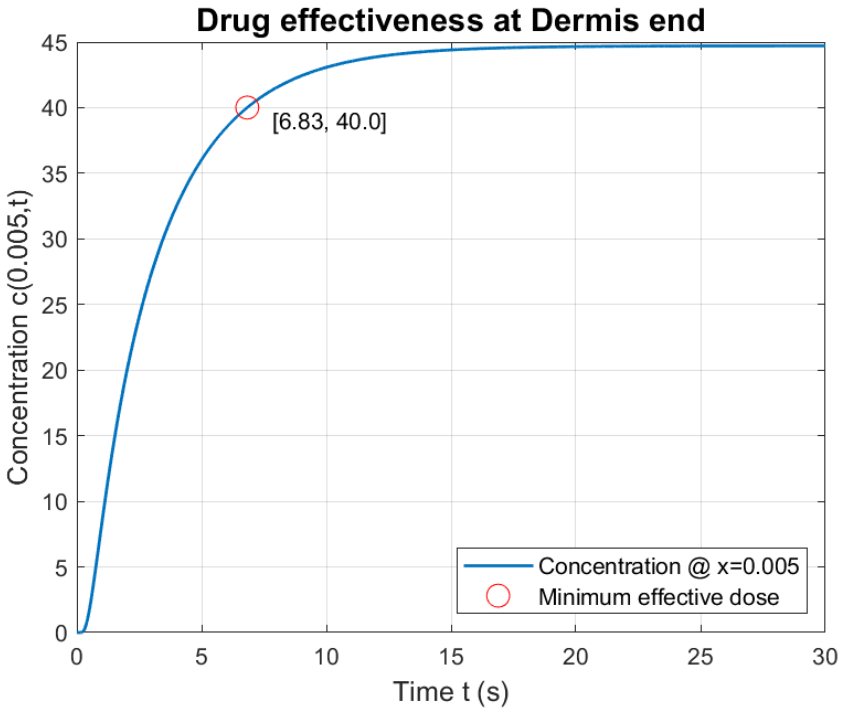

Figure 12. Minimum dose of 71 becoming effective after just under 7 seconds.

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 $x = 0.005$.

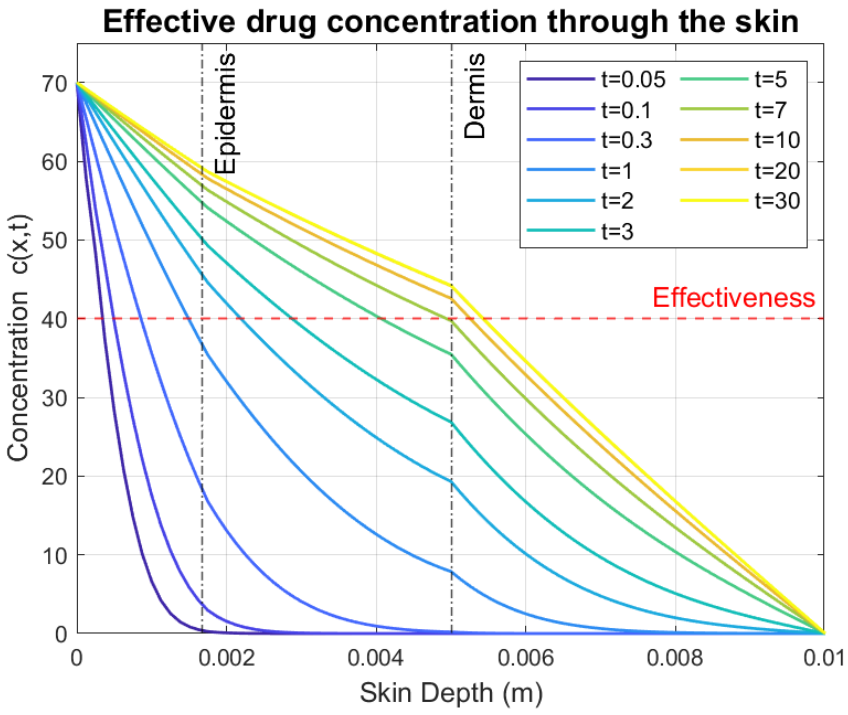

Figure 13. Concentration of the minimum effective drug applied to the skin.

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 constant 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.

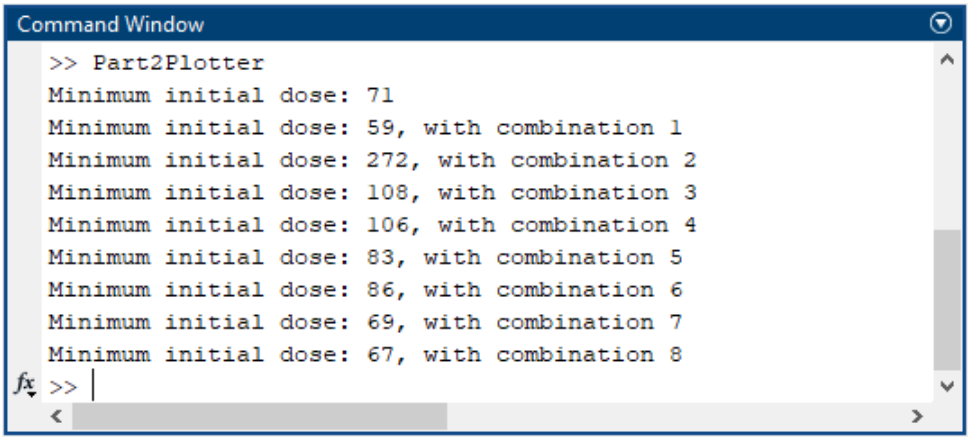

Figure 14. Minimum dose on the skin surface to reach effectiveness for different parameter combinations (Table 6).

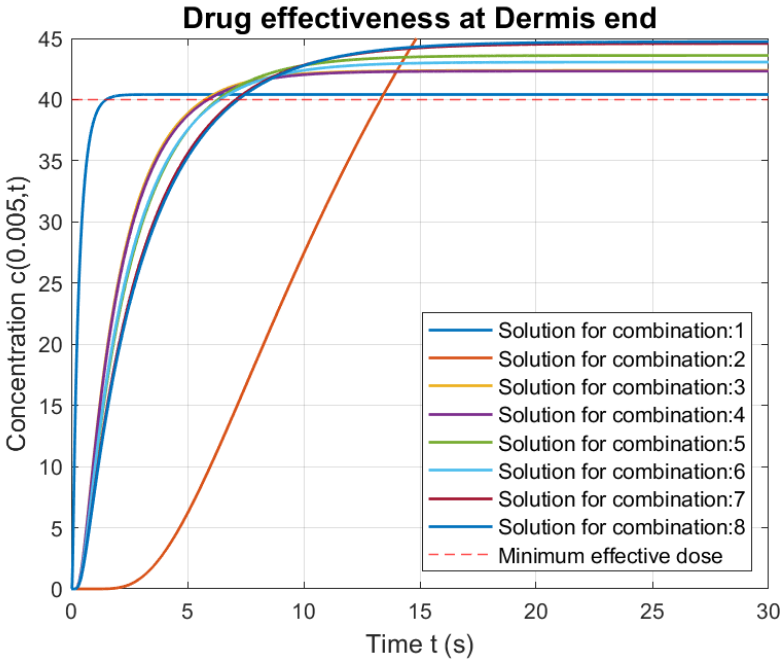

Figure 15. Concentration at $x = 0.005$ for different parameter combinations (Table 6).

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

```
function [c,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters)
%Solves the full transient form of the diffusion reaction equation.
% [M + \theta\DeltatK]cn+1 = [M  (1 - \theta)\Delta tK]cn + \Delta t\theta[Fn+1 + NBcn+1] + \Delta t
    (1 - \theta)[Fn + NBcn]
%
% Input:
%  xmin : Lower spatial boundary
%  xmax : Upper spatial boundary
%  Ne : Number of elements
%  order : weather the basis functions is linear or quadratic
%  theta : Method selection
%  time : All time related values combined
%  GQ : Gaussian Quadrature parameters
%  boundary : Dirichlet and Neumann conditions combined
%  parameters : Parameters for current material
% Return:
%  c : Solution to the full transient FEM
%  mesh : Finite element mesh
%  GQ : Gaussian Quadrature parameters
%  time : All time related values combined
%
%Candidate \#13442 - November 2023

    %finite element mesh between Xmin and Xmax with Ne number of elements
    mesh = OneDimLinearMeshGen(Xmin,Xmax,Ne,order,parameters);

    %get Gaussian points and weights
    [GQ] = GQscheme(GQ);

    %delta t
    dt = time.dt;

    %get global mass matrix
    [GMmass] = GlobalMassMatrix(Ne,mesh,GQ,order);

    %get global stiffness matrix
    [GMstiffness] = GlobalStiffnessMatrix(Ne,mesh,GQ,order);

    %current global matrix [M + \theta\Delta tK]
    GM = GMmass + theta*dt*GMstiffness;
    %previous global matrix [M - (1-\theta)\Delta tK]
    prevGM = GMmass - ((1-theta)*dt*GMstiffness);

    %initialise list of vector c with zeros
    c = zeros(mesh.ngn,length(time.t));
    %initial condition c(x,0)=initial condition
    c(:,1) = time.ic;

    %get global source vector
    [GVsource] = GlobalSourceVector(Ne,mesh,GQ,order);

    %Neumann boundary conditions, current and following
    [NB,NBnext] = NeumannBoundary(boundary,mesh);

    %compute RHS hence c at each time point
    for n = 1:time.N

        %RHS vector of equation
        RHS = (prevGM*c(:,n)) + dt*theta*(GVsource+NBnext) + dt*(1-theta)*(GVsource + NB);

        %Dirichlet boundary conditionsfor RHS and GM
        [RHS,GM] = DirichletBoundary(boundary,RHS,GM);

        %next numerical according to equation
        c(:,n+1) = GM\RHS;

```

```
    end
end
```

Listing 2. One dimensional mesh generator

```
function [mesh] = OneDimLinearMeshGen(xmin,xmax,Ne,order,parameters)
%%This function generates a one dimensional, equispaced, linear finite
%%element mesh, with Ne number of elements, between the points at x
%%position xmin and xmax.

    mesh.ne = Ne; %set number of elements
    mesh.ngn = Ne+1 + (Ne*(order -1)); %set number of global nodes
    mesh.nvec = zeros(mesh.ngn,1); %allocate vector to store global node values
    dx = (xmax - xmin)/Ne; %calculate element size

    mesh.nvec = xmin:dx-(0.5*dx*(order-1)):xmax;

    %loop over elements & set the element properties
    for i=1:Ne
        %set element Jacobian based on mapping to standard element
        mesh.elem(i).J = 0.5*dx; %this is assuming standard element of -1 to 1

        switch order
            case 1 %linear
                %set spatial positions of nodes
                mesh.elem(i).x(1) = xmin + (i-1)*dx;
                mesh.elem(i).x(2) = xmin + i*dx ;
                %set global IDs of the nodes
                mesh.elem(i).n(1) = i;
                mesh.elem(i).n(2) = i+1;

            case 2 %quadratic
                %set spatial positions of nodes
                mesh.elem(i).x(1) = xmin + (i-1)*dx;
                mesh.elem(i).x(2) = xmin + i*dx - dx/2;
                mesh.elem(i).x(3) = xmin + i*dx ;
                %set global IDs of the nodes
                mesh.elem(i).n(1) = (i*2)-1;
                mesh.elem(i).n(2) = (i*2);
                mesh.elem(i).n(3) = (i*2)+1;
        end

        %check weather using material 1 or 2
        if parameters.selection == '1'

            %diffusion coefficient of current element
            mesh.elem(i).D = parameters.D;
            %reaction coefficient of current element
            mesh.elem(i).lambda = parameters.lambda;
            %source term of current element
            mesh.elem(i).f = parameters.f;

        elseif parameters.selection == '2'
            %position on the mesh of current element
            position = mesh.elem(i).x(1);

            % Set parameters values based on current position
            %epidermis layer
            if(position >= 0 && position < parameters.Xe)
                %diffusion coefficient and no blood flow
                mesh.elem(i).D = parameters.De;
                mesh.elem(i).beta = parameters.betaE;
                mesh.elem(i).gamma = parameters.gammaE;

            %dermis layer
            elseif (position >= parameters.Xe && position < parameters.Xd)
                %diffusion coefficient
                mesh.elem(i).D = parameters.Dd;
                mesh.elem(i).beta = parameters.betaD;
                mesh.elem(i).gamma = parameters.gammaD;
```

```
            %sub-cutaneous layer
            elseif (position >= parameters.Xd && position <= parameters.Xb)
                %diffusion coefficient
                mesh.elem(i).D = parameters.Db;
                mesh.elem(i).beta = parameters.betaB;
                mesh.elem(i).gamma = parameters.gammaB;
            end

            %sum of the reaction terms
            mesh.elem(i).lambda = -(mesh.elem(i).beta + mesh.elem(i).gamma);
            %no source term
            mesh.elem(i).f = 0;
        end
    end
end
```

Listing 3. Global mass matrix calculator

```
function [GMmass] = GlobalMassMatrix(Ne,mesh,GQ,order)
%Assembles the single local mass elements into a global matrix
% The local mass elements are computed at each element in the finite
% element mesh. Based on their location they are then inserted in the
% global mass matrix of size mesh.ngn-by-mesh.ngn.
%
% Input:
%  Ne : Number of elements
%  mesh : Finite element mesh
%  GQ : Gaussian Quadrature parameters
%  order : weather the basis functions is linear or quadratic
% Return:
%  GMmass : Global Mass matrix
%
%Candidate \#13442 - November 2023

    %initialize global mass matrix with zeros
    GMmass = zeros(mesh.ngn);

    %calculate and assembly the global mass matrix
    for eN = 1:Ne

        %local matrix for mass element
        [LMmass] = LocalMassElem(eN,mesh,GQ,order);

        %location in GMatrix
        i = order*eN - (order-1);

        GMmass(i:i+order,i:i+order) = GMmass(i:i+order,i:i+order) + LMmass;
    end
end
```

Listing 4. Global stiffness matrix calculator

```
function [GMstiffness] = GlobalStiffnessMatrix(Ne,mesh,GQ,order)
%Assembles the single local diffusion and reaction elements into a global matrix
% The local diffusion and reaction elements are computed at each element
% in the finite element mesh. Based on their location they are then
% inserted in the global stiffness matrix of size mesh.ngn-by-mesh.ngn.
%
% Input:
%  Ne : Number of elements
%  mesh : Finite element mesh
%  GQ : Gaussian Quadrature parameters
%  order : weather the basis functions is linear or quadratic
% Return:
%  GMstiffness : Global Stiffness matrix
%
%Candidate \#13442 - November 2023

    %initialize matrix with zeros
```

```
    GMstiffness = zeros(mesh.ngn);

    %calculate and assembly the global stiffness matrix
    for eN = 1:Ne
        %local Laplace Element Matrix for diffusion
        [LMdiffusion] = LEMdiffusion(eN, mesh,GQ,order);
        %local Laplace Element Matrix for reaction
        [LMreaction] = LEMreaction(eN, mesh,GQ,order);

        %difference between diffusion and reaction
        diff = LMdiffusion - LMreaction;
        %location in GMatrix
        i = order*eN - (order-1);
        %add local matrices into global matrix
        GMstiffness(i:i+order,i:i+order) = GMstiffness(i:i+order,i:i+order) + diff;
    end
end
```

Listing 5. Global source vector calculator

```
function [GVsource] = GlobalSourceVector(Ne,mesh,GQ,order)
%Assembles the single local source elements into a global vector
% The local source elements are computed at each element in the finite
% element mesh. Based on their location they are then inserted in the
% global souce vector of size mesh.ngn-by-1.
%
% Input:
%  Ne : Number of elements
%  mesh : Finite element mesh
%  GQ : Gaussian Quadrature parameters
%  order : weather the basis functions is linear or quadratic
% Return:
%  GVsource : Global Source vector
%
%Candidate \#13442 - November 2023

    %initialize matrix with zeros
    GVsource = zeros(mesh.ngn, 1);
    for eN = 1:Ne
        %local 2-by-1 vector for corresponding eN
        [LVsource] = LocalSourceElem(eN,mesh,GQ,order);

        %location in GVector
        i = order*eN - (order-1);
        %add local vectors into global vector
        GVsource(i:i+order,1) = GVsource(i:i+order,1) + LVsource;
    end

end
```

Listing 6. Neumann boundary calculator

```
function [NB,NBnext] = NeumannBoundary(boundary,mesh)
%Computes the Neumann Boundary Conditions
% This is done according to the lower and upper boundary values contained
% within boundary
%
% Input:
%  Boundary : Data structure containing all the boundary conditions
%  mesh : Finite element mesh
% Return:
%  NB : current Neumann boundary conditions vector
%  NBnext : next Neumann boundary conditions vector
%
%Candidate \#13442 - November 2023

    %initialise boundary conditions vector with zeros
    NB = zeros(mesh.ngn,1);

    %Neumann boundaries implementation
```

```
    if isnumeric(boundary.NeumannL)
        %lower Neumann BC
        NB(1) = -boundary.NeumannL;
    elseif isnumeric(boundary.NeumannU)
        %upper Neumann BC
        NB(mesh.ngn) = boundary.NeumannU;
    end
    %store the next BC
    NBnext = NB;
end
```

Listing 7. Local mass element calculator

```
function [LMmass] = LocalMassElem(eN,mesh,GQ,order)
%Calculates the Local m-by-n Element Matrix for the linear mass operator
% This is done for any element eN in the finite element mesh.
%
% Input:
%  eN : Element number
%  mesh : Finite element mesh
%  GQ : Gaussian Quadrature parameters
%  order : weather the basis functions is linear or quadratic
% Return:
%  LMmass : Local mass matrix
%
%Candidate \#13442 - November 2023

    %Jacobian for a given element
    J = mesh.elem(eN).J;
    %initialise mass matrix with zeros
    LMmass = zeros(order+1);
    %initialise local element matrix with zeros
    Int00 = zeros(order+1);

    %if manual integration
    if GQ.switch == '0'
        %elements of local element matrix
        Int00 = 2*J/3;
        %local 2x2 Element Matrix for mass
        LMmass = [Int00 Int00/2; Int00/2 Int00];

    %if gaussian quadrature
    elseif GQ.switch == '1'
        %number of gauss points
        N = GQ.npts;
        %get Gaussian points and weights
        [GQ] = GQscheme(GQ);

        for i = 1:N
            w = GQ.gw(i);          %Gauss weights
            xipts = GQ.xipts(i);   %Gauss points
           %basis function value at given point
            [psi,~] = EvalBasis(order,xipts);
            %local element mass matrix
            for m = 1:order+1
                for n = 1:order+1
                    Int00(m,n) = psi(m)*psi(n)*J;
                    LMmass(m,n) = LMmass(m,n) + w*(Int00(m,n));
                end
            end
        end
    end
end
```

Listing 8. Diffusion local element matrix calculator

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

```
% Input:
%  eN : Element number
%  mesh : Finite element mesh
%  GQ : Gaussian Quadrature parameters
%  order : weather the basis functions is linear or quadratic
% Return:
%  DiffusionMatrix : Local diffusion matrix
%
%Candidate \#13442 - November 2023

    %Jacobian for a given element
    J = mesh.elem(eN).J;
    %initialise matrix with zeros
    DiffusionMatrix = zeros(order+1);
    %initialise values of matrix with zeros
    Int00 = zeros(order +1);
    %diffusion coefficient values
    D = mesh.elem(eN).D;

    %if manual integration
    if GQ.switch == '0'
        %elements of local element matrix 2J = (x1-x0)
        Int00 = D/(2*J);
        %local 2x2 Element Matrix for diffusion
        DiffusionMatrix = [Int00 -Int00 ; -Int00 Int00];

    %if Gaussian Quadrature
    elseif GQ.switch == '1'
        %number of gauss points
        N = GQ.npts;
        %get Gaussian points and weights
        [GQ] = GQscheme(GQ);

        for i = 1:N
            w = GQ.gw(i);          %Gauss weights
            xipts = GQ.xipts(i);   %Gauss points
            %basis function at given point
            [~,dpsidxigrad] = EvalBasis(order,xipts);
            %local element diffusion matrix
            for m = 1:order+1
                for n = 1:order+1
                    Int00(m,n) = D*dpsidxigrad(m)*dpsidxigrad(n)/J;
                    DiffusionMatrix(m,n) = DiffusionMatrix(m,n) + w*(Int00(m,n));
                end
            end
        end
    end
end
```

Listing 9. Reaction local element matrix calculator

```
function [ReactionMatrix] = LEMreaction(eN,mesh,GQ,order)
%Calculates the Local m-by-n Element Matrix for the linear reaction operator
% This is done for any element eN in the finite element mesh.
%
% Input:
%  eN : Element number
%  mesh : Finite element mesh
%  GQ : Gaussian Quadrature parameters
%  order : weather the basis functions is linear or quadratic
% Return:
%  ReactionMatrix : Local reaction matrix
%
%Candidate \#13442 - November 2023

    %Jacobian of given element
    J = mesh.elem(eN).J;
    %initialise matrix with zeros
    ReactionMatrix = zeros(order+1);
    %initialise values of matrix with zeros
```

```
    Int00 = zeros(order +1);
    %reaction coefficient values
    lambda = mesh.elem(eN).lambda;

    %if manual integration
    if GQ.switch == '0'
        %elements of local element matrix
        Int00 = 2*(lambda*J)/3;
        %local 2x2 Element Matrix for reaction
        ReactionMatrix = [Int00 Int00/2 ; Int00/2 Int00];

    %if gaussian quadrature
    elseif GQ.switch == '1'
        %number of gauss points
        N = GQ.npts;
        %get Gaussian points and weights
        [GQ] = GQscheme(GQ);

        for i = 1:N
            w = GQ.gw(i);          %Gauss weights
            xipts = GQ.xipts(i);   %Gauss points
            %basis function value at given point
            [psi,~] = EvalBasis(order,xipts);
            %local element reaction matrix
            for m = 1:order+1
                for n = 1:order+1
                    Int00(m,n) = lambda*psi(m)*psi(n)*J;
                    ReactionMatrix(m,n) = ReactionMatrix(m,n) + w*(Int00(m,n));
                end
            end
        end
    end
end
```

Listing 10. Local source element calculator

```
function [LVsource] = LocalSourceElem(eN,mesh,GQ,order)
%Calculates the Local m-by-1 Element vector for the source term
% This is done for any element eN in the finite element mesh.
%
% Input:
%  eN : Element number
%  mesh : Finite element mesh
%  GQ : Gaussian Quadrature parameters
%  order : weather the basis functions is linear or quadratic
% Return:
%  LVsource : Local element source vector
%
%Candidate \#13442 - November 2023

    %Jacobian for a given element
    J = mesh.elem(eN).J;
    %initialise source vector with zeros
    LVsource = zeros(order+1,1);
    %initialise local element vector with zeros
    Int00 = zeros(order+1);
    %source term coefficient values
    f = mesh.elem(eN).f;

    %if manual integration
    if GQ.switch == '0'
        %value of elements in matrix
        Int00 = f*J;
        %local 2-by-1 vector for corresponding eN
        LVsource = [Int00;Int00];

    %if gaussian quadrature
    elseif GQ.switch == '1'
        %number of gauss points
        N = GQ.npts;
```

```
        %get Gaussian points and weights
        [GQ] = GQscheme(GQ);

        for i = 1:N
            w = GQ.gw(i);          %Gauss weights
            xipts = GQ.xipts(i);   %Gauss points
            %basis function value at given point
            [psi,~] = EvalBasis(order,xipts);
            %local element source vector
            for m = 1:order+1
                Int00(m) = f*psi(m)*J;
                LVsource(m) = LVsource(m) + w*(Int00(m));
            end
        end
    end
end
```

Listing 11. Gaussian Quadrature scheme generator

```
function [gq] = GQscheme(gq)
%Creates a Gauss-Legendre Quadrature scheme data structure of order N
% The scheme stores both the quadrature weights and the Legendre points.
% Values found @
% https://en.wikipedia.org/wiki/Gauss-Legendre_quadrature#Definition
%
% Input:
%  GQ : Gaussian Quadrature initial parameters
% Return:
%  GQ : Gaussian Quadrature with weights and points
%
%Candidate \#13442 - November 2023

    %order of quadrature rule
    N = gq.npts;

    if (N>0) && (N<6)
        %initialise zero array of size equal gauss quadrature order
        gq.gw = zeros(N,1);     %Gauss weights
        gq.xipts = zeros(N,1);  %Gauss points

        switch N
            case 1
              gq.gw(1) = 2;
              gq.xipts(1) = 0;
            case 2
              gq.gw(1) = 1;
              gq.gw(2) = 1;
              gq.xipts(1) = -sqrt(1/3);
              gq.xipts(2) = sqrt(1/3);
            case 3
              gq.gw(1) = 5/9;
              gq.gw(2) = 8/9;
              gq.gw(3) = 5/9;
              gq.xipts(1) = -sqrt(3/5);
              gq.xipts(2) = 0;
              gq.xipts(3) = sqrt(3/5);
            case 4
              gq.gw(1) = (18+sqrt(30))/36;
              gq.gw(2) = (18+sqrt(30))/36;
              gq.gw(3) = (18-sqrt(30))/36;
              gq.gw(4) = (18-sqrt(30))/36;
              gq.xipts(1) = -sqrt((3/7)-(2/7)*sqrt(6/5));
              gq.xipts(2) = sqrt((3/7)-(2/7)*sqrt(6/5));
              gq.xipts(3) = -sqrt((3/7)+(2/7)*sqrt(6/5));
              gq.xipts(4) = sqrt((3/7)+(2/7)*sqrt(6/5));
            case 5
              gq.gw(1) = 128/225;
              gq.gw(2) = (322+13*sqrt(70))/900;
              gq.gw(3) = (322+13*sqrt(70))/900;
              gq.gw(4) = (322-13*sqrt(70))/900;
```

```
            gq.gw(5) = (322-13*sqrt(70))/900;
            gq.xipts(1) = 0.0;
            gq.xipts(2) = -(1/3)*sqrt(5-2*sqrt(10/7));
            gq.xipts(3) = (1/3)*sqrt(5-2*sqrt(10/7));
            gq.xipts(4) = -(1/3)*sqrt(5+2*sqrt(10/7));
            gq.xipts(5) = (1/3)*sqrt(5+2*sqrt(10/7));
        end
    else
      fprintf('Invalid number of Gauss points (N)');
    end
end
```

Listing 12. Basis function evaluator

```
function [psi, dpsidxigrad] = EvalBasis(order,xipts)
%Evaluates the basis functions used in the model
% The basis functions are evaluated by calculating their values and gradients
% for a given local node (lnid) at a xipts.
%
% Input:
%  order : weather the basis functions is linear or quadratic
%  xipts : x points between [-1,1]
% Return:
%  psi : the value of the basis function
%  nvecnext : the gradient of the basis function
%
%Candidate \#13442 - November 2023

    %initialise psi as zeros array, with size order + 1
    psi = zeros(order+1,1);
    dpsidxigrad = zeros(order+1,1);

    %linear or quadratic basis functions
    switch order
        %linear
        case 1
            %function values
            for lnid = 0:1
                sign = (-1)^(lnid+1);
                psi(lnid+1) = 0.5*(1+((sign*xipts)));
            end
            %function gradients
            for lnid = 1:order+1
                sign1 = (-1)^(lnid);
                dpsidxigrad(lnid) = 0.5 * sign1;
            end
        %quadratic
        case 2
            %function values
            psi(1) = xipts*(xipts - 1)*0.5;
            psi(2) = (1 - xipts^2);
            psi(3) = xipts*(xipts + 1)*0.5;
            %function gradients
            dpsidxigrad(order-1) = xipts-0.5;
            dpsidxigrad(order) = -2*xipts;
            dpsidxigrad(order+1) = xipts+0.5;
    end
end
```

Listing 13. Dirichlet boundary calculator

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

```
% Return:
%  RHS : RHS vector with DBC applied
%  GlobalMatrix : Global Matrix with DBC applied
%
%Candidate \#13442 - November 2023

    %upper Dirichlet BC
    GlobalMatrix(end, :) = 0;
    GlobalMatrix(end) = 1;
    RHS(end) = boundary.DirichletU;

    %lower Dirichlet BC
    GlobalMatrix(1, :) = 0;
    GlobalMatrix(1, 1) = 1;
    RHS(1) = boundary.DirichletL;

end
```

Listing 14. Main script running the equation solver

```
%This script runs the transient FEM solver and plots some graphs used to
% prove the code functioning for Part 1 of ME40064, Transient MATLAB-Based
% FEM Modelling, CourseWork.
%
%Candidate \#13442 - November 2023
close all

%% ------- Imput parameters -----------------------------------------------%

% Parameters of current material
parameters.selection = '1';  %material for Part 1
parameters.D = 1;            %Diffusion coefficient
parameters.lambda = 0;       %Reaction coefficient
parameters.f = 0;            %Source term

Xmin = 0;   %lower spatial boundary
Xmax = 1;   %upper spatial boundary
Ne = 10;    %number of elements
order = 1;  %order (linear(1) or quadratic(2)) of basis function
theta = 1;  %0:Forward Euler, 1:Backward Euler, 0.5:Crank-Nicolson
Xpoints = Xmin:0.01:Xmax; %vector with x points for analytical plotting

% Time related values
time.tmin = 0;  %start time
time.tmax = 1;  %end time
time.ic = 0;    %initial condition time
time.dt = 0.01; %delta t
time.t = time.tmin:time.dt:time.tmax;   %entire time vector
time.N = (time.tmax-time.tmin)/time.dt; %number of time steps
tpoints = [0.05 0.1 0.3 1.0];   %given time points

GQ.switch = '1';    %Gaussian Quadrature yes or no
GQ.npts = 2;        %number of gauss points (2N-1)
[GQ] = GQscheme(GQ);%create Gaussian Quadrature

% All boundaries combined
boundary.DirichletL = 0;    %Lower Dirichlet Boundary Condition
boundary.DirichletU = 1;    %Upper Dirichlet Boundary Condition
boundary.NeumannL = 'Na';   %Lower Neumann Boundary Condition
boundary.NeumannU = 'Na';   %Upper Neumann Boundary Condition

%get numerical solution for the transient FEM
[Cnum,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);


%% ------- Graphs for 1.1 -------------------------------------------------%
%first figure for computed solution alone
figure('Name','Part1 numerical')
for i = 1:length(tpoints)

```

```
    %Plot c(x) against x
    element = time.t == tpoints(i);
    plot(mesh.nvec,Cnum(:,element),'DisplayName',(strcat('t=',num2str(tpoints(i)))),'LineWidth',1.3)
    ;
    hold on
end
grid on %use grid lines
title('Transient FEM numerical solutions','FontSize',14)
xlabel('Spatial distance x','FontSize',12)
ylabel('Concentration c(x,t)','FontSize',12)
legend('Location','NorthWest','FontSize',10)
% save plot as picture
saveas(gcf,'TransientFEM','png')

%second figure for analytical vs numerical comparison
figure('Name', 'Part1 analytical')
%Defines line colours using MATLAB RGB triplet
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]};
Canalytical = zeros(length(Xpoints),length(tpoints));

for i = 1:length(tpoints)
    for m = 1:length(Xpoints)
        %Calculates analytical solution
        Canalytical(m,i) = TransientAnalyticSoln(Xpoints(m),tpoints(i));
    end

    %plot analytical solution against x
    plot(Xpoints, Canalytical(:,i),'DisplayName',strcat('Analytical Solution @ t=',num2str(tpoints(i
    ))),'LineStyle','--','LineWidth',1.3,'color', colours{i});
    hold on

    %plot c(x) against x
    element = time.t == tpoints(i);
    plot(mesh.nvec,Cnum(:,element),'DisplayName',(strcat('Numerical Solution @ t=',num2str(tpoints(i
    )))),'LineWidth',1.3,'color',colours{i});
    hold on
end
grid on %use grid lines
title('Numerical vs Analytical solutions','FontSize',14)
xlabel('Spatial distance x','FontSize',12)
ylabel('Concentration c(x,t)','FontSize',12)
legend('Location','NorthWest','FontSize',10)
% save plot as picture
saveas(gcf,'TransientFEM-analytical','png')

%% ------- Graphs for 1.2 ------------------------------------------------%
%figure for analytical vs numerical comparison @ x = 0.8
figure('Name','Part1 analytical @ 0.8')

element = mesh.nvec == 0.8;
plot(time.t,Cnum(element,:),'DisplayName','Numerical Solution @ x=0.8','LineWidth',1.3);
hold on

Canalytical = TransientAnalyticSoln(0.8,time.t);
plot(time.t,Canalytical,'DisplayName','Analytical Solution @ x=0.8','LineWidth',1.3);
hold on

grid on %use grid lines
title('Numerical vs Analytical solution at x=0.8','FontSize',14)
xlabel('Time t (s)','FontSize',12)
ylabel('Concentration c(0.8,t)','FontSize',12)
legend('Location','SouthEast','FontSize',10)
% save plot as picture
saveas(gcf,'TransientFEM-analytical@08','png')

%% ------- Backward Euler vs Crank-Nicolson ------------------------------%
%figure for different theta scheme comparison @ x = 0.8
figure('Name','Crank-Nicolson, Backward and Forward @ x = 0.8')

```

```
for theta = [0 0.5 1]
    %get numerical solution for given theta
    [Cnum,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);

    element = mesh.nvec == 0.8;
    plot(time.t,Cnum(element,:),'-x','DisplayName',strcat('Numerical Solution   =',num2str(theta)),'
    LineWidth',1.3);
    hold on
end

%Plots numerical solution at x = 0.8 for the established time interval
Canalytical = TransientAnalyticSoln(0.8,time.t);
plot(time.t,Canalytical,'DisplayName','Analytical Solution','LineWidth',1.3);

grid on %use grid lines
title('Crank-Nicolson vs Euler methods','FontSize',14)
xlabel('Time t (s)','FontSize',12)
ylabel('Concentration c(0.8,t)','FontSize',12)
legend('Location','SouthEast','FontSize',10)
ylim([0,0.8])
xlim([0,0.2])
% save plot as picture
saveas(gcf,'Crank-Nicolson_vs_Euler','png')

%% ------- Linear vs Quadratic basis function ----------------------------%
%figure for different basis function order
figure('Name','Basis function order')

Canalytical = zeros(length(Xpoints),1);
for m = 1:length(Xpoints)
    %Calculates analytical solution
    Canalytical(m,1) = TransientAnalyticSoln(Xpoints(m),0.05);
end
%plot analytical solution against x
plot(Xpoints, Canalytical(:,1),'DisplayName','Analytical Solution','LineWidth',1.3);
hold on

for order = [1 2]
    [Cnum,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);
    %plot c(x) against x
    element = time.t == 0.05;
    plot(mesh.nvec,Cnum(:,element),'-x','DisplayName',(strcat('Numerical Solution order=',num2str(
    order))),'LineWidth',1.3);
    hold on
end

grid on %use grid lines
title('Basis Function order','FontSize',14)
xlabel('Spatial distance x','FontSize',12)
ylabel('Concentration c(x,0.05)','FontSize',12)
legend('Location','NorthWest','FontSize',10)
xlim([0.4,0.8])
% save plot as picture
saveas(gcf,'BasisFunctionOrder','png')
```

Listing 15. Analytical solution calculator

```
function [ c ] = TransientAnalyticSoln(x,t)
%TransientAnalyticSonl Analytical solution to transient diffusion equation
%   Computes the analytical solution to the transient diffusion equation for
%   the domain x=[0,1], subject to initial condition: c(x,0) = 0, and Dirichlet
%   boundary conditions: c(0,t) = 0, and c(1,t) = 1.
%   Input Arguments:
%   x is the point in space to evaluate the solution at
%   t is the point in time to evaluate the solution at
%   Output Argument:
%   c is the value of concentration at point x and time t, i.e. c(x,t)

trans = 0.0;

```

```
for k=1:1000
    trans = trans + ((((-1)^k)/k) * exp(-k^2*pi^2*t)*sin(k*pi*x));
end

c = x + (2/pi)*trans;

end
```

Listing 16. L2-norm calculator

```
function [L2Norm,h] = L2Norm(Xmin,Xmax,elements,order,theta,time,GQ,boundary,parameters)
%Computes the L2 Norm of the transient diffusion reaction equation.
% The L2 Norm is the error between the numerical and the analytical
% solution. This is done to test convergence of the TransientFEM calculator
% over a range of elements.
%
% Input:
%  Xmin : Lower spatial boundary
%  Xmax : Upper spatial boundary
%  elements : given elements
%  order : weather the basis functions is linear or quadratic
%  theta : Method selection
%  time : All time related values combined
%  GQ : Gaussian Quadrature parameters
%  boundary : Dirichlet and Neumann conditions combined
%  parameters : Parameters for current material
% Return:
%  L2Norm : L2 Norm error
%  h : Characteristic length
%
%Candidate \#13442 - November 2023

    %number of gauss points
    N = GQ.npts;
    %preallocations for fast processing
    h = zeros(1,length(elements));
    L2Norm = zeros(1,length(elements));
    E = zeros(1,N);

    for b = 1:length(elements)
        %Defines number of elements
        Ne = elements(b);
        Etot = zeros(1,Ne); %preallocation for fast processing
        %length for each element
        h(b) = ((Xmax-Xmin)/Ne);
        %solve transient FEM
        [Cnum,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);
        %Returns the solution column at the selected time
        cnumerical = Cnum(:,time.t == time.range);
        for eN = 1:Ne
            %Jacobian for a given element
            J = mesh.elem(eN).J;
            %rows of the local nodes at given eN
            Cnoderows = cnumerical(mesh.elem(eN).n(1):mesh.elem(eN).n(end),1);
            for i = 1:N
                w = GQ.gw(i);          %Gauss weights
                xipts = GQ.xipts(i);   %Gauss points
                %basis function at given point
                [psi,~] = EvalBasis(order,xipts);
                %interpolation between nodes
                CNum = psi'*Cnoderows;
                %Finds x position
                x = mesh.elem(eN).x*psi;
                %analytical solution
                CAnalyt = TransientAnalyticSoln(x,time.range);
                %error squared
                E(i) = w*J*(CAnalyt-CNum)^2;
            end
            %total error
            Etot(eN) = sum(E);
```

```
        end
        %L2 Norm is RMS of total error
        L2Norm(b) = sqrt(sum(Etot));
    end
    %coefficients of polynomial to fit the data
    y = polyfit(log(h),log(L2Norm),1);
    %error gradient
    gradient = y(1);
end
```

Listing 17. Script running the L2-norm

```
%This script runs the transient FEM solver and plots the L2 norm used to
% test the convergence rate of the finite element method function.
%
%Candidate \#13442 - November 2023
close all

%------- Imput parameters for L2 norm ------------------------------------%

% Parameters of current material
parameters.selection = '1';  %material for Part 1
parameters.D = 1;            %Diffusion coefficient
parameters.lambda = 0;       %Reaction coefficient
parameters.f = 0;            %Source term

Xmin = 0;   %lower spatial boundary
Xmax = 1;   %upper spatial boundary
Ne = 10;    %number of elements
elements = [2 5 10 15 20 25 30 35 40 45]; %choosen elements

% Time related values
time.tmin = 0;  %start time
time.tmax = 1;  %end time
time.ic = 0;    %initial conditiopn time
time.dt = 0.0001; %delta t
time.t = time.tmin:time.dt:time.tmax;   %entire time vector
time.N = (time.tmax-time.tmin)/time.dt; %number of time steps
tpoints = [0.05 0.1 0.3 1.0];   %given time points

GQ.switch = '1';    %Gaussian Quadrature yes or no
GQ.npts = 3;        %number of gauss points (2N-1)
[GQ] = GQscheme(GQ);%create Gaussian Quadrature

% All boundaries combined
boundary.DirichletL = 0;    %Lower Dirichlet Boundary Condition
boundary.DirichletU = 1;    %Upper Dirichlet Boundary Condition
boundary.NeumannL = 'Na';   %Lower Neumann Boundary Condition
boundary.NeumannU = 'Na';   %Upper Neumann Boundary Condition


%------- Linear Vs Quadratic Basis Function ------------------------------%
%Loops through selected elements range switching between linear and quadratic basis functions
for order = [1 2]
    figure('Name',strcat('L2 Norm test, basis order = ',num2str(order)))
    %predefined line colours [rgb]
    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]};
    %Crank-Nicolson or Backward Euler methods
    for theta = [0.5 1]

        for i = 1:length(tpoints)
            %get L2 Norm error
            time.range = tpoints(i);
            [L2N, h] = L2Norm(Xmin,Xmax,elements,order,theta,time,GQ,boundary,parameters);
            %Crank-Nicolson or Backward Euler methods
            switch theta
                case 0.5
                    %plot L2 Norm error against characteristic lenght using ln plot
                    plot(log(h),log(L2N),'DisplayName',strcat('t=',num2str(time.range)),'LineStyle',
    '-.','LineWidth',1.3,'color',linecolor{i});
```

```
                case 1
                    %plot L2 Norm error against characteristic lenght using ln plot
                    plot(log(h),log(L2N),'DisplayName',strcat('t=',num2str(time.range)),'LineStyle',
    '--','LineWidth',1.3,'color',linecolor{i});
            end
            hold on
        end
    end
    set(gca, 'XScale', 'log', 'YScale', 'log'); %use log axis
    grid on %use grid lines
    title(strcat('L2 Norm with basis order=',num2str(order)),'FontSize',14)
    xlabel('ln(h)','FontSize',12);
    ylabel('ln(E)','FontSize',12);
    legend('Location','SouthEast','FontSize',10,'NumColumns',2)
    % save plot as picture
    saveas(gcf,strcat('L2Norm_order=',num2str(order)),'png')
end


%------- Manual integration Vs. Gaussian Quadrature ----------------------%
order = 1;
time.range = time.tmax;
figure('Name','Numerical Methods Convergence')
%manual or gaussian quadrature integration methods
for j = [0 1]
    GQ.Switch = num2str(j);
    %Crank-Nicolson or Backward Euler methods
    for theta = [0.5 1]
        %get L2 Norm error
        [L2N, h] = L2Norm(Xmin,Xmax,elements,order,theta,time,GQ,boundary,parameters);
        %two line colours
        if theta == 0.5
            linecolor = [0.00 0.45 0.75];
        elseif theta == 1
            linecolor = [0.85 0.35 0.10];
        end
        %manual or gaussian quadrature
        switch j
            case 0
                %plot L2 Norm error against characteristic lenght using ln plot
                plot(log(h),log(L2N),'LineWidth',1.3,'color',linecolor);
            case 1
                %plot L2 Norm error against characteristic lenght using ln plot
                plot(log(h),log(L2N),'--','LineWidth',1.3,'color',linecolor);
        end
        hold on
    end
end
set(gca, 'XScale', 'log', 'YScale', 'log'); %use log axis
grid on %use grid lines
title('Numerical Methods Convergence','FontSize',14)
xlabel('ln(h)','FontSize',12);
ylabel('ln(E)','FontSize',12)
legend('Crank-Nicolson (Manual Integration)','Backward Euler (Manual Integration)',...
'Crank-Nicolson (Gaussian Quadrature)','Backward Euler (Gaussian Quadrature)','Location','SouthEast'
    ,'FontSize',10)
% save plot as picture
saveas(gcf,'NumericalMethodsConvergence','png')
```

Listing 18. Main script running the skin drug diffusion analysis

```
%This script solves the drug delivery through uman skin problem.
% Various graphs are used to show the results and to do some analysis for
% Part 2 of ME40064, Transient MATLAB-Based FEM Modelling, CourseWork.
%
%Candidate \#13442 - December 2023
close all

%% ------- Imput parameters ------------------------------------------------%

```

```
% Parameters of current material
parameters.selection = '2';  %material for Part 2
parameters.De = 4e-6;        %Diffusion coefficient epidermis
parameters.Dd = 5e-6;        %Diffusion coefficient dermis
parameters.Db = 2e-6;        %Diffusion coefficient sub-cutaneous
parameters.betaE = 0;        %Reaction coefficient blood epidermis
parameters.betaD = 0.01;     %Reaction coefficient blood dermis
parameters.betaB = 0.01;     %Reaction coefficient blood sub-cutaneous
parameters.gammaE = 0.02;    %Reaction coefficient degradation epidermis
parameters.gammaD = 0.02;    %Reaction coefficient degradation dermis
parameters.gammaB = 0.02;    %Reaction coefficient degradation sub-cutaneous
parameters.Xe = 0.00166667;  %Epidermis x coordinate
parameters.Xd = 0.005;       %Dermis x coordinate
parameters.Xb = 0.01;        %Sub-cutaneous x coordinate

% Space related values
Xmin = 0;   %lower spatial boundary
Xmax = 0.01;   %upper spatial boundary
Ne = 40;    %number of elements
order = 2;  %order (linear(1) or quadratic(2)) of basis function
theta = 0.5;  %0:Forward Euler, 1:Backward Euler, 0.5:Crank-Nicolson

% Time related values
time.tmin = 0;  %start time
time.tmax = 30; %end time
time.ic = 0;    %initial condition time
time.dt = 0.01; %delta t
time.t = time.tmin:time.dt:time.tmax;   %entire time vector
time.N = (time.tmax-time.tmin)/time.dt; %number of time steps
tpoints = [0.05 0.1 0.3 1 2 3 5 7 10 20 30];   %given time points

GQ.switch = '1';    %Gaussian Quadrature yes or no
GQ.npts = 4;        %number of gauss points (2N-1)
[GQ] = GQscheme(GQ);%create Gaussian Quadrature

% All boundaries combined
boundary.DirichletL = 30;   %Lower Dirichlet Boundary Condition
boundary.DirichletU = 0;    %Upper Dirichlet Boundary Condition
boundary.NeumannL = 'Na';   %Lower Neumann Boundary Condition
boundary.NeumannU = 'Na';   %Upper Neumann Boundary Condition

%colours for plotting (parula colormap)
colours = parula(length(tpoints));

%Calculates numerical solution for the transient problem
[c,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);

%% ------- Drug concentration ---------------------------------------------%
figure('Name', 'Concentration')

%Loop through all the selected time points
for i = 1:length(tpoints)

    %Plots temperature distribution through the tissue for a range of
    %time points
    element = time.t == tpoints(i);
    plot(mesh.nvec,c(:,element),'DisplayName',(strcat('t=',num2str(tpoints(i)))),'LineWidth',1.3,'
    color',colours(i,:));
    hold on
end

%show where the epidermis and dermis end
xline(parameters.Xe,'-.k','Epidermis','FontSize',12,'LineWidth',0.75,'HandleVisibility','off');
xline(parameters.Xd,'-.k','Dermis','FontSize',12,'LineWidth',0.75,'HandleVisibility','off');

grid on %use grid lines
title('Drug concentration through the skin','FontSize',14)
xlabel('Skin Depth (m)','FontSize',12);
ylabel('Concentration  c(x,t)','FontSize',12);
```

```
legend('Location','NorthEast','FontSize',10,'NumColumns',2)
ylim([0,35])
% save plot as picture
saveas(gcf,'DrugConcentration','png')


%% ------- Drug effectiveness ----------------------------------------------%
figure('Name', 'Effectiveness')

% concentration for effectiveness at given position
ceff = 40;
Xpos = 0.005;

% increase boundary dose untill integral of c is less that 1000
K = 0;
while K < 1000
    [c,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);

    [K,teff,position,element] = MinEffectiveDose(mesh,time,order,c,Xpos,ceff);

    % increase dose applied to skin
    boundary.DirichletL = boundary.DirichletL + 1;
end

plot(time.t,c(element,:),'DisplayName','Concentration @ x=0.005','LineWidth',1.3);
hold on
plot(teff,c(element,position),'ro', 'MarkerSize', 10,'DisplayName','Minimum effective dose')
% Add label with coordinates
label = sprintf('[%.2f, %.1f]', teff, c(element,position));
text(teff + 1, c(element,position) - 1, label, 'FontSize', 10);

grid on %use grid lines
title('Drug effectiveness at Dermis end','FontSize',14)
xlabel('Time t (s)','FontSize',12);
ylabel('Concentration c(0.005,t)','FontSize',12);
legend('Location','SouthEast','FontSize',10)
% save plot as picture
saveas(gcf,'DrugEffect','png')

fprintf("Minimum initial dose: %d \n",boundary.DirichletL);

%% ------- Effective drug concentration ----------------------------------%
figure('Name', 'Concentration')

%Loop through all the selected time points
for i = 1:length(tpoints)

    %Plots temperature distribution through the tissue for a range of
    %time points
    element = time.t == tpoints(i);
    plot(mesh.nvec,c(:,element),'DisplayName',(strcat('t=',num2str(tpoints(i)))),'LineWidth',1.3,'
    color',colours(i,:));
    hold on
end

%show where the epidermis and dermis end
xline(parameters.Xe,'-.k','Epidermis','FontSize',12,'LineWidth',0.75,'HandleVisibility','off');
xline(parameters.Xd,'-.k','Dermis','FontSize',12,'LineWidth',0.75,'HandleVisibility','off');
%show the minimum concentration for effectiveness
yline(ceff,'--r','Effectiveness','FontSize',12,'LineWidth',1,'HandleVisibility','off')

grid on %use grid lines
title('Effective drug concentration through the skin','FontSize',14)
xlabel('Skin Depth (m)','FontSize',12);
ylabel('Concentration  c(x,t)','FontSize',12);
legend('Location','NorthEast','FontSize',10,'NumColumns',2)
ylim([0,75])
% save plot as picture
saveas(gcf,'EffectiveDrugConcentration','png')
```

```

%% ------- Parameters influence ------------------------------------------%
figure('Name', 'Parameters')

for i = [1 2 3 4 5 6 7 8]
    parameters = CombinationsDR(i,parameters);
    boundary.DirichletL = 30;
    % increase boundary dose untill integral of c is less that 1000
    K = 0;
    while K < 1000
        [c,mesh,GQ,time] = TransientFEM(Xmin,Xmax,Ne,order,theta,time,GQ,boundary,parameters);

        [K,teff,position,element] = MinEffectiveDose(mesh,time,order,c,Xpos,ceff);

        % increase dose applied to skin
        boundary.DirichletL = boundary.DirichletL + 1;
    end

    fprintf("Minimum initial dose: %d, with combination %d \n",boundary.DirichletL,i);
    plot(time.t,c(element,:),'DisplayName',strcat('Solution for combination: ',num2str(i)),'
    LineWidth',1.3);
    hold on
end

yline(ceff,'--r','LineWidth',0.75,'DisplayName','Minimum effective dose')
grid on %use grid lines
title('Drug effectiveness with different combinations','FontSize',14)
xlabel('Time t (s)','FontSize',12);
ylabel('Concentration c(0.005,t)','FontSize',12);
legend('Location','SouthEast','FontSize',10)
ylim([0,45])
% save plot as picture
saveas(gcf,'ParameterComparison','png')
```

Listing 19. Minimum effective dose calculator

```
function [K,teff,position,element] = MinEffectiveDose(mesh,time,order,c,Xpos,ceff)
%Computes the Minimum effective drug dose
% Given a position in the mesh and a value for the concentration for
% effectivensess this returns the integral of the curve between the
% effectiveness point and the end.
%
% Input:
%  mesh : Finite element mesh
%  time : All time related values combined
%  order : weather the basis functions is linear or quadratic
%  c : Solution to the full transient FEM
%  Xpos : Position along the mesh
%  ceff : effectiveness concentration
% Return:
%  K : current Neumann boundary conditions vector
%  teff : time of effectiveness
%  position : index of given Xpos
%  element : index of effective time
%
%Candidate \#13442 - November 2023

    % find in c(Xpos,:) where the effectiveness concentration is reached
    element = round(order*Xpos/(mesh.nvec(end)/mesh.ne));
    position = find(c(element,:) > ceff,1,'first');
    % use the index to return a time point
    teff = time.t(position);

    % compute integral between current time and final of c in dt
    K = trapz(c(element,position:end))*time.dt;
end
```

Listing 20. Combination of different diffusion reaction coefficients

```
function [parameters] = CombinationsDR(I,parameters)
```

```
%Stores a series of parameters combinations for diffusion and reaction coefficients.
% Changes the parameters data structure to have different coefficients for
% diffusion, beta and gamma. Entera a combination value in range [1:1:8]
%
% Input:
%  I : combination number
%  parameters : current parameters data structure
% Return:
%  parameters : new parameters data structure
%
%Candidate \#13442 - November 2023

    % this function should only be called in material 2 testing
    parameters.selection = '2';  %material for Part 2

    switch I
        case 1  % High diffusion
            parameters.De = 4e-5;        %Diffusion coefficient epidermis
            parameters.Dd = 5e-5;        %Diffusion coefficient dermis
            parameters.Db = 2e-5;        %Diffusion coefficient sub-cutaneous
            parameters.betaE = 0;        %Reaction coefficient blood epidermis
            parameters.betaD = 0.01;     %Reaction coefficient blood dermis
            parameters.betaB = 0.01;     %Reaction coefficient blood sub-cutaneous
            parameters.gammaE = 0.02;    %Reaction coefficient degradation epidermis
            parameters.gammaD = 0.02;    %Reaction coefficient degradation dermis
            parameters.gammaB = 0.02;    %Reaction coefficient degradation sub-cutaneous
        case 2  % Low diffusion
            parameters.De = 4e-7;        %Diffusion coefficient epidermis
            parameters.Dd = 5e-7;        %Diffusion coefficient dermis
            parameters.Db = 2e-7;        %Diffusion coefficient sub-cutaneous
            parameters.betaE = 0;        %Reaction coefficient blood epidermis
            parameters.betaD = 0.01;     %Reaction coefficient blood dermis
            parameters.betaB = 0.01;     %Reaction coefficient blood sub-cutaneous
            parameters.gammaE = 0.02;    %Reaction coefficient degradation epidermis
            parameters.gammaD = 0.02;    %Reaction coefficient degradation dermis
            parameters.gammaB = 0.02;    %Reaction coefficient degradation sub-cutaneous
        case 3  % high degradation
            parameters.De = 4e-6;        %Diffusion coefficient epidermis
            parameters.Dd = 5e-6;        %Diffusion coefficient dermis
            parameters.Db = 2e-6;        %Diffusion coefficient sub-cutaneous
            parameters.betaE = 0;        %Reaction coefficient blood epidermis
            parameters.betaD = 0.01;     %Reaction coefficient blood dermis
            parameters.betaB = 0.01;     %Reaction coefficient blood sub-cutaneous
            parameters.gammaE = 0.2;    %Reaction coefficient degradation epidermis
            parameters.gammaD = 0.2;    %Reaction coefficient degradation dermis
            parameters.gammaB = 0.2;    %Reaction coefficient degradation sub-cutaneous
        case 4  % high degradation, low blood
            parameters.De = 4e-6;        %Diffusion coefficient epidermis
            parameters.Dd = 5e-6;        %Diffusion coefficient dermis
            parameters.Db = 2e-6;        %Diffusion coefficient sub-cutaneous
            parameters.betaE = 0;        %Reaction coefficient blood epidermis
            parameters.betaD = 0.001;     %Reaction coefficient blood dermis
            parameters.betaB = 0.001;     %Reaction coefficient blood sub-cutaneous
            parameters.gammaE = 0.2;    %Reaction coefficient degradation epidermis
            parameters.gammaD = 0.2;    %Reaction coefficient degradation dermis
            parameters.gammaB = 0.2;    %Reaction coefficient degradation sub-cutaneous
        case 5  % low degradation, high blood
            parameters.De = 4e-6;        %Diffusion coefficient epidermis
            parameters.Dd = 5e-6;        %Diffusion coefficient dermis
            parameters.Db = 2e-6;        %Diffusion coefficient sub-cutaneous
            parameters.betaE = 0;        %Reaction coefficient blood epidermis
            parameters.betaD = 0.1;     %Reaction coefficient blood dermis
            parameters.betaB = 0.1;     %Reaction coefficient blood sub-cutaneous
            parameters.gammaE = 0.002;    %Reaction coefficient degradation epidermis
            parameters.gammaD = 0.002;    %Reaction coefficient degradation dermis
            parameters.gammaB = 0.002;    %Reaction coefficient degradation sub-cutaneous
        case 6  % high blood
            parameters.De = 4e-6;        %Diffusion coefficient epidermis
            parameters.Dd = 5e-6;        %Diffusion coefficient dermis
```

```
            parameters.Db = 2e-6;         %Diffusion coefficient sub-cutaneous
            parameters.betaE = 0;         %Reaction coefficient blood epidermis
            parameters.betaD = 0.1;      %Reaction coefficient blood dermis
            parameters.betaB = 0.1;      %Reaction coefficient blood sub-cutaneous
            parameters.gammaE = 0.02;     %Reaction coefficient degradation epidermis
            parameters.gammaD = 0.02;     %Reaction coefficient degradation dermis
            parameters.gammaB = 0.02;     %Reaction coefficient degradation sub-cutaneous
        case 7  % low blood
            parameters.De = 4e-6;         %Diffusion coefficient epidermis
            parameters.Dd = 5e-6;         %Diffusion coefficient dermis
            parameters.Db = 2e-6;         %Diffusion coefficient sub-cutaneous
            parameters.betaE = 0;         %Reaction coefficient blood epidermis
            parameters.betaD = 0.001;      %Reaction coefficient blood dermis
            parameters.betaB = 0.001;      %Reaction coefficient blood sub-cutaneous
            parameters.gammaE = 0.02;     %Reaction coefficient degradation epidermis
            parameters.gammaD = 0.02;     %Reaction coefficient degradation dermis
            parameters.gammaB = 0.02;     %Reaction coefficient degradation sub-cutaneous
        case 8  % low degradation
            parameters.De = 4e-6;         %Diffusion coefficient epidermis
            parameters.Dd = 5e-6;         %Diffusion coefficient dermis
            parameters.Db = 2e-6;         %Diffusion coefficient sub-cutaneous
            parameters.betaE = 0;         %Reaction coefficient blood epidermis
            parameters.betaD = 0.01;      %Reaction coefficient blood dermis
            parameters.betaB = 0.01;      %Reaction coefficient blood sub-cutaneous
            parameters.gammaE = 0.002;     %Reaction coefficient degradation epidermis
            parameters.gammaD = 0.002;     %Reaction coefficient degradation dermis
            parameters.gammaB = 0.002;     %Reaction coefficient degradation sub-cutaneous

    end
end
```

B APPENDIX: UNIT TESTS

Listing 21. Global mass matrix unit tests

```
%% Test 1: test symmetry of the matrix
% % Test that this matrix is symmetric
tol = 1e-14;
Ne=4;
parameters.selection = '1';
parameters.D = 2;
parameters.lambda = 2;
parameters.f = 1;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);

%global matrix and transpose with values above
GM = GlobalMassMatrix(Ne,msh,GQ,order);
transposedGM = transpose(GM);

diff = GM - transposedGM;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 2: test that the Global Matrix is evaluated correctly
tol = 1e-14;
Ne=4;
parameters.selection = '1';
parameters.D = 1;
parameters.lambda = 1;
parameters.f = 1;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

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)
```

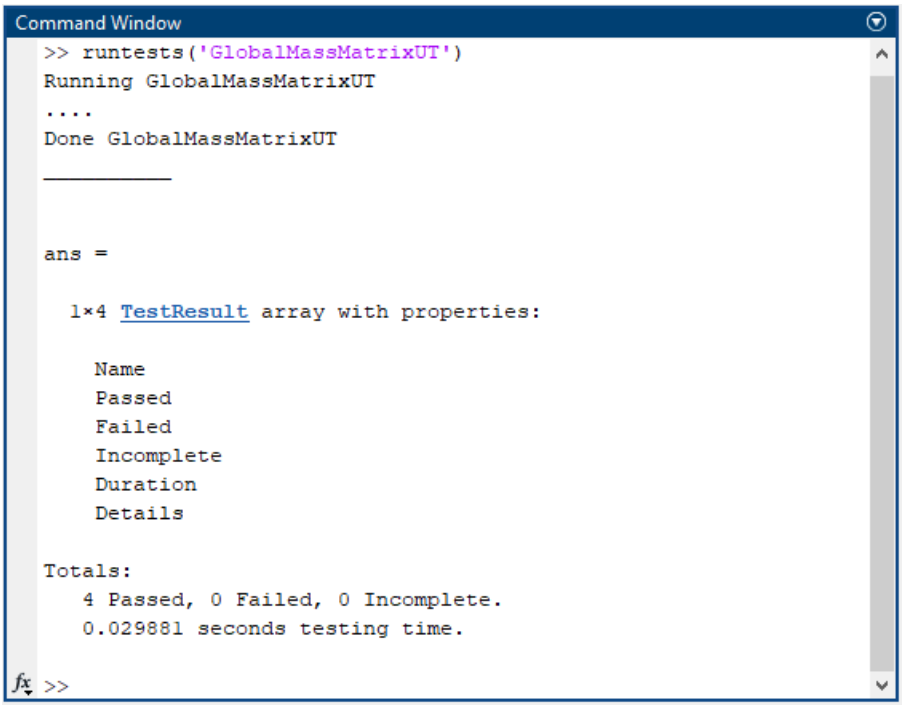

Figure 16. Global mass matrix unit tests results.

Listing 22. Global stiffness matrix unit tests

```
%% Test 1: test symmetry of the matrix
% % Test that this matrix is symmetric
tol = 1e-14;
Ne=4;
parameters.selection = '1';
parameters.D = 1;
parameters.lambda = 1;
parameters.f = 0;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);

%global matrix and transpose with values above
GM = GlobalStiffnessMatrix(Ne,msh,GQ,order);
transposedGM = transpose(GM);

diff = GM - transposedGM;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 2: test that the Global Matrix is evaluated correctly
tol = 1e-14;
Ne=4;
parameters.selection = '1';
parameters.D = 5;
parameters.lambda = 0;
parameters.f = 0;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);

%global matrix with values above
```

```
GM = GlobalStiffnessMatrix(Ne,msh,GQ,order);

%analytical global matrix
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];

diff = GM - analyticalGM;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 3: test Global Matrix Size
% Test that the size is equal to that of the nodes in the mesh
Ne=4;
parameters.selection = '1';
parameters.D = 5;
parameters.lambda = 5;
parameters.f = 0;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);

%global matrix with values above and size
GM = GlobalStiffnessMatrix(Ne,msh,GQ,order);
sizeGM = size(GM);

assert(sizeGM(1)==(msh.ngn)); %columns are same as node number
assert(sizeGM(2)==(msh.ngn)); %rows are same as node number

%% Test 4: test that values are only in the diagonals
% Tests that all values are only in the diagonals, and the rest are zeros.
tol = 1e-14;
Ne=4;
parameters.selection = '1';
parameters.D = 1;
parameters.lambda = 1;
parameters.f = 0;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);

%global matrix with values above
GM = GlobalStiffnessMatrix(Ne,msh,GQ,order);
%zeros same size as GM
zeroMatrix = zeros(msh.ngn);

mainGMDiagonal = diag(diag(GM));            %main diagonal
upperGMDiagonal = diag(diag(GM, 1), 1);     %upper diagonal
lowerGMDiagonal = diag(diag(GM, -1), -1);   %lower diagonal

%remove above diagonals from GM
emptyGM = GM - mainGMDiagonal - lowerGMDiagonal - upperGMDiagonal;

diff = emptyGM - zeroMatrix;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)
```

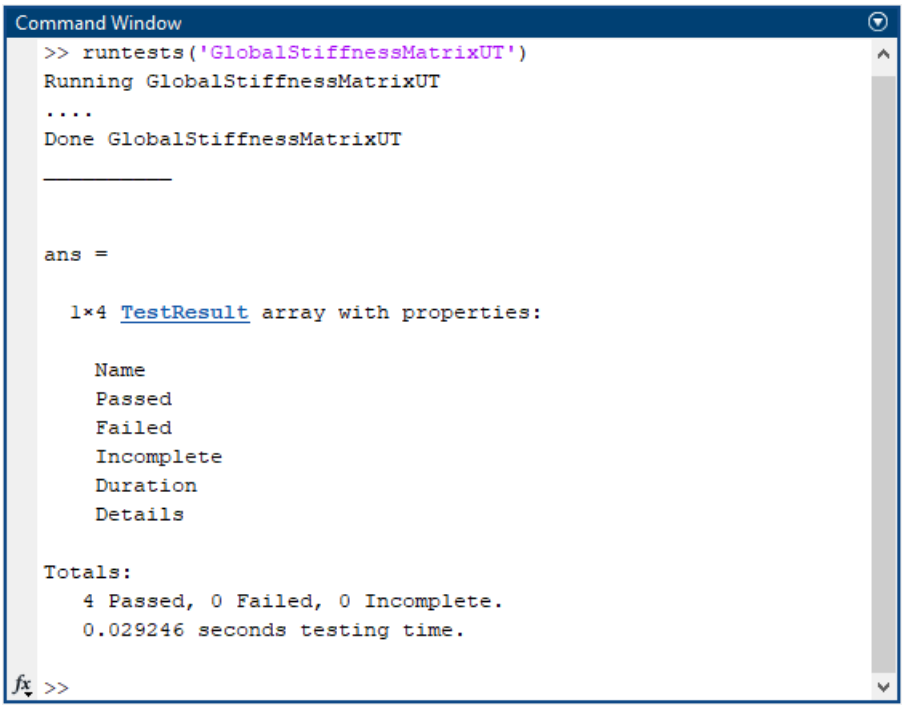

Figure 17. Global stiffness matrix unit tests results.

Listing 23. Global source vector unit tests

```
%% Test 1: test symmetry of the vector
% % Test that this vector is symmetric
tol = 1e-14;
Ne=4;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 0;
parameters.f = 1;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);

%global vector and flipped with values above
GV = GlobalSourceVector(Ne,msh,GQ,order);
flippedGV = flip(GV);

diff = GV - flippedGV;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 2: test that the Global Vector is evaluated correctly.
tol = 1e-14;
Ne=4;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 0;
parameters.f = 5;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);

%global vector with values above
```

```
GV = GlobalSourceVector(Ne,msh,GQ,order);

%analytical global vector
analyticalGV = [0.625; 1.25; 1.25; 1.25; 0.625];

diff = GV - analyticalGV;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 3: test Global Vector Size
% Test that the size is equal to that of the nodes in the mesh
Ne=4;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 0;
parameters.f = 5;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,Ne,order,parameters);

%global vector with values above and size
GV = GlobalSourceVector(Ne,msh,GQ,order);
GVsize = size(GV);

assert(GVsize(1)==(msh.ngn)); %columns are same as node number
assert(GVsize(2)==1);         %rows are 1
```

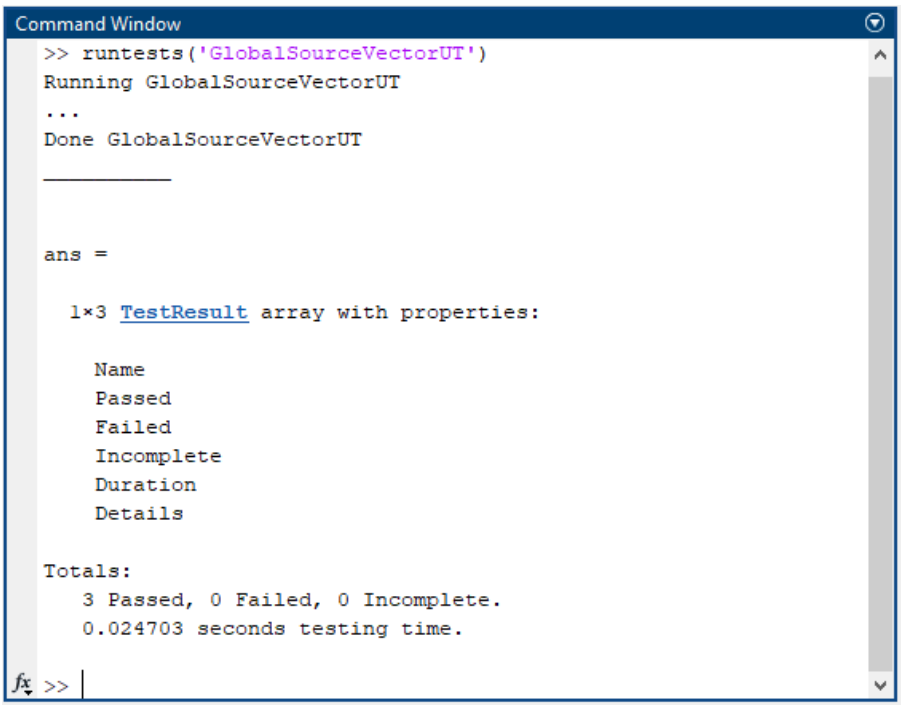

Figure 18. Global source vector unit tests results.

Listing 24. Local mass element unit tests

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

```
parameters.D = 2;
parameters.lambda = 2;
parameters.f = 1;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat = LocalMassElem(eN,msh,GQ,order);

assert(abs(elemat(1,2) - elemat(2,1)) <= tol)

%% Test 2: test 2 different elements of the same size produce same matrix
% % Test that for two elements of an equispaced mesh, the element matrices
% % calculated are the same using manual integration
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 5;
parameters.lambda = 5;
parameters.f = 1;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat1 = LocalMassElem(eN,msh,GQ,order);

eN=2;

elemat2 = LocalMassElem(eN,msh,GQ,order);

diff = elemat1 - elemat2;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 3: test that one matrix is evaluted correctly
% % Test that the element matrix is evaluated correctly using manual integration
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 5;
parameters.lambda = 5;
parameters.f = 1;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,2,order,parameters);

elemat1 = LocalMassElem(eN,msh,GQ,order);

elemat2 = [ 1/6 1/12; 1/12 1/6];
diff = elemat1 - elemat2; %calculate the difference between the two matrices
diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
assert(abs(diffnorm) <= tol)

%% Test 4: test symmetry of the matrix
% % Test that this matrix is symmetric using Gaussian Quadrature
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 2;
parameters.lambda = 2;
parameters.f = 1;
GQ.switch = '1';
GQ.npts = 2;
```

```
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat = LocalMassElem(eN,msh,GQ,order);

assert(abs(elemat(1,2) - elemat(2,1)) <= tol)

%% Test 5: test 2 different elements of the same size produce same matrix
% % Test that for two elements of an equispaced mesh, the element matrices
% % calculated are the same using Gaussian Quadrature
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 5;
parameters.lambda = 5;
parameters.f = 1;
GQ.switch = '1';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat1 = LocalMassElem(eN,msh,GQ,order);

eN=2;

elemat2 = LocalMassElem(eN,msh,GQ,order);

diff = elemat1 - elemat2;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 6: test that one matrix is evaluted correctly
% % Test that the element matrix is evaluated correctly using Gaussian Quadrature
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 5;
parameters.lambda = 5;
parameters.f = 1;
GQ.switch = '1';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,2,order,parameters);

elemat1 = LocalMassElem(eN,msh,GQ,order);

elemat2 = [ 1/6 1/12; 1/12 1/6];
diff = elemat1 - elemat2; %calculate the difference between the two matrices
diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
assert(abs(diffnorm) <= tol)
```

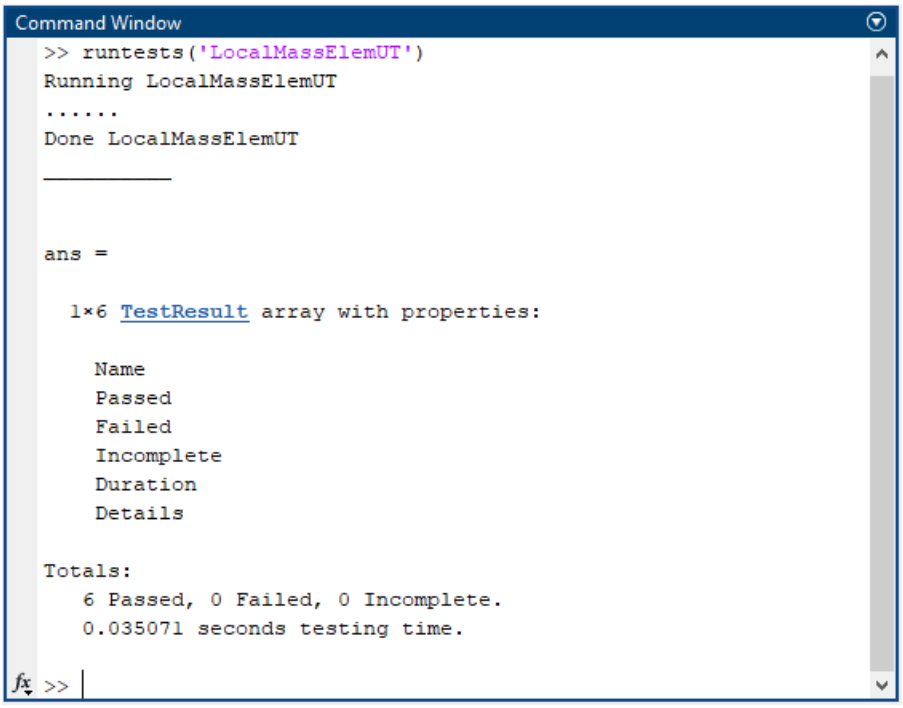

Figure 19. Local mass element unit tests results.

Listing 25. Diffusion local element matrix unit tests

```
%% Test 1: test symmetry of the matrix
% % Test that this matrix is symmetric using manual integration
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 2;
parameters.lambda = 0;
parameters.f = 0;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat = LEMdiffusion(eN,msh,GQ,order);

assert(abs(elemat(1,2) - elemat(2,1)) <= tol)

%% Test 2: test 2 different elements of the same size produce same matrix
% % Test that for two elements of an equispaced mesh, the element matrices
% % calculated are the same using manual integration
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 5;
parameters.lambda = 0;
parameters.f = 0;
GQ.switch = '0';
GQ.npts = 2;
order = 1;
msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat1 = LEMdiffusion(eN,msh,GQ,order);

eN=2;

```

```
elemat2 = LEMdiffusion(eN,msh,GQ,order);

diff = elemat1 - elemat2;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 3: test that one matrix is evaluted correctly
% % Test that the element matrix is evaluated correctly using manual integration
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 2.5;
parameters.lambda = 0;
parameters.f = 0;
GQ.switch = '0';
GQ.npts = 2;
order = 1;
msh = OneDimLinearMeshGen(0,1,3,order,parameters);

elemat1 = LEMdiffusion(eN,msh,GQ,order);

elemat2 = [ 7.5 -7.5; -7.5 7.5];
diff = elemat1 - elemat2; %calculate the difference between the two matrices
diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
assert(abs(diffnorm) <= tol)

%% Test 4: test symmetry of the matrix
% % Test that this matrix is symmetric using Gaussian Quadrature
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 2;
parameters.lambda = 0;
parameters.f = 0;
GQ.switch = '1';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat = LEMdiffusion(eN,msh,GQ,order);

assert(abs(elemat(1,2) - elemat(2,1)) <= tol)

%% Test 5: test 2 different elements of the same size produce same matrix
% % Test that for two elements of an equispaced mesh, the element matrices
% % calculated are the same using Gaussian Quadrature
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 5;
parameters.lambda = 0;
parameters.f = 0;
GQ.switch = '1';
GQ.npts = 2;
order = 1;
msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat1 = LEMdiffusion(eN,msh,GQ,order);

eN=2;

elemat2 = LEMdiffusion(eN,msh,GQ,order);

diff = elemat1 - elemat2;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 6: test that one matrix is evaluted correctly
```

```
% % Test that the element matrix is evaluated correctly using Gaussian
% % Quadrature
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 2.5;
parameters.lambda = 0;
parameters.f = 0;
GQ.switch = '1';
GQ.npts = 2;
order = 1;
msh = OneDimLinearMeshGen(0,1,3,order,parameters);

elemat1 = LEMdiffusion(eN,msh,GQ,order);

elemat2 = [ 7.5 -7.5; -7.5 7.5];
diff = elemat1 - elemat2; %calculate the difference between the two matrices
diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
assert(abs(diffnorm) <= tol)
```

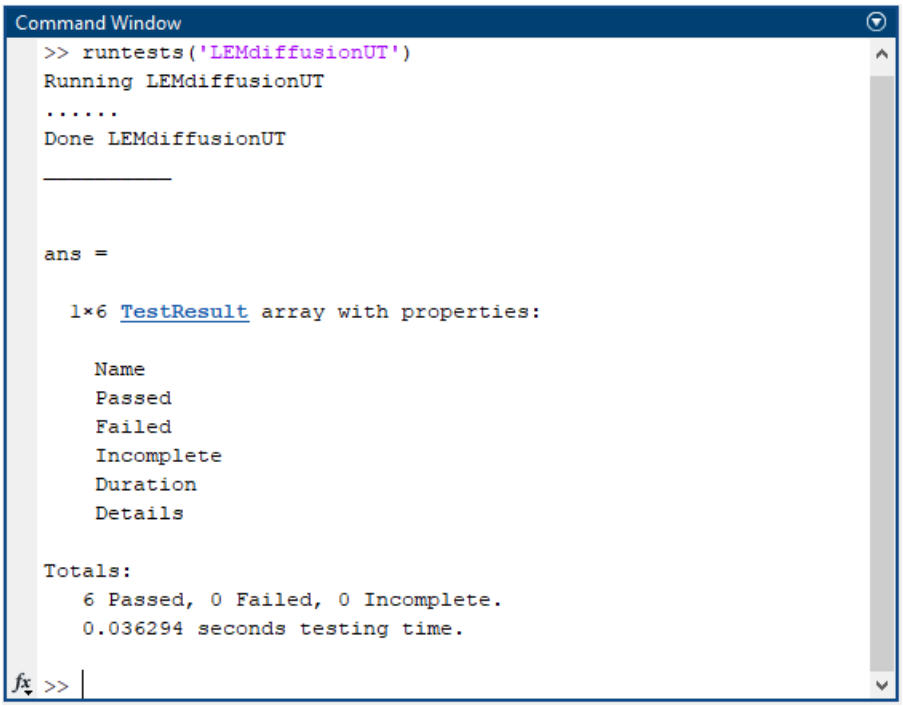

Figure 20. Diffusion local element matrix unit tests results.

Listing 26. Reaction local element matrix unit tests

```
%% Test 1: test symmetry of the matrix
% % Test that this matrix is symmetric using manual integration
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 2;
parameters.f = 0;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

```

```
elemat = LEMreaction(eN,msh,GQ,order);

assert(abs(elemat(1,2) - elemat(2,1)) <= tol)

%% Test 2: test 2 different elements of the same size produce same matrix
% % Test that for two elements of an equispaced mesh, the element matrices
% % calculated are the same using manual integration
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 5;
parameters.f = 0;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat1 = LEMreaction(eN,msh,GQ,order);

eN=2;

elemat2 = LEMreaction(eN,msh,GQ,order);

diff = elemat1 - elemat2;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 3: test that one matrix is evaluted correctly
% % Test that the element matrix is evaluated correctly using manual integration
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 5;
parameters.f = 0;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,2,order,parameters);

elemat1 = LEMreaction(eN,msh,GQ,order);

elemat2 = [ 5/6 5/12; 5/12 5/6];
diff = elemat1 - elemat2; %calculate the difference between the two matrices
diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
assert(abs(diffnorm) <= tol)

%% Test 4: test symmetry of the matrix
% % Test that this matrix is symmetric using Gaussian Quadrature
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 2;
parameters.f = 0;
GQ.switch = '1';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat = LEMreaction(eN,msh,GQ,order);

assert(abs(elemat(1,2) - elemat(2,1)) <= tol)

%% Test 5: test 2 different elements of the same size produce same matrix
```

```
% % Test that for two elements of an equispaced mesh, the element matrices
% % calculated are the same using Gaussian Quadrature
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 5;
parameters.f = 0;
GQ.switch = '1';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat1 = LEMreaction(eN,msh,GQ,order);

eN=2;

elemat2 = LEMreaction(eN,msh,GQ,order);

diff = elemat1 - elemat2;
diffnorm = sum(sum(diff.*diff));
assert(abs(diffnorm) <= tol)

%% Test 6: test that one matrix is evaluted correctly
% % Test that the element matrix is evaluated correctly using Gaussian Quadrature
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 5;
parameters.f = 0;
GQ.switch = '1';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,2,order,parameters);

elemat1 = LEMreaction(eN,msh,GQ,order);

elemat2 = [ 5/6 5/12; 5/12 5/6];
diff = elemat1 - elemat2; %calculate the difference between the two matrices
diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
assert(abs(diffnorm) <= tol)
```

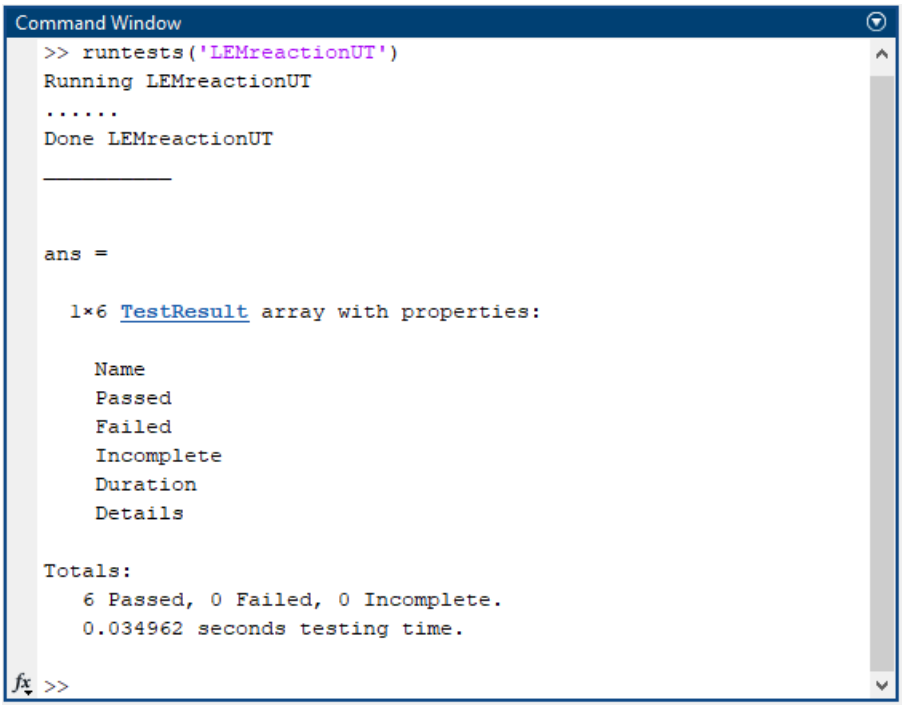

Figure 21. Reaction local element matrix unit tests results.

Listing 27. Local source element unit tests

```
%% Test 1: test symmetry of the vector
% % Test that this vector is symmetric using manual integration
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 0;
parameters.f = 1;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat = LocalSourceElem(eN,msh,GQ,order);

assert(abs(elemat(1) - elemat(2)) <= tol)

%% Test 2: test 2 different elements of the same size produce same vector
% % Test that for two elements of an equispaced mesh, the element matrices
% % calculated are the same using manual integration
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 0;
parameters.f = 1;
GQ.switch = '0';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,10,order,parameters);

elemat1 = LocalSourceElem(eN,msh,GQ,order);

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 evaluted 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));
```

```
assert(abs(diffnorm) <= tol)

%% Test 6: test that one vector is evaluted correctly
% % Test that the element vector is evaluated correctly using Gaussian Quadrature
tol = 1e-14;
eN=1;
parameters.selection = '1';
parameters.D = 0;
parameters.lambda = 0;
parameters.f = 1;
GQ.switch = '1';
GQ.npts = 2;
order = 1;

msh = OneDimLinearMeshGen(0,1,4,order,parameters);

elemat1 = LocalSourceElem(eN,msh,GQ,order);

elemat2 = [ 0.1250; 0.1250 ];
diff = elemat1 - elemat2;  %calculate the difference between the two matrices
diffnorm = sum(sum(diff.*diff)); %calculates the total squared error between the matrices
assert(abs(diffnorm) <= tol)
```

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


ans =

  1×6 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.