Math 179 Project

Alison Peard, PID: A15759408

Spring 2019

1 Introduction

Reaction-diffusion equations can be used to model various physical phenomena in a variety of circumstances and, in particular, to address emerging ecological problems and trends, such as the effects of climate change and deforestation on biodiversity and the survival of species.

This project looks into methods of solving linear and nonlinear reaction-diffusion (particularly Fisher's) partial differential equations $\mathbb{R}$. The following cases will be addressed:

1. Solving linear reaction-diffusion equation via Crank-Nicolson finite difference methods
2. Solving Burger's equation via Crank-Nicolson method and observing the error relative to the exact solution
3. Solving nonlinear Fisher equations via backward-Euler/Crank-Nicolson methods and the multi-step Newton method
4. Finite element method for the linear reaction-diffusion equation in (1)

2 Fishers Equation

Fishers equations are a classic example of reaction-diffusion equations and can be used to model a wide variety of situations in ecology, physiology, combustion, crystallization, plasma physics, and in general phase transition problems *[Wikipedia]*. The general form of Fishers equation in $\mathbb{R}^n$ is given by

$$u_t = D\Delta u + f(u) \tag{1}$$

with $D > 0$ constant. The reaction part of the equation is supplied by f, and D is the diffusion coefficient.

If homogeneous Neumann boundary conditions are used, the constant,
or equilibrium state $u(x,t) \equiv C$ is a solution whenever $f(C) = 0$. The equilibrium state turns out to be stable if $f'(C) < 0$, meaning that nearby solutions

tend toward the equilibrium state. *[Sauer's Numerical Analysis 2nd Edition]*

3 FDM for Linear Reaction-Diffusion Equations

3.1 Crank-Nicolson Method

Crank–Nicolson is unconditionally stable and second-order accurate in both space and time. *[Sauer ch.8]*

Consider the linear reaction-diffusion equation with proportional growth and Dirichlet boundary conditions in $\mathbb{R}$ *(parabolic)*:

$$\begin{cases} u_t = Du_{xx} + Cu, x \in \Omega = [0,1] \\ u(x,0) = \sin^2(\frac{\pi}{L}), \forall\ 0 \le x \le L \\ u(0,t) = u(L,t) = 0, \forall t \ge 0 \end{cases}$$

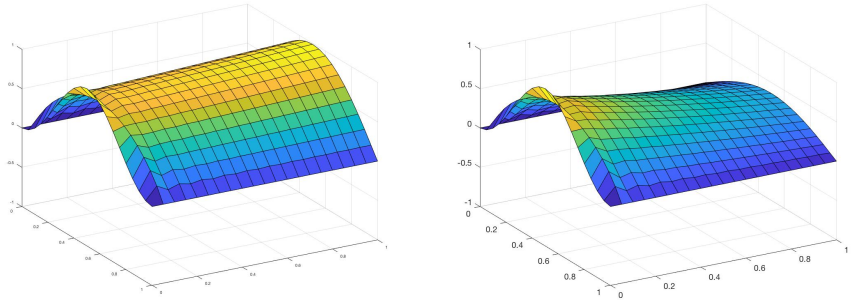
This is perhaps the simplest possible example of a reaction-diffusion equation and a simplified version of Fisher's equation. The diffusion term Du_{xx} causes the population to spread along the x-direction, while the reaction term Cu contributes population growth of rate C. Whether the population survives or proceeds toward extinction depends on the competition between the diffusion parameter D, the growth rate C, and the patch size L. *[Sauer ch.8.5]*

The model survives when $C \ge \frac{\pi^2 D}{L^2}$, or when $C \ge \pi^2$ if $C = L = 1$.

Solving this using the Crank-Nicolson method as described in *Sauer p.390*:

```
% input: space interval [xl,xr], time interval [yb,yt],
% number of space steps M, number of time steps N
% output: solution w
% Example usage: w=fisher_cn(0,1,0,1,20,20)
function w=fisher_cn(xl,xr,yb,yt,M,N)
D=1; C=10; L=xr-xl;                      % diffusion coefficient L = (xr-xl);
f=@(x) sin((pi/L)*x).*sin((pi/L).*x);
l=@(t) 0*t;
r=@(t) 0*t;
h=(xr-xl)/M;k=(yt-yb)/N;            % step sizes
sigma=D*k/(h*h); m=M-1; n=N;
a=diag(2-k*C+2*sigma*ones(m,1))+diag(-sigma*ones(m-1,1),1);
a=a+diag(-sigma*ones(m-1,1),-1);    % define tridiagonal matrix a
b=diag(2+k*C-2*sigma*ones(m,1))+diag(sigma*ones(m-1,1),1);
b=b+diag(sigma*ones(m-1,1),-1);     % define tridiagonal matrix b
lside=l(yb+(0:n)*k); rside=r(yb+(0:n)*k);
w(:,1)=f(xl+(1:m)*h)';              % initial conditions
```

C10.jpg C9.jpg



h=k=0.05

(a) Solution plot for $C = 10$ (b) Solution plot for $C = 9$

```
for j=1:n
    sides=[lside(j)+lside(j+1);zeros(m-2,1);rside(j)+rside(j+1)];
    w(:,j+1)=a\(b*w(:,j)+sigma*sides);
end
w=[lside;w;rside];
x=xl+(0:M)*h;t=yb+(0:N)*k;
surf(x,t,w');
```

It is clear that once C crosses below the value of π^2 the solution begins decaying towards zero and the population approaches extinction.
Fixing $C = D = 1$, one can also show the minimum value for L for which this population will survive is $L = \pi$. This solution will be compared to the solution using the finite element method later on.

4 FDM for Nonlinear Fisher Equations

We will solve two nonlinear Fisher equations using Backward-Euler or Crank-Nicolson discretization with the multi-step Newton solver. Here Fisher's equation with homogeneous Neumann boundary conditions will be given by:

(a) $\begin{cases} u_t = Du_{xx} + Cu(1-u), x \in \Omega = [0,1] \\ u(x,0) = \frac{1}{2} + \frac{1}{2}\cos(\pi x)), \forall\ 0 \le x \le L \\ u_x(0,t) = u_x(L,t) = 0, \forall t \ge 0 \end{cases}$ (b) $\begin{cases} u_t = Du_{xx} + Cu(u-1)(2-u), x \in \Omega = [0,1] \\ u(x,0) = \frac{1}{2} + \frac{1}{2}\cos(\pi x)), \forall\ 0 \le x \le L \\ u_x(0,t) = u_x(L,t) = 0, \forall t \ge 0 \end{cases}$

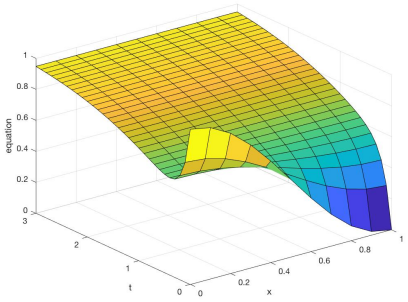
For (a) the stable equilibrium position is $u = 1$ and for (b) the stable equilibrium position is at $u = 0$ so we expect the solution in (a) to approach 1 and the solution in (b) to approach 0.

4.1 Backward-Euler

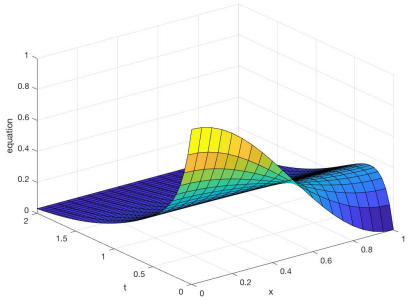
The MATLAB code for (a) with initial condition $u(x,0) = \sin(\pi x)$ by this method is provided in *Sauer* and altering it slightly yields the solution in the figure below.
To compute (b):

```
% Fisher from Computer Problem 8.4
% f(u) = u(u-1)(2-u) I think
%Example usage: w=fisher8_4(0,1,0,2,20,40)
function w = fisher8_4(xl,xr,tb,te,M,N)
    D=1;
    f=@(x)  0.5 + 0.5*cos(pi.*x);
    h = (xr-xl)/M;k= (te-tb)/N;n=N;m=M+1;
    lambda = (D*k)/(h*h);
    w(:,1) = f(xl + (0:M)*h)';
    w1=w;
    %Define Jacobian
    for j = 1:n
        for it=1:3
            DF1=zeros(m,m);DF2=zeros(m,m);DF3=zeros(m,m);
            DF1 = diag(-lambda*ones(m-1,1),-1)+diag(-lambda*ones(m-1,1),1);
            DF1 = DF1 + diag(1 + 2*lambda + 2*k*ones(m,1));
            DF2 = diag([0;-6*k*w1(2:m-1);0]);
            DF3 = diag([0;3*k*w1(2:m-1).*w1(2:m-1);0]);
            DF = DF1+DF2+DF3;
            F = (DF1+DF2/2+DF3/3)*w1 - w(:,j);
            DF(1,:)=[-3 4 -1 zeros(1,m-3)];F(1)=DF(1,:)*w1; % Neumann Conditions
            DF(m,:)=[zeros(1,m-3) -1 4 -3];F(m)=DF(m,:)*w1; % Neumann Conditions
            w1 = w1-DF\F;
        end
        w(:,j+1)=w1;
    end
    x=xl+(0:M)*h;t=tb+(0:n)*k;

    surf(x,t,w')
    xlabel('x')
    ylabel('t')
    zlabel('equation')
```

1D BE (a).jpg



1D BE (b).jpg



4.2 Crank-Nicolson for Burger's Equation

The Crank–Nicolson Method is a combination of the explicit and implicit methods, is unconditionally stable, and has error $O(h^2) + O(k^2)$. The formulas are slightly more complicated, but worth the trouble because of the increased accuracy and guaranteed stability. *[Sauer p. 385]* I will first apply Crank-Nicolson discretization to Burger's equation to allow us to look at the error in the Crank-Nicolson method. Burger's equation with Dirichlet boundary conditions in $\mathbb{R}$ is

$$\begin{cases} u_t + uu_x = Du_{xx} \\ u(x,0) = \frac{2D\beta \sin \pi x}{\alpha + \beta \cos \pi x}, & 0 \leq x \leq 1 \\ u((0,t) = u(1,t) = 0, & \forall t \geq 0 \end{cases}$$

with exact solution

$$u(x,t) = \frac{2D\beta\pi e^{-D\pi^2 t} \sin \pi x}{\alpha + \beta e^{-D\pi^2 t} \cos \pi x}$$

As in *Sauer* we set $\alpha = 5$, $\beta = 4$.

The Crank-Nicolson method discretizes partial differential equations,

$$u_t = F(u, x, t, u_x, u_{xx})$$

according to,

$$\frac{U_i^n - U_i^{n-1}}{\Delta t} = \frac{1}{2}\left(F_i^n(u, x, t, u_x, u_{xx}) + F_i^{n-1}(u, x, t, u_x, u_{xx})\right)$$

For Burger's equation with Where $\lambda = \frac{Dk}{h^2}$ this yields,

$$\begin{aligned} &-\left(\frac{\lambda}{2}+\frac{k}{2h}U_j^n\right)U_{j-1}^n+\left(1+\lambda+\frac{k}{2h}\left(U_j^n-U_{j-1}^n\right)\right)U_j^n-\frac{\lambda}{2}U_{j+1}^n \\ &=\left(\frac{\lambda}{2}+\frac{k}{2h}U_j^{n-1}\right)U_{j-1}^{n-1}+\left(1-\lambda-\frac{k}{2h}\left(U_j^{n-1}-U_{j-1}^{n-1}\right)\right)U_j^{n-1}+\frac{\lambda}{2}U_{j+1}^{n-1} \end{aligned} \tag{2}$$

Giving us the matrices below (with indexing from 1 since this will be coded in `MATLAB`) (slightly unsure if 1's for z_1 and z_m should be in A or B),

$$\mathrm{A}=\begin{bmatrix} 1 & 0 & \cdots & \\ -\left(\frac{\lambda}{2}+\frac{k}{2h}z_2\right) & 1+\lambda+\frac{k}{2h}\left(z_2-z_1\right) & -\frac{\lambda}{2} & \vdots \\ \ddots & \ddots & \ddots & \\ & -\left(\frac{\lambda}{2}+\frac{k}{2h}z_{m-1}\right) & 1+\lambda+\frac{k}{2h}\left(z_{m-1}-z_{m-2}\right) & -\frac{\lambda}{2} \\ & \cdots & 0 & 1 \end{bmatrix}$$

$$\mathrm{B}=\begin{bmatrix} 0 & \cdots & & \\ \left(\frac{\lambda}{2}+\frac{k}{2h}z_2\right) & 1-\lambda-\frac{k}{2h}\left(z_2-z_1\right) & \frac{\lambda}{2} & \\ \ddots & \ddots & \ddots & \\ & \left(\frac{\lambda}{2}+\frac{k}{2h}z_{m-1}\right) & 1-\lambda-\frac{k}{2h}\left(z_{m-1}-z_{m-2}\right) & \frac{\lambda}{2} \\ & & \cdots & 0 \end{bmatrix}$$

At time-step n we try to find the roots of the system of equations below for the m unknowns $z_1, .., z_m$,

$$F\left(\bar{z}^n\right)=A\bar{z}^n-B\bar{U}^{n-1} \tag{3}$$

To apply Newton's Multivariate method we calculate the Jacobian of F which is the simply the Jacobian of $A\bar{z}^n$ with the tridiagonal form,

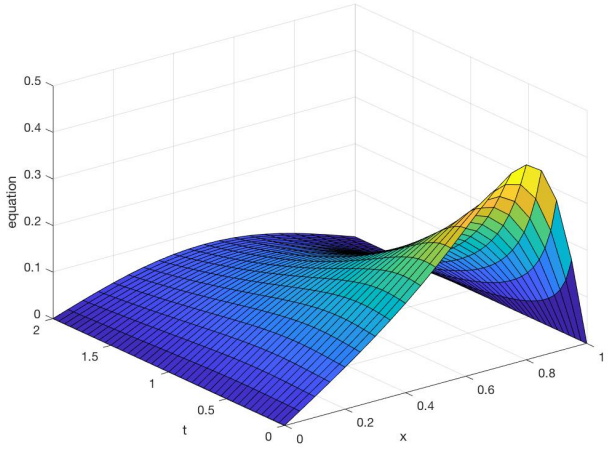
$$\mathrm{DF}=\begin{bmatrix} 1 & 0 & \cdots & \\ -\left(\frac{\lambda}{2}+\frac{k}{2h}z_2\right) & 1+\lambda-\frac{k}{h}z_2+\frac{k}{2h}z_1 & -\frac{\lambda}{2} & \vdots \\ \ddots & \ddots & \ddots & \\ & -\left(\frac{\lambda}{2}+\frac{k}{2h}z_{m-1}\right) & 1+\lambda+\frac{k}{h}z_{m-1}\frac{k}{2h}z_{m-2} & -\frac{\lambda}{2} \\ & \cdots & 0 & 1 \end{bmatrix}$$

Solve for $z^n = \bar{u}^n$ by,

$$\bar{z}^n = \bar{z}^{n-1} - DF\left(\bar{z}^{n-1}\right)^{-1} F\left(\bar{z}^{n-1}\right).$$

Using Lemma 8.11 from *Sauer p.420* the Jacobian `DF` can be split into matrices of different degrees of `F`'s terms. `DF1` collects derivatives of degree 1 terms of `F` etc.
The solution I got when trying to run the code by this method didn't converge to anything reasonable; so in my code below I have taken the simpler approach of defining `DF`, `A` and `B` separately, ideally this would have worked by the Lemma 8.11 method but that is something I will have to look into at a later date. Other options for improving this code would be replacing the initial guess `w1` in the line `F = A*w1 - B*w1` with `z1` calculated by explicit Backward-Euler iteration.

```
% Burger's Equation (\textit{Sauer p.419}) solved by the Crank-Nicolson Method
% Example usage: w=burgers_cn(0,1,0,2,20,40)
function w = burgers_cn(xr,xl,tb,te,M,N)
    alf=5;bet=4;D=.05;
    f=@(x) 2*D*bet*pi*sin(pi*x)./(alf+bet*cos(pi*x));
    l=@(t) 0*t;
    r=@(t) 0*t;
    h=(xr-xl)/M; k=(te-tb)/N; m=M+1; n=N; % => m-2=M-1 & m-2=M-3
    x=xl+(0:M)*h;t=tb+(0:n)*k;
    lambda=D*k/(h*h);
    w(:,1)=f(xl+(0:M)*h)'; % creates inital vector of coefficients
    w1=w;

    for j=1:n
        for it=1:3
            A = diag([-(lambda/2)*ones(m-2,1);0],-1)...
            + diag([0;(-lambda/2).*ones(m-2,1)],1);
            A = A + diag([0;(1+lambda)*ones(m-2,1);0]);
            A = A + diag([-k*w1(2:m-1)/(2*h);0],-1);
            A = A - diag([0;k*w1(2:m-1)/(2*h);0]);
            A = A + diag([0;k*w1(1:m-2)/(2*h);0]);
            A(1,:) = [1 zeros(1,m-1)];
            A(m,:) = [zeros(1,m-1) 1];

            B = diag([(lambda/2)*ones(m-2,1);0],-1)...
            + diag([0;(lambda/2).*ones(m-2,1)],1);
            B = B + diag([0;(1-lambda)*ones(m-2,1);0]);
            B = B + diag([k*w1(2:m-1)/(2*h);0],-1);
            B = B - diag([0;k*w1(2:m-1)/(2*h);0]);
            B = B - diag([0;k*w1(1:m-2)/(2*h);0]);
```



(a) Burgers Crank-Nicolson

```
            DF1 = eye(m);
            DF1 = DF1 + diag([0;1+lambda*ones(m-2,1);0]);
            DF1 = DF1 - diag([(lambda/2)*ones(m-2,1);0],-1);
            DF1 = DF1 - diag([0;(lambda/2)*ones(m-2,1)],1);

            DF2=-diag([-k*w1(2:m-1)/(2*h);0],-1);
            DF2=DF2 - diag([0;(k/h)*w1(2:m-1);0])+diag([0;k*w1(1:(m-2))/(2*h);0]);

            DF = DF1+DF2;
            %F = (DF1+DF2/2)*w1 - B*w1;
            F = A*w1 - B*w1;
            w1=w1-DF\F;
    end
    w(:,j+1)=w1;
    end
surf(x,t,w')
xlabel('x')
ylabel('t')
zlabel('equation')
% 3-D plot of solution w
```

Using `w=burgers cn(0,1,0,1,25,25)`, `w=burgers cn(0,1,0,1,50,50)` and `w=burgers cn(0,1,0,1,100,100)` and `w(13,end)` etc. we calculate and observe the errors,

h	k	u(0.5,1)	w(0.5,1)	error
0.01	0.04	0.153435	0.1232	0.0302
0.01	0.02	0.153435	0.1377	0.0157
0.01	0.01	0.153435	0.1440	0.0094

Though the plot of the solution is comparable to the backward-Euler plot for the same equation given in *Sauer*, the convergence appears second-order, which is not what I was expecting and something I will have to look into later.

4.3 Crank-Nicolson for Fisher Equations

By the same method, the matrices for the Fisher equation (a) with $C = 1$ and homogeneous Neumann boundary conditions (as derived from the second-order formula for the first derivative in *Sauer ch. 5*, are given by:

$$\mathrm{A} = \begin{bmatrix} -3 & 4 & -1 & \\ -\frac{\lambda}{2} & 1+\lambda-\frac{Ck}{2}(1-z_2) & -\frac{\lambda}{2} & \vdots \\ \ddots & \ddots & \ddots & \\ & -\frac{\lambda}{2} & 1+\lambda-\frac{Ck}{h}(1-z_{m-1}) & -\frac{\lambda}{2} \\ & -1 & 4 & -3 \end{bmatrix}$$

$$\mathrm{B} = \begin{bmatrix} 0 & 0 & \cdots & \\ \frac{\lambda}{2} & 1-\lambda+\frac{Ck}{2}(1-z_2) & \frac{\lambda}{2} & \vdots \\ \ddots & \ddots & \ddots & \\ & \frac{\lambda}{2} & 1-\lambda+\frac{Ck}{2}(1-z_{m-1}) & \frac{\lambda}{2} \\ & \cdots & 0 & 0 \end{bmatrix}$$

$$\mathrm{DF} = \begin{bmatrix} -3 & 4 & -1 & \\ -\frac{\lambda}{2} & 1+\lambda-\frac{Ck}{2}+Ckz_2 & -\frac{\lambda}{2} & \vdots \\ \ddots & \ddots & \ddots & \\ & -\frac{\lambda}{2} & 1+\lambda-\frac{Ck}{2}+Ckz_{m-1} & -\frac{\lambda}{2} \\ & -1 & 4 & -3 \end{bmatrix}$$

In `MATLAB`,

```
% Fisher's Equation solved by the Crank-Nicolson Method
% where f(u) = u(1-u)
```

```
% should have stable equib at u=1
% Example usage: w=fisher_cn(0,1,0,1,50,100)
function w = fisher_cn(xl,xr,tb,te,M,N)
    D=1;
    f=@(x)  0.5 + 0.5*cos(pi.*x);
    l=@(t) 0*t;
    r=@(t) 0*t;
    h=(xr-xl)/M; k=(te-tb)/N; m=M+1; n=N;
    x=xl+(0:M)*h; t=tb+(0:n)*k;
    lambda=D*k/(h*h);
    w(:,1)=f(xl+(0:M)*h)'; % creates inital vector of coefficients
    w1=w;

    for j=1:n
        for it=1:3

          A = diag([-(lambda/2)*ones(m-2,1);0],-1)...
          + diag([0;(-lambda/2).*ones(m-2,1)],1);
          A = A + diag([0;(1+lambda)*ones(m-2,1);0]);
          A = A - diag([0;k*(1-w1(2:m-1))/2;0]);
          A(1,:)=[-3 4 -1 zeros(1,m-3)]; % Neumann boundary conditions
          A(m,:)=[zeros(1,m-3) -1 4 -3]; % Neumann boundary conditions

          B = diag([(lambda/2)*ones(m-2,1);0],-1)...
          + diag([0;(lambda/2).*ones(m-2,1)],1);
          B = B + diag([0;(1 - lambda)*ones(m-2,1);0]);
          B = B + diag([0;k*(1-w1(2:m-1))/2;0]);

          DF1 = zeros(m,m);
          DF1 = DF1 - diag([(lambda/2)*ones(m-2,1);0],-1);
          DF1 = DF1 - diag([0;(lambda/2)*ones(m-2,1)],1);
          DF1 = DF1 + diag([0;1 + lambda*ones(m-2,1);0]);
          DF1(1,:)=[-3 4 -1 zeros(1,m-3)]; % Neumann boundary conditions
          DF1(m,:)=[zeros(1,m-3) -1 4 -3]; % Neumann boundary conditions

          DF2=diag([0;k*w1(2:m-1);0])...
              -diag([0;(k/2)*ones(m-2,1);0]);

          DF = DF1+DF2;
          %F = (DF1+DF2/2)*w1 - B*w1;
          F = A*w1 - B*w1;
          w1=w1-DF\F;
        end
w(:,j+1)=w1;
    end
```

```
% 3-D plot of solution w
surf(x,t,w')
```

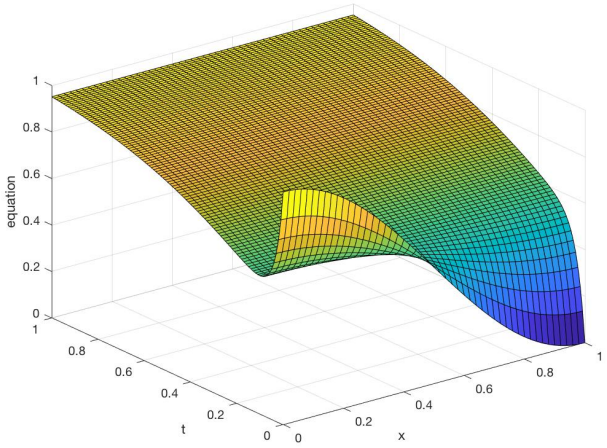


Figure 4: Fisher equation (a) with C=1, h=0.02, k=0.01

5 Finite Element Method

Looking at the linear reaction-diffusion equation from 3.1 with Dirichlet boundary conditions:

$$\begin{cases} u_t = Du_{xx} + Cu, x \in \Omega = [0,1] \\ u(x,0) = \sin^2(\frac{\pi}{L}), \forall\ 0 \leq x \leq L = 1 \\ u(0,t) = u(L,t) = 0, \forall t \geq 0 \end{cases}$$

We derive the variational form by multiplying by a smooth function v which also vanishes on the boundary and then integrating over the space Ω. Solutions are trivially C^∞ so there is no issue about regularity.
`https://web.ma.utexas.edu/mediawiki/index.php/Semilinear_equations`

$$\int_\Omega u_t v = D \int_\Omega u_{xx} v + \int_\Omega Cuv \tag{4}$$

Using integration by parts and Green's formula we arrive at

$$\int_\Omega u_t v = D\int_\Gamma \frac{\partial u}{\partial n} v ds - \int_\Omega u'v' + C\int_\Omega uv \tag{5}$$

with v also vanishing on the boundary, yielding,

$$\begin{aligned} &\int_\Omega u_t v + \int_\Omega (Du'v' - Cuv) = 0 \\ &(u_t, v) + a(u, v) = f(v) = 0,\ \forall v\ ? \in H_0^1 \end{aligned} \tag{6}$$

with $a(u, v)$ the bilinear form with $a = D$, $b = 0$, $c = -C$.
We begin discretizing by setting $u = \sum_{i=0}^{n+1} \alpha_i(t)\phi_i(x)$ and $v_i(x) = \phi_i(x)$. Where $\phi_i(x)$ are piecewise-linear B-splines, i.e. $\phi_i(x_j) = \delta_{ij}$. This gives us,

$$\begin{aligned} &\alpha'(t)\sum_i (\phi_i, \phi_j) + \alpha(t)\sum_i a(\phi_i, \phi_j) = 0 \\ &B\alpha'(t) + A\alpha(t) = 0 \end{aligned} \tag{7}$$

with stiffness matrix $A = (a_{kj})$ such that,

$$\begin{aligned} (a_{kj}) = a(\phi_j, \phi_k) &= \int_\Omega D\phi_j'\phi_k' - C\phi_j\phi_k dx \\ &= \begin{cases} D\frac{2}{h} - C\frac{2h}{3}, & j = k \\ -(\frac{D}{h} + C\frac{h}{6}), & j = k-1, j = k+1 \\ 0, & \text{else} \end{cases} \end{aligned}$$

and mass matrix $B = (b_{kj})$ such that,

$$\begin{aligned} (b_{kj}) = (\phi_j, \phi_k) &= \int_\Omega \phi_j\phi_k dx \\ &= \begin{cases} \frac{2}{3}h, & j = k \\ \frac{h}{6}, & j = k+1, j = k-1 \\ 0, & \text{else} \end{cases} \end{aligned}$$

Applying backward-Euler to the semidiscrete solution,

$$\begin{aligned} &B\alpha^n + kA\alpha^n = B\alpha^{n-1} \\ &\alpha^n = (B + kA)^{-1}B\alpha^{n-1} \end{aligned} \tag{8}$$

With $h = [x_i - x_{i-1}]$ and $k = [t_i - t_{i-1}]$ and using the identities provided for piecewise-linear B-splines in *Sauer p.370*, the matrices A and B are:

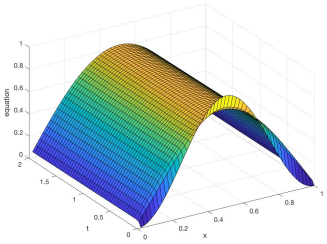
$$\mathrm{A} = \begin{bmatrix} D\frac{2}{h} - C\frac{2h}{3} & -(\frac{D}{h} + C\frac{h}{6}) & 0 & 0 \\ -(\frac{D}{h} + C\frac{h}{6}) & D\frac{2}{h} - C\frac{2h}{3} & -(\frac{D}{h} + C\frac{h}{6}) & \vdots \\ \ddots & \ddots & \ddots & \\ & & & \\ & & & \\ & & \ddots & -(\frac{D}{h} + C\frac{h}{6}) \\ 0 & \dots & -(\frac{D}{h} + C\frac{h}{6}) & D\frac{2}{h} - C\frac{2h}{3} \end{bmatrix}$$

$$\mathrm{B} = \begin{bmatrix} \frac{2}{3}h & \frac{h}{6} & 0 & 0 & \\ \frac{h}{6} & \frac{2}{3}h & \frac{h}{6} & \vdots & \\ \ddots & \ddots & \ddots & & \\ & & & & \\ & & & & \\ & & \ddots & \ddots & \frac{h}{6} \\ 0 & \dots & & \frac{h}{6} & \frac{2}{3}h \end{bmatrix}$$

We compare our solution to the Crank-Nicolson Finite Difference method used in Section 1. We still expect the solution to have an equilibrium at $C = \pi^2$ when $D = L = 1$ and to increase for any values above that and tend towards 0 for anything below that.

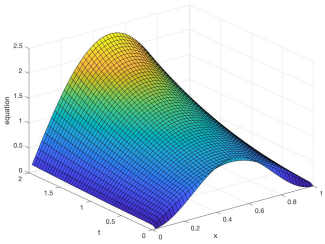
```
% Example usage: w=fisher_fem(0,1,0,2,50,50)
% Current equilibrium at C=pi*pi-0.39 (unstable equilibrium)
% Seems to tend towards 0 for other values
% Finite Element Method and Backward Euler Discretization]
% Solving linear Fisher equation with Dirichlet BCs

function w = fisher_fem(xr,xl,tb,te,M,N)

    D=1; C=(pi*pi); L= xr-xl;
    f=@(x) sin((pi/L)*x).*sin((pi/L).*x);
    l=@(t) 0*t;
    r=@(t) 0*t;
    h=L/M; k=(te-tb)/N; m=M; n=N;

    w(:,1)=f(xl+(0:m-1)*h)';                % initial conditions

    % FEM matrices
    A = diag(2*D/h-(C*2*h/3)*ones(m,1));
    A = A + diag(-D/h-(C*h/6)*ones(m-1,1),-1) + diag(-D/h-(C*h/6)*ones(m-1,1),1);
    B = diag((h/6)*ones(m-1,1),-1)+diag((h/6)*ones(m-1,1),1);
    B = B + diag((2*h/3)*ones(m,1));
```

C = pisq.jpg



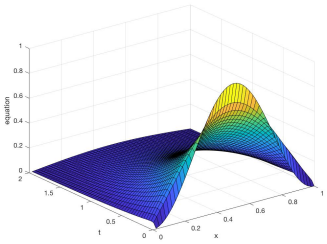
(a) C=π^2 - 0.39

C=10.jpg



(b) C=10

C=8.jpg



(c) C=8

```
for j=1:n
    w(:,j+1) = (B+k.*A)\B*w(:,j);
end
x=xl+(0:M-1)*h; t=tb+(0:N)*k;

figure('Name','Finite Element','NumberTitle','off')
surf(x,t,w')
xlabel('x')
ylabel('t')
zlabel('equation')
```

As shown in the figures above, the system has equilibrium closer to $C = \pi^2 - 0.39$ than at $C = \pi^2$. I'm not sure why this is happening and would need to investigate it further. The behaviour of the system as a rough approximation is fairly satisfactory however, the system thrives for values of $C > \pi^2$ and decays to zero for values of $C < \pi^2$, mirroring the behaviour displayed by the backward-Euler approximation in Section 1. The approximated model is a lot more volatile than that in Section 1 which leads to questions about accuracy, it would also be worth checking if different step sizes/step size ratio has any effect on this.