Numerical Solution of reaction-diffusion problems

Computational Biology Group (CoBi)

Department for Biosystems Science and Engineering (D-BSSE), ETH Zurich, Swiss Institute of Bioinformatics, Basel, Switzerland

Email: Dagmar Iber - dagmar.iber@bsse.ethz.ch;

*Corresponding author

Abstract

This tutorial is concerned with the solution of reaction-diffusion-convection equations of the form

$$\frac{\partial c}{\partial t} + \nabla(cu) = D\Delta c + R(c)$$

on static and growing domains. In the first part we will introduce analytical tools to solve PDEs. In the second part we will introduce finite difference methods for numerical solutions. In the third section we introduce the finite-difference-based solver pdepe in the commercial software package MATLAB to solve reaction-diffusion equations on static and uniformly growing 1D domains. In the fifth part we will provide a brief introduction to finite element methods, and show how these can be used in MATLAB (section 6) and COMSOL (section 7) to solve also reaction-diffusion problems on more complex domains. Finally, we will show how to segment images and solve reaction-diffusion models on these shapes using either MATLAB or COMSOL.

Contents

1 Types of PDEs

Dependent on the type of problem we need to specify initial conditions and/or boundary conditions in addition to the PDE that describes the physical phenomenon. Time-dependent diffusion processes are described by parabolic equations and require the boundary conditions, describing the physical nature of our problem on the boundaries and the initial conditions, describing the physical phenomenon at the start of the experiment. Steady state diffusion equations represent elliptic problems and require only the boundary conditions while hyperbolic reaction-diffusion equations require only initial conditions. The classification of linear PDEs of the form

$$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu = G \tag{1}$$

can be made based on the coefficients, i.e.

- parabolic, if $B^2 - 4AC = 0$ (diffusion and heat transfer)
- hyperbolic, if $B^2 - 4AC > 0$ (vibrating systems and wave phenomena)
- elliptic, if $B^2 - 4AC < 0$ (steady-state phenomena)

The three most important types of boundary conditions are

- Dirichlet boundary condition: concentration is specified on the boundary ($u = g(t)$)
- Mixed boundary condition: concentration of the surrounding medium is specified ($\frac{\partial u}{\partial n} + \lambda u = g(t)$ with n being the *outward normal* direction to the boundary)
- Von Neumann boundary condition: concentration flux across the boundary is specified ($\frac{\partial u}{\partial n} = g(t)$)

2 Solving Reaction-Diffusion Equations analytically

Most reaction-diffusion equations cannot be solved analytically. However, for a small subset a range of techniques has been developed to obtain either exact solutions or good approximations.

2.1 Separation of Variables

This is the oldest technique for solving inital-boundary-value problems (IBVPs) and dates back to Joseph Fourier. The technique is useful if

- PDE is linear and homogenous (not necessarily constant coefficients)
- Boundary conditions are linear and homogenous, i.e.

$$\begin{aligned} \alpha u_x(0,t) + \beta u(0,t) &= 0 \\ \gamma u_x(1,t) + \delta u(1,t) &= 0 \end{aligned}$$

  where $\alpha, \beta, \gamma, \delta$ are constants, $0 < x < 1$.

General Idea Find an infinite number of solutions to the PDE (which at the same time, also satisfy BCs), so-called **fundamental solutions**

$$u_n(x,t) = X_n(x)T_n(t).$$

that can be split into the product of a function of x , $X(x)$, and a function of t, $T(t)$. We are thus looking for solutions that keep their initial shape for different values of time t. The solution $u(x,t)$ is obtained by adding these fundamental solutions in a way that the resulting sum

$$u(x,t) = \sum_{n=1}^{\infty} A_n X_n(x) T_n(t) \tag{2}$$

satisfies the initial conditions.

Example Find the function $u(x,t)$ that satisfies the following four conditions

$$\text{PDE} \qquad u_t = \alpha^2 u_{xx} \quad 0 < x < 1 \quad 0 < t < \infty \tag{3}$$

$$\text{BC} \qquad u(0,t) = u(1,t) = 0 \quad 0 < t < \infty \tag{4}$$

$$\text{IC} \qquad u(x,0) = \phi(x) \quad 0 \le x \le 1 \tag{5}$$

STEP1: Finding elementary solutions to the PDE Substitute

$$u(x,t) = X(x)T(t) \tag{6}$$

such that

$$\text{PDE} \qquad X(x)T'(t) = \alpha^2 X''(x)T(t) \quad 0 < x < 1 \quad 0 < t < \infty \tag{7}$$

Divide each side of this equation by $\alpha^2 X(x)T(t)$, such that

$$\text{PDE} \qquad \frac{T'(t)}{\alpha^2 T(t)} = \frac{X''(x)}{X(x)} \quad 0 < x < 1 \quad 0 < t < \infty \tag{8}$$

This is what is called **separated variables**.

The Trick Inasmuch as x and t are independent of each other, each side must be a fixed constant (say k); hence we can write

$$\text{PDE} \qquad \frac{T'(t)}{\alpha^2 T(t)} = \frac{X''(x)}{X(x)} = -k^2 \quad 0 < x < 1 \quad 0 < t < \infty$$

or

$$\begin{aligned} T' + k^2\alpha^2 T &= 0 \\ X'' + k^2 X &= 0 \end{aligned} \tag{9}$$

Now we can solve each of these ODEs, and multiply them together to get as solution to the PDE

$$u(x,t) = \exp\left(-k^2\alpha^2 t\right)(A\sin(kx) + B\cos kx) \tag{10}$$

where A, B, k are arbitrary.

STEP2: Finding solutions to the PDE and the BCs Choose the subset of solutions

$$u(x,t) = \exp\left(-k^2\alpha^2 t\right)(A\sin(kx) + B\cos kx) \tag{11}$$

that satisfy the boundary conditions

$$\text{BC} \qquad u(0,t) = u(1,t) = 0 \quad 0 < t < \infty \tag{12}$$

We realize that we require $B = 0$ and $k = \pm\pi, \pm 2\pi \pm 3\pi \cdots$. We thus have

$$u_n(x,t) = A_n \exp\left[(-n\pi\alpha)^2 t\right]\sin(n\pi x), \qquad n = 1, 2, \cdots \tag{13}$$

STEP3: Finding solutions to the PDE, BCs, and ICs The last step is to add the fundamental solutions

$$u(x,t) = \sum_{n=1}^{\infty} u_n(x,t) = \sum_{n=1}^{\infty} A_n \exp\left[(-n\pi\alpha)^2 t\right]\sin(n\pi x) \tag{14}$$

and pick the coefficients A_n in such a way that

$$\text{IC} \qquad u(x,0) = \phi(x) \quad 0 \le x \le 1 \tag{15}$$

Substituting the sum into the IC gives

$$\phi(x) = \sum_{n=1}^{\infty} A_n \sin(n\pi x) \tag{16}$$

Orthogonallity of Functions Use orthogonality of $\{\sin(n\pi x); \quad n = 1, 2, \cdots\}$ in the sense that

$$\int_0^1 \sin(n\pi x)\sin(m\pi x)dx = \begin{cases} 0 & m \neq n \\ 1/2 & m = n \end{cases} \tag{17}$$

such that

$$A_m = 2\int_0^1 \phi(x)\sin(m\pi x)dx. \tag{18}$$

Remember

$$u(x,t) = \sum_{n=1}^{\infty} u_n(x,t) = \sum_{n=1}^{\infty} A_n \exp\left[(-n\pi\alpha)^2 t\right]\sin(n\pi x) \tag{19}$$

We now have a solution $u(x,t)$ to the PDE that satisfies the boundary and initial conditions.

2.2 Transforming Nonhomogenous BCs into Homogenous Ones

A PDE with nonhomogenous BCs of the form

$$\text{PDE} \qquad u_t - \alpha^2 u_{xx} = f(x,t) \quad 0 < x < L \quad 0 < t < \infty$$

$$\text{BC} \qquad \begin{aligned} &\alpha u_x(0,t) + \beta u(0,t) = g_1(t) \\ &\gamma u_x(L,t) + \delta u(L,t) = g_2(t) \end{aligned}$$

$$\text{IC} \qquad u(x,0) = \phi(x) \quad 0 \leq x \leq 1$$

can be transformed into a new one with zero BCs

$$\text{PDE} \qquad U_t - \alpha^2 U_{xx} = F(x,t) \quad 0 < x < L \quad 0 < t < \infty$$

$$\text{BC} \qquad \begin{aligned} &\alpha U_x(0,t) + \beta U(0,t) = \mathbf{0} \\ &\gamma U_x(L,t) + \delta U(L,t) = \mathbf{0} \end{aligned}$$

$$\text{IC} \qquad U(x,0) = \phi(x) \quad 0 \leq x \leq 1.$$

The approach is simple: adsorb the non-zero BCs in the PDE, i.e.

$$u(x,t) = \text{steady state} + \text{transient}$$

Hence write

$$u(x,t) = [g_1(t) + \frac{x}{L}(g_2(t) - g_1(t))] + U(x,t) \tag{20}$$

and adjust the BCs and IC. Note that if g_1 and/or g_2 vary with time, then the PDE will become inhomogenous, unless the extra term cancels with $f(x,t)$! The new problem can then be solved by

- Separation of variables if the new PDE just happens to be homogenous $[F(x,t) = 0]$
- Integral transforms and eigenfunction expansion if $F(x,t) \neq 0$ (as is the case for reaction-diffusion problems...)

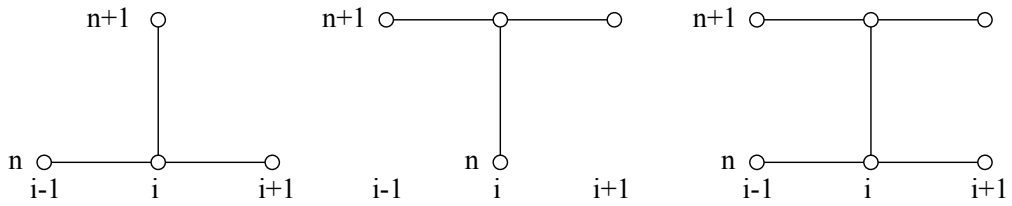


Figure 1: **Different stencils for the approximation to the diffusion equation. (A)** The stencil for a forward-difference scheme for the time derivative and a central difference scheme for the spatial derivative (Eq. 25). **(B)** The stencil for a forward-difference scheme for the time derivative and a central difference scheme for the spatial derivative (Eq. 46). **(C)** The stencil for the Crank-Nicholson scheme (Eq. 47).

3 Finite Difference Approximation

For most PDE problems analytical solutions are difficult or impossible to obtain and solutions must be approximated numerically. To approximate the model equations by finite differences we divide the closed domain by a set of lines parallel to the spatial and time axes to form a grid or a mesh. We shall assume, for simplicity, that the sets of lines are equally spaced such that the distance between crossing points is Δx and Δt respectively. The crossing points are called the grid points or the mesh points. We seek approximations of the solution $u(x_j, t_n)$ to the simple diffusion equation

$$u_t = u_{xx} \tag{21}$$

at these mesh points $(j\Delta x, n\Delta t)$; these approximate values will be denoted

$$U_j^n \approx u(x_j, t_n). \tag{22}$$

We need to approximate the derivatives by finite differences and then solve the resulting difference equations in an evolutionary manner starting at $n = 0$ with the initial conditions. The simplest difference scheme based at the mesh point (x_j, t_n) uses a forward difference for the time derivative (Fig. 1A), i.e.

$$\frac{u(x_j, t_{n+1}) - u(x_j, t_n)}{\Delta t} \approx \frac{\partial u}{\partial t}(x_j, t_n) \tag{23}$$

for any function u with a continuous time derivative. We will use a centered second difference for the second order space derivative:

$$\frac{u(x_{j-1}, t_n) - 2u(x_j, t_n) + u(x_{j+1}, t_n)}{2\Delta x} \approx \frac{\partial^2 u}{\partial x^2}(x_j, t_n). \tag{24}$$

In summary we obtain as approximation

$$U_j^{n+1} = U_j^n + \nu(U_{j-1}^n - 2U_j^n + U_{j+1}^n) \qquad \nu = \frac{\Delta t}{\Delta x^2}. \tag{25}$$

When we carry out simulations with this approximation we soon realize that the quality of the results critically depends on the value of ν. The numerical solution is stable only for sufficiently small ν. The scheme is thus said to be conditionally stable.

3.1 Accuracy and Truncation Errors

There is of course many ways to discretize a derivative (Fig. 2) and the truncation error specifies the accuracy with which the discretized scheme approximates the solution. To determine the truncation error we expand the exact solution at the mesh points of the scheme with a Taylor series and insert the Taylor

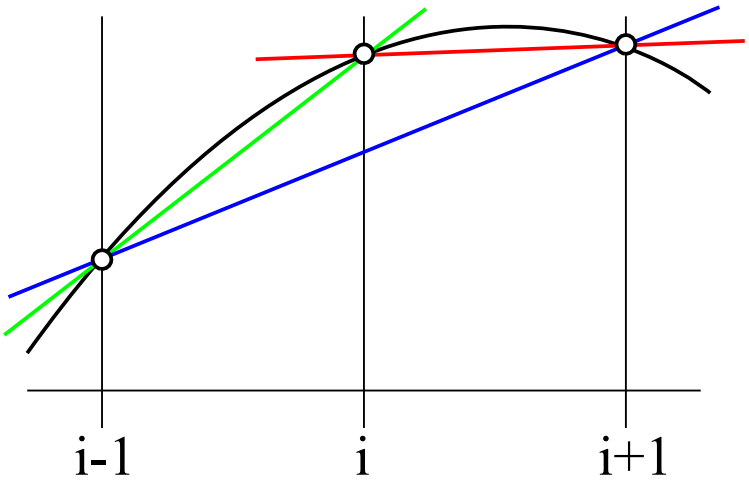


Figure 2: **Discretizations of a derivative.** Graphical comparison of the discretization of a derivative (black) using the forward-difference (red), backward-dfference (green) or central difference scheme (blue).

expansions in the scheme. We then calculate the difference between this approximation and the derivative. In case of a fully accurate scheme we would obtain zero. The non-zero remainder is called truncation error.

Let us look at the one-sided approximations first, i.e.

$$D_+u(x_i) = \frac{u(x_i+h)-u(x_i)}{h} \tag{26}$$

$$D_-u(x_i) = \frac{u(x_i)-u(x_i-h)}{h} \tag{27}$$

We first expand $u(x_i)$ in a Taylor series:

$$u(x_i+h) = u(x_i) + hu'(x_i) + \frac{1}{2}h^2u''(x_i) + \frac{1}{6}h^3u'''(x_i) + O(h^4) \tag{28}$$

$$u(x_i-h) = u(x_i) - hu'(x_i) + \frac{1}{2}h^2u''(x_i) - \frac{1}{6}h^3u'''(x_i) + O(h^4) \tag{29}$$

In case of a one-sided approximation we obtain

$$D_+u(x_i) = \frac{u(x_i+h)-u(x_i)}{h} = u'(x_i) + \frac{1}{2}hu''(x_i) + \frac{1}{6}h^2u'''(x_i) + O(h^3) \tag{30}$$

and thus as truncation error

$$E(h) = D_+u(x_i) - u'(x_i) = \frac{1}{2}hu''(x_i) + \ldots \tag{31}$$

Let us next look at a centered approximation, i.e.

$$D_0u(x_i) = \frac{u(x_i+h)-u(x_i-h)}{2h} = \frac{1}{2}\left(D_+u(x_i) - D_-u(x_i)\right) \tag{32}$$

with

$$u(x_i+h) - u(x_i-h) = 2hu'(x_i) + \frac{1}{3}h^3u'''(x_i) + O(h^5) \tag{33}$$

and thus as truncation error

$$E(h) = \frac{u(x_i + h) - u(x_i - h)}{2h} - u'(x_i) = \frac{1}{6}h^2 u'''(x_i) + O(h^4). \tag{34}$$

We notice that the truncation error increases with h in case of the one-sided approximation and with h^2 in case of the centered approximation. We say that the one-sided approximation is first order accurate while the centered solution is second order accurate. It is important to note that approximations differ in their accuracy.

Determine the truncation error for the approximation:

$$D_3 u(x_i) = \frac{1}{6h}\left(2u(x_i + h) + 3u(x_i) - 6u(x_i - h) + u(x_i - 2h)\right). \tag{35}$$

3.1.1 Deriving Finite Difference Approximations

It is also possible to determine schemes that give the highest order of accuracy with a particular stencil. Suppose we want to find the most accurate approximation for $u'(x_i)$ based on a stencil composed of $u(x_i)$, $u(x_i - h)$, $u(x_i - 2h)$. We thus need to determine a, b, c in

$$D_2 u(x_i) = au(x_i) + bu(x_i - h) + cu(x_i - 2h) \tag{36}$$

such that the truncation error has the highest possible order. We therefore carry out the Taylor expansion

$$\begin{aligned} D_2 u(x_i) &= (a + b + c)u(x_i) - (b + 2c)hu'(x_i) + \frac{1}{2}(b + 4c)h^2 u''(x_i) \ldots \\ &\quad \cdots - \frac{1}{6}(b + 8c)h^3 u'''(x_i) + \ldots \end{aligned} \tag{37}$$

and solve the system

$$a + b + c = 0 \qquad b + 2c = -1/n \qquad b + 4c = 0 \tag{38}$$

The resulting stencil is

$$D_2 u(x_i) = \frac{3u(x_i) - 4u(x_i - h) + u(x_i - 2h)}{2h}. \tag{39}$$

For the second order derivative based on $u(x_i)$, $u(x_i - h)$, $u(x_i + h)$ we obtain

$$D^2 u(x_i) = D_+ D_- u(x_i) = \frac{1}{h^2}\left[u(x_i - h) - 2u(x_i) + u(x_i + h)\right] \tag{40}$$

$$\cdots = u''(x) + \frac{1}{12}h^2 u'''(x_i) + O(h^4) \tag{41}$$

3.1.2 Order of Accuracy for the Approximation of the Diffusion Equation

We previously used as a discretisation for the diffusion equation

$$u_i^{n+1} = u_i^n + \frac{\Delta t}{h^2}\left[u_{i-1}^n - 2u_i^n + u_{i+1}^n\right] \tag{42}$$

The local truncation error is given by

$$\tau(x, t) = \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} - \frac{1}{h^2}\left[u(x - h, t) - 2u(x, t) + u(x + h, t)\right] \tag{43}$$

and with a Taylor expansion in (43) we obtain

$$\tau(x,t) = \left(u_t + \frac{1}{2}\Delta t u_{tt} + \frac{1}{6}\Delta t^2 u_{ttt} + \dots\right) - \left(u_{xx} + \frac{1}{12}h^2 u_{xxxx} + \dots\right) \tag{44}$$

Recall that $u_t = u_{xx}$, $u_{tt} = u_{txx} = u_{xxxx}$ such that

$$\tau(x,t) = \left(\frac{1}{2}\Delta t - \frac{1}{12}h^2\right) u_{xxxx} + O(\Delta t^2 + h^4). \tag{45}$$

The scheme is therefore $O(\Delta t)$ in time, $O(\Delta h^2)$ in space.

3.2 An implicit Method

The stability limit $\Delta t \leq \frac{1}{2}(\Delta x)^2$ is a very severe restriction, and implies that many steps will be necessary to follow the solution over a reasonably large time interval. Moreover, if we want to reduce Δx for higher accuracy Δt needs to be reduced for stability. If we replace the forward time difference (Fig. 1A) by the backward time difference (Eq. 25, Fig. 1B), the space difference remaining the same, we obtain the scheme

$$U_j^{n+1} = U_j^n + \nu(U_{j-1}^{n+1} - 2U_j^{n+1} + U_{j+1}^{n+1}) \qquad \nu = \frac{\Delta t}{\Delta x^2}. \tag{46}$$

This scheme is unconditionally stable but we now need to solve a system of linear equations simultanously.

3.2.1 Crank-Nicholson

The Crank-Nicolson method (Fig. 1C) is based on central difference in space, and the trapezoidal rule in time, and unlike the Backward Euler scheme gives second-order convergence in time ($O(\Delta t^2)$ in time, $O(\Delta h^2)$ in space). The method is unconditionally stable but the approximate solutions can still contain (decaying) spurious oscillations if the ratio of time step to the square of space step is large (typically larger than 1/2). For this reason, whenever large time steps or high spatial resolution is necessary, the less accurate backward Euler method (see above) is often used, which is both stable and immune to oscillations.

$$\begin{aligned}
\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{1}{2}\left[D^2 u_i^n + D^2 u_i^{n+1}\right] &= \frac{1}{2h^2}\left[u_{i-1}^n - 2u_i^n + u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}\right] \\
-ru_{i-1}^{n+1} + (1+2r)u_i^{n+1} - ru_{i+1}^{n+1} &= ru_{i-1}^n + (1-2r)u_i^n + ru_{i+1}^n \qquad r = \frac{\Delta t}{2h^2} \\
\begin{bmatrix} (1+2r) & -r & & \\ -r & (1+2r) & r & \\ & -r & (1+2r) & -r \\ & & & \end{bmatrix} \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ \dots \end{bmatrix} &= \begin{bmatrix} r(g_0^n + g_0^{n+1}) + (1+2r)u_1^n + ru_2^n \\ ru_1^n + (1+2r)u_2^n + ru_3^n \\ \dots \end{bmatrix}
\end{aligned} \tag{47}$$

3.2.2 Multidimensional Problem

Let us consider 2D diffusion as an example of a multidimensional problem.

$$u_t = \Delta u = u_{xx} + u_{yy} \qquad u(x,y,0) = \eta(x,y) \;\; u(x,y,t) = u_\partial(x,y,ts) \tag{48}$$

Discretized Laplacian:

$$\nabla_h^2 u_{ij} = \frac{1}{h^2}\left[u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + 1u_{i,j+1} - 4u_{ij}\right] \tag{49}$$

Discretize in time:

$$\frac{u_{ij}^{n+1} - u_{ij}^{n}}{\Delta t} = \frac{1}{2}\left[\nabla_h^2 u_{ij}^{n} + \nabla_h^2 u_{ij}^{n+1}\right] \tag{50}$$

If we use an implicit scheme a linear system needs to be solved in every time step.

3.3 Boundary Conditions

Dirichlet boundary conditions are easy to implement as the value on the boundary remains fixed in time. To incorporate von-Neumann boundary conditions appropriate discretisations have to be chosen at the boundary that do not reduce the accuracy of the approximation. Consider the boundary condition $u'(0) = \sigma$. There are several reasonable approximations of different accuracy:

- one-sided approx:

$$\frac{u_1 - u_0}{h} = \sigma \tag{51}$$

- centered approx:

$$\frac{1}{h^2}\left[u_{-1} - 2u_0 + u_1\right] = f(x_0) \tag{52}$$

$$\frac{1}{2h}\left[u_1 - u_{-1}\right] = \sigma \tag{53}$$

$$\frac{1}{h}\left[-u_0 + u_1\right] = \sigma + \frac{h}{2} f(x_0) \qquad O(h) \tag{54}$$

- based on u_1, u_2, u_3 (see deriving FD approx):

$$\frac{1}{h}\left[\frac{3}{2}u_0 - 2u_1 + \frac{1}{2}u_2\right] = \sigma \qquad O(h^2) \tag{55}$$

$$\frac{1}{h^2}\begin{bmatrix} \frac{3}{2}h & -2h & \frac{1}{2}h & & \\ 1 & -2 & 1 & & \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}\begin{bmatrix} u_0 \\ u_1 \\ \cdots \\ \\ u_m \end{bmatrix} = \begin{bmatrix} \sigma \\ f(x_1) \\ \cdots \\ \\ f(x_m) - \beta/h^2 \end{bmatrix} \tag{56}$$

3.4 Elliptic Equations

1D Elliptic Equation Elliptic equations correspond to the class of differential operators

$$\sum a_{ij}(x)\partial^2_{x_i x_j} u(x) + \sum b_i(x)\partial x_i u(x) + c(x)u(x) = f(x) \tag{57}$$

which describes time-independent (stationary)/ minimal energy problems such as the steady-state solution of the diffusion equation. Consider the isotropic diffusion problem of the form

$$u_t(x,t) = D u_{xx}(x,t) + \Psi(x,t) \tag{58}$$

IC: $u(x,0) = u_0(x)$, BC: $u(a,t) = \alpha(t)$, $u(b,t) = \beta(t)$. The steady state with $f(x) = -\Psi(x)/D$ can be expressed as

$$u''(x) = f(x) \qquad 0 < x < 1 \qquad u(0) = \alpha \;\; u(1) = \beta \tag{59}$$

We have as unknowns $\{u1, u2, \ldots, u_m\}$ and use as discretization: $\{x_0, x_1, \ldots, x_{m-1}, x_m, x_{m+1}\}$, $h = \frac{1}{m+1}$. With a second order centered difference we have

$$\frac{1}{h^2}\left[u_{i-1} - 2u_i + u_{i+1}\right] = f(x_i) \qquad i = 1 \ldots m \tag{60}$$

which we write as $AU = F$

$$\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_1 \\ \ldots \\ \\ \\ u_m \end{bmatrix} = \begin{bmatrix} f(x_1) - \alpha/h^2 \\ f(x_2) \\ \ldots \\ f(x_{m-1}) \\ f(x_m) - \beta/h^2 \end{bmatrix} \tag{61}$$

4 Solving PDEs using finite-difference methods (FDM) in MATLAB

In this section you will learn how to solve 1-dimensional initial-boundary value problems for PDEs using *pdepe* in MATLAB. *pdepe* can be used to solve parabolic and elliptic PDEs in one spatial variable $x = [a, b]$ and time $t = [t_0, maxt]$. The PDEs have to be passed to the solver in the form of

$$c(x, t, u, \frac{\partial u}{\partial t})\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}(x^m f(x, t, u, \frac{\partial u}{\partial x})) + s(x, t, u, \frac{\partial u}{\partial x}) \tag{62}$$

where m defines the geometry of the problem (slab, cylindrical or spherical geometry). All BCs have to be passed in the form

$$p(x, t, u) + q(x, t)f(x, t, u, \frac{\partial u}{\partial x}) = 0. \tag{63}$$

Thus, within this function p and q have to be defined for both, the left and the right boundary.

The most efficient and clearest way to use *pdepe* is to split the problem into four MATLAB functions:

1. Example.m - The main function containing the definition of the parameters, the call of the solver and the post-processing of the results.
2. ExamplePDEfun.m - Function containing the PDEs in the form of eq. 62.
3. ExampleICfun.m - Function setting the initial concentrations of u for $t = 0$. The initial concentrations can be a function of x.
4. ExampleBCfun.m - Function defining the boundary conditions for u on $x = a$ and $x = b$, respectively.

4.1 Solving reaction-diffusion equations on a static 1D domain

We will use *pdepe* to solve simple reaction systems on a static 1-dimensional domain. We will start this section with a simple diffusion process before expanding the model to include a reaction term. Finally, we will model a system of coupled PDEs. Here we will simulate a Turing pattern using Schnakenberg reactions.

Exercise 1: Diffusion on a static 1D domain
In this first example you will get to know the syntax of *pdepe* and its basic usage. Solve the diffusion equation for one species $u(x,t)$ with diffusion coefficient $D = 1$ on a 1-D domain of length $L = 1$ according to

$$\text{PDE} \qquad \frac{\partial u}{\partial t} = D\Delta u \tag{64}$$

$$\text{BC} \qquad u(0,t) = 1 \quad 0 < t < \infty \tag{65}$$

$$\frac{\partial u(L,t)}{\partial x} = 0 \quad 0 < t < \infty \tag{66}$$

$$\text{IC} \qquad u(0,x) = 0 \quad 0 < x \leq 1 \tag{67}$$

Before simulating the model try to work out whether this model will result in a steady state gradient along the x-axis that could be read out by cells. Check your results by simulating the system in MATLAB; the commented code below shows a possible implementation. Your results should look similar to those in Fig. 3(a) and 3(b).

Exercise 2: Diffusion and degradation on a static 1D domain
Next check the impact of degradation on the concentration profile by including a linear decay term of species u in Eq. 64. Set the degradation constant to $kdeg = 2$. Your simulation results should look similar to Fig. 4(a) and 4(b). Compare your results to those obtained in Exercise 4.1 and discuss the biological implications.

Listing 1: Diffusion.m

```
function Diffusion
% This is the main function. Within this function the meshes are defined,
% PDEPE is called and the results are plotted
clear; close all;

%% Parameters
P(1) = 1; %Diffusion coefficient D
P(2) = 1; %c0
L = 1; %Length of domain
maxt = 1; %Max. simulation time
m = 0; %Parameter corresponding to the symmetry of the problem (see help)
t = linspace(0,maxt,100); %tspan
x = linspace(0,L,100); %xmesh

%%
% Call of PDEPE. It needs the following arguments
% m: see above
% DiffusionPDEfun: Function containg the PDEs
% DiffusionICfun: Function containing the ICs for t = 0 at all x
% DiffusionBCfun: Function containing the BCs for x = 0 and x = L
% x: xmesh and t: tspan
% PDEPE returns the solution as multidimensional array of size
% xmesh x tspan x (# of variables)
sol = pdepe(m,@DiffusionPDEfun,@DiffusionICfun,@DiffusionBCfun,x,t,[],P);
u = sol;

%% Plotting
% 3-D surface plot
```

```
figure(1)
surf(x,t,u,'edgecolor','none');
xlabel('Distance x','fontsize',20,'fontweight','b','fontname','arial')
ylabel('Time t','fontsize',20,'fontweight','b','fontname','arial')
zlabel('Species u','fontsize',20,'fontweight','b','fontname','arial')
axis([0 L 0 maxt 0 P(2)])
set(gcf(), 'Renderer', 'painters')
set(gca,'FontSize',18,'fontweight','b','fontname','arial')

% 2-D line plot
figure(2)
hold all
for n = linspace(1,length(t),10)
    plot(x,sol(n,:),'LineWidth',2)

end
xlabel('Distance x','fontsize',20,'fontweight','b','fontname','arial')
ylabel('Species u','fontsize',20,'fontweight','b','fontname','arial')
axis([0 L 0 P(2)])
set(gca,'FontSize',18,'fontweight','b','fontname','arial')
```

Listing 2: DiffusionPDEfun.m

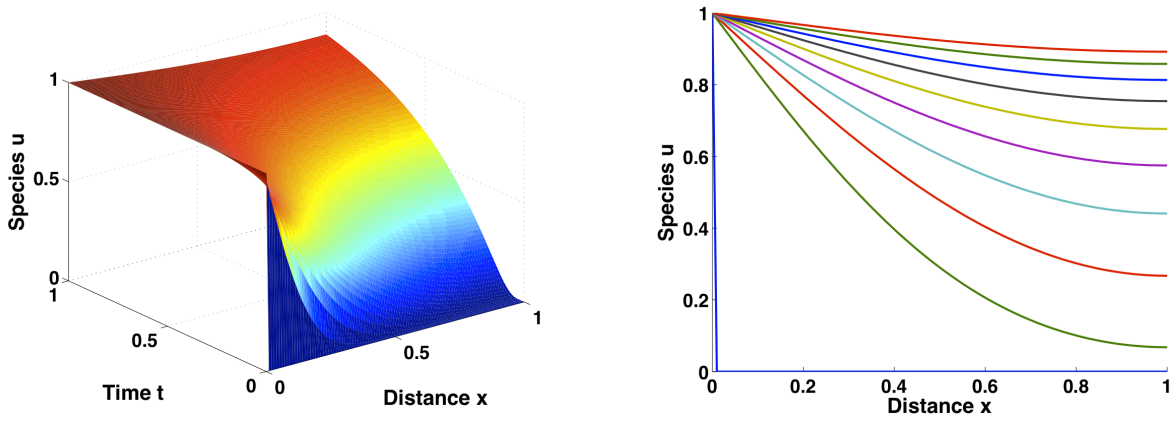
```
function [c,f,s] = DiffusionPDEfun(x,t,u,dudx,P)
% Function defining the PDE

% Extract parameters
D = P(1);
% PDE
c = 1;
f = D.*dudx;
s = 0;
```

Listing 3: DiffusionICfun.m

```
function u0 = DiffusionICfun(x,P)
% Initial conditions for t = 0; can be a funciton of x
u0 = 0;
```

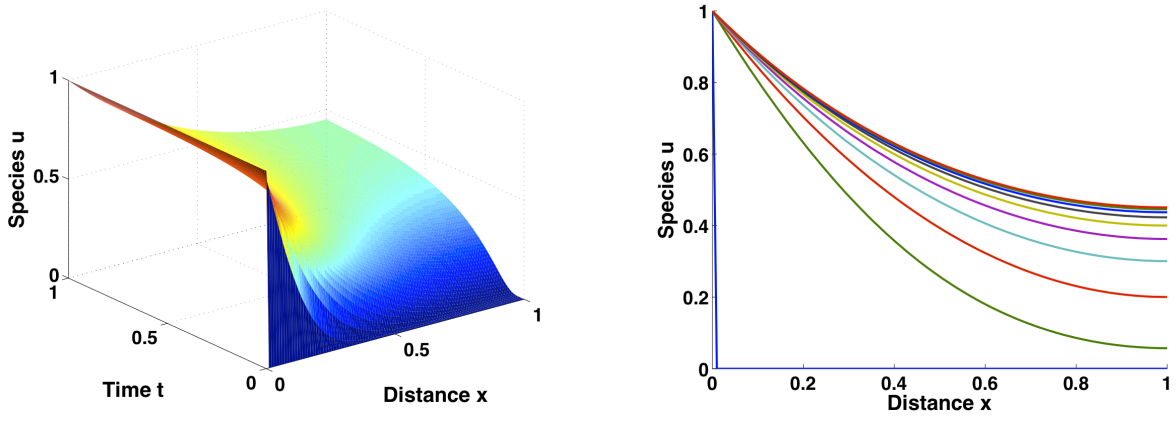
Listing 4: DiffusionBCfun.m

```
function [pl,ql,pr,qr] = DiffusionBCfun(xl,ul,xr,ur,t,P)
% Boundary conditions for x = 0 and x = L;

% Extract parameters
c0 = P(2);

% BCs: No flux boundary at the right boundary and constant concentration on
% the left boundary
pl = ul-c0;     ql = 0;     pr = 0;         qr = 1;
```



(a) (b)

Figure 3: **Diffusion on a static domain.** The domain fills up with species u over time. Read-out of any spatial information is not possible in such a system, as after sufficient long time the whole domain will be filled homogeneously.



(a) (b)

Figure 4: **Reaction-diffusion model on a static domain.** After some time, the system reaches a steady-state where loss by degradation and production by diffusion are balanced. Thus, already simple degradation can provide a way to keep spatial information over time.

Schnakenberg-Turing model on a static 1D domain

In the previous two examples we simulated two very simple PDEs to get to know *pdepe*. Now, we will expand our system to coupled PDEs. We will use the Schnakenberg reactions which result in a Turing pattern. Please refer to the extensive literature for more information on this system. The Schnakenberg kinetics are given by two coupled PDEs as

$$\frac{\partial u_1}{\partial t} = \frac{\partial^2 u_1}{\partial x^2} + \gamma(a - u_1 + u_1^2 u_2) \tag{68}$$

$$\frac{\partial u_2}{\partial t} = d\frac{\partial^2 u_2}{\partial x^2} + \gamma(b - u_1^2 u_2) \tag{69}$$

Exercise 3: Schnakenberg-Turing model

Simulate the Schnakenberg-Turing model described by equations 68 and 69. Choose the parameter values as follows: $L = 1$, $a = 0.2$, $b = 0.5$, $\gamma = 70$, $d = 100$, $maxt = 100$. Choose $u_1(0, x) = 0$ and $u_2(0, x) = 0$ as ICs and no flux BCs on both boundaries. Your final output for species u_1 should like Fig. 5.

Change the parameter values and determine their effects on the model. In particular, what happens when you modify the model in the following ways:

1. increase a or b by factor 10
2. divide a by 10
3. multiply γ with 100
4. vary d

Listing 5: SchnakenbergTuring.m

```
function SchnakenbergTuring
clear; close all;

%% Parameters
L = 1;
maxt = 10;

P(1) = 0.2; %a
P(2) = 0.5; %b
P(3) = 70; %gamma
P(4) = 100; %d

m = 0;
t = linspace(0,maxt,100); %tspan
x = linspace(0,L,100); %xmesh

%% PDEPE
sol = pdepe(m,@SchnakenbergTuringPDEfun,@SchnakenbergTuringICfun,@SchnakenbergTuringBCfun,x,t,[],
    P);
% sol = xmesh x tspan x variables
u1 = sol(:,:,1);
u2 = sol(:,:,2);

%% Plotting
```

```
figure(1)
surf(x,t,u1,'edgecolor','none');
xlabel('Distance x','fontsize',20,'fontweight','b','fontname','arial')
ylabel('Time t','fontsize',20,'fontweight','b','fontname','arial')
zlabel('Species u1','fontsize',20,'fontweight','b','fontname','arial')
axis([0 L 0 maxt 0 max(max(u1))])
set(gcf(), 'Renderer', 'painters')
set(gca,'FontSize',18,'fontweight','b','fontname','arial')

figure(2)
surf(x,t,u2,'edgecolor','none');
xlabel('Distance x','fontsize',20,'fontweight','b','fontname','arial')
ylabel('Time t','fontsize',20,'fontweight','b','fontname','arial')
zlabel('Species u2','fontsize',20,'fontweight','b','fontname','arial')
axis([0 L 0 maxt 0 max(max(u2))])
set(gcf(), 'Renderer', 'painters')
set(gca,'FontSize',18,'fontweight','b','fontname','arial')
```

Listing 6: SchnakenbergTuringPDEfun.m

```
function [c,f,s] = SchnakenbergTuringPDEfun(x,t,u,dudx,P)
% Extract parameters
a = P(1);
b = P(2);
g = P(3);
d = P(4);
% PDEs
c = [1;1];
f = [1; d].*dudx;
s = [g*(a-u(1)+u(1)^2*u(2)); g*(b-u(1)^2*u(2))];
```

Listing 7: SchnakenbergTuringICfun.m

```
function u0 = SchnakenbergTuringICfun(x,P)
u0 = [0;0];
```

Listing 8: SchnakenbergTuringBCfun.m

```
function [pl,ql,pr,qr] = SchnakenbergTuringBCfun(xl,ul,xr,ur,t,P)
% No flux BCs on both sides
pl = [0;0]; ql = [1;1];
pr = [0;0]; qr = [1;1];
```

4.2 Solving reaction-diffusion equations on an uniformly growing 1D domain

In this section, you will learn how to simulate reaction-diffusion PDEs on a growing domain. Once more, we will start from a simple system, diffusion only, and go to more complex systems. While deforming growth of a domain has to be modeled by using finite element methods, as for instance implemented in COMSOL, uniform growth of a simple geometry can be modeled with *pdepe*. The Lagrangian framework used to implement this growth will be only shortly introduced here. For a more detailed discussion please see the references.

The difference between the Eulerian and Lagrangian framework is best explained using a piece of lock on a river. In the Eulerian framework, you as an external observer, see the the piece of lock passing by. In

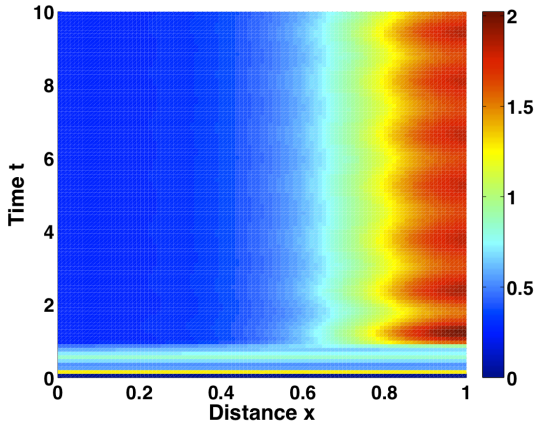


Figure 5: **Concentration profiles for *u1*.** After some time steps a stable distribution is formed along the spatial axis x.

contrast to that, in the Lagrangian framework you are sitting on a boat on the river and are traveling down the river at the same speed. In the context of growing domains, the Lagrangian framework can be used to map an uniform growing domain to a stationary one using a mapping function ψ. It an be shown that this results in Eq. 70 for a reaction-diffusion system on a growing domain.

$$\frac{\partial c}{\partial t} + \frac{\dot{L(t)}}{L(t)}c = D\frac{1}{L(t)^2}\frac{\partial^2 c}{\partial X^2} + R(c) \tag{70}$$

where $L(t)$ is the length of the domain at time t, $R(c)$ is some reaction term, X is the spatial variable in the Lagrangian framework and the term $\frac{\dot{L(t)}}{L(t)}c$ describing the dilution of species c . The reaction-diffusion equation on a growing domain thus becomes a reaction diffusion equation on a fixed domain with a dilution term and a time- and thus length-dependent diffusion coefficient.

Exercise 4: Diffusion on an uniformly growing domain
Simulate diffusion on an uniformly growing domain according to Eq. 70. Consider diffusion only ($R(c) = 0$). This gives us the possibility to exclusively study the effect of diluting species u through the growth of the domain. Assume no flux on both boundaries. Choose the parameter values as follows:

- $D = 1$ (Diffusion coefficient)
- $L = 1$ (Length of domain)
- $c0 = 1$ (initial concentration of u on domain x)
- $maxt = 10$
- $v = 1/10$ (velocity of growth)

Listing 9: Uniform growth - Main function

```
function UniGrowDiffusion
clear; close all;

%% Parameters
P(1) = 1; %Diffusion coefficient D
P(2) = 1; %c0 inital concentration
P(3) = 1/10; %velocity v of growth
P(4) = 1; %L length of domain

maxt = 10; %Max. simulation time
m = 0;
t = linspace(0,maxt,100); %tspan
x = linspace(0,P(4),100); %xmesh

%%
sol = pdepe(m,@UniGrowDiffusionPDEfun,@UniGrowDiffusionICfun,@UniGrowDiffusionBCfun,x,t,[],P);
u = sol;    % sol: xmesh x tspan x variablee

% 3-D surface plot
figure(1)
surf(x,t,u,'edgecolor','none');
xlabel('X (Lagrangian framework)','fontsize',20,'fontweight','b','fontname','arial')
ylabel('Time t','fontsize',20,'fontweight','b','fontname','arial')
zlabel('Species u','fontsize',20,'fontweight','b','fontname','arial')
axis([0 P(4) 0 maxt 0 max(max(u))])
set(gcf(), 'Renderer', 'painters')
set(gca,'FontSize',18,'fontweight','b','fontname','arial')
```

Listing 10: Uniform growth - PDE

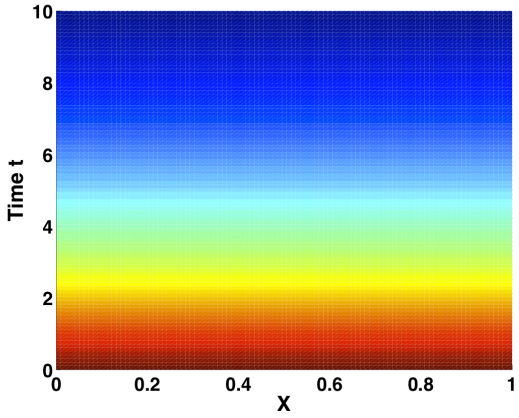
```
function [c,f,s] = UniGrowDiffusionPDEfun(x,t,u,dudx,P)
% Extract parameters
D = P(1);
v = P(3);
L = P(4);
% PDE
c = 1;
f = (D/(L+v*t)^2).*dudx;
s = -v/(L+v*t)*u;
```

Listing 11: Uniform growth - ICs

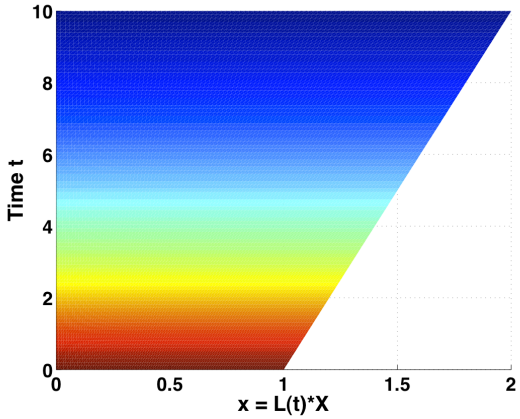
```
function u0 = UniGrowDiffusionICfun(x,P)
c0 = P(2);
u0 = c0;
```

Listing 12: Uniform growth - BCs

```
function [pl,ql,pr,qr] = UniGrowDiffusionBCfun(xl,ul,xr,ur,t,P)
% BCs: No flux
pl = 0; ql = 1;
pr = 0; qr = 1;
```



(a) Lagrangian framework



(b) Eulerian framework

Figure 6: **Impact of dilution in case of uniform growth.** Species u gets diluted to $c0/2$ if the domain size is doubled.

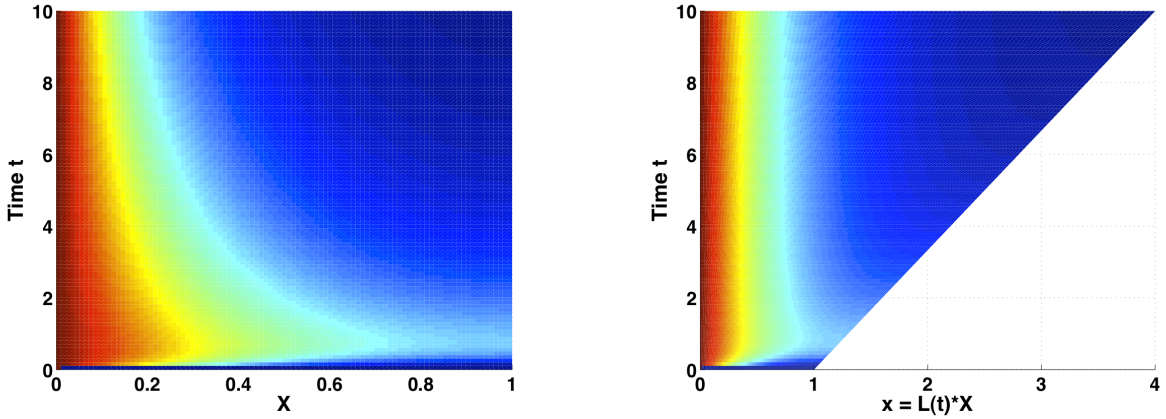
Exercise 5: Reaction-diffusion model on an uniformly growing domain
Include linear degradation (as in example 2 from section 4.1) into your model of diffusion on an uniformly growing domain. Choose the parameter as follows:

- $D = 1$ (Diffusion coefficient)
- $L = 1$ (Length of domain)
- $c0 = 1$ (Concentration on boundary $x = a$
- no flux BC on second boundary
- $u(0, x) = 0$ as IC
- $maxt = 10$
- $v = 3/10$ (velocity of growth)

Your final result should look like Fig. 7. Interpret the results. Imagine you are sitting on the very tip of the growing domain. Which concentration would you see over time? Show the concentration profile at this point over time.

Listing 13: Uniform growth - Main function

```
function UniGrowReactDiff
clear; close all;

%% Parameters
P(1) = 1; %Diffusion coefficient D
P(2) = 1; %c0
P(3) = 3/10; %v
P(4) = 1; %L
P(5) = 2; %kdeg
```



(a) Lagrangian framework

(b) Eulerian framework

Figure 7: **Impact of growth on concentration profile shown for the whole domain over time.**

```
10
11 maxt = 10; %Max. simulation time
12 m = 0;
13 t = linspace(0,maxt,100); %tspan
14 x = linspace(0,P(4),100); %xmesh
15
16 %%
17 sol = pdepe(m,@UniGrowReactDiffPDEfun,@UniGrowReactDiffICfun,@UniGrowReactDiffBCfun,x,t,[],P);
18 % sol xmesh x tspan x variablee
19 u = sol;
20
21 % 3-D surface plot
22 figure(1)
23 surf(x,t,u,'edgecolor','none');
24 xlabel('X (Lagrangian framework)','fontsize',20,'fontweight','b','fontname','arial')
25 ylabel('Time t','fontsize',20,'fontweight','b','fontname','arial')
26 zlabel('Species u','fontsize',20,'fontweight','b','fontname','arial')
27 axis([0 P(4) 0 maxt 0 P(2)])
28 set(gcf(), 'Renderer', 'painters')
29 set(gca,'FontSize',18,'fontweight','b','fontname','arial')
```

Listing 14: Uniform growth - PDE

```
1 function [c,f,s] = UniGrowReactDiffPDEfun(x,t,u,dudx,P)
2 % Extract parameters
3 D = P(1); v = P(3); L = P(4); kdeg = P(5);
4 % PDE
5 c = 1;
6 f = (D/(L+v*t)^2).*dudx;
7 s = -v/(L+v*t)*u-kdeg*u;
```

Exercise 6: Schnakenberg-Turing model on an uniformly growing domain
Transferring Eq. 68 and 69 into the Lagrangian framework results in

$$\frac{\partial u_1}{\partial t} = \frac{1}{L(t)^2}\frac{\partial^2 u_1}{\partial x^2} + \gamma(a - u_1 + u_1^2 u_2) - \frac{\dot{L(t)}}{L(t)}u_1 \tag{71}$$

$$\frac{\partial u_2}{\partial t} = \frac{d}{L(t)^2}\frac{\partial^2 u_2}{\partial x^2} + \gamma(b - u_1^2 u_2) - \frac{\dot{L(t)}}{L(t)}u_2. \tag{72}$$

Implement this model on an uniformly growing domain. Choose the velocity of growth as $v = 5$. Your final figures should like Fig. 8.

Listing 15: Schnakenberg reactions in the Lagrange framework

```
function [c,f,s] = UniGrowSchnakenbergPDEfun(x,t,u,dudx,P)
% Extract parameters
a = P(1); b = P(2); g = P(3);
d = P(4); v = P(5); L0 = P(6);
% PDEs
grow = (v*t+L0);
dil1 = v/(v*t+L0)*u(1);
dil2 = v/(v*t+L0)*u(2);

c = [1;1];
f = [1/grow; d/grow].*dudx;
s = [g*(a-u(1)+u(1)^2*u(2))-dil1; g*(b-u(1)^2*u(2))-dil2];
```

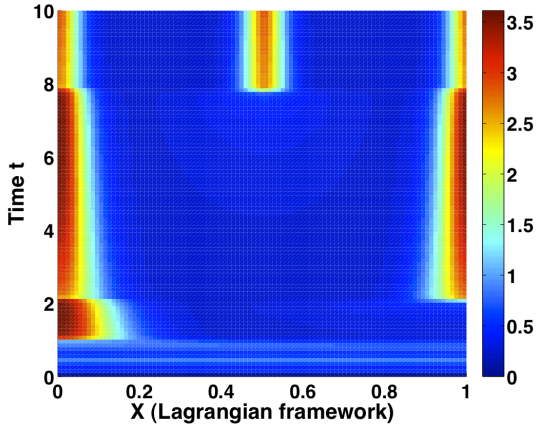


Figure 8: **Impact of growth on Turing patterns created through Schnakenberg reactions.**

5 Finite Element Methods (FEM)

FEM is a numerical method to solve boundary value problems. The power of the FEM is the flexibility regarding the geometry. A highly accessible introduction to FEM can be found at http://www.mathworks.ch/ch/help/pde/ug/basics-of-the-finite-element-method.html.

Summary:

- Find the weak formulation of the problem
- Divide the domain into subdomains
- Choose basic functions for the subdomains
- Formulate a system of linear equations and solve it. The linear system is typically sparse.

5.1 Introduction to FEM

Let us consider the following ODE:

$$\frac{d^2u}{dx^2} + 1 = 0, \quad x \in \Omega \tag{73}$$

$$u(x) = 0, \quad x \in \partial\Omega \tag{74}$$

where $\Omega = [0, 1] \subset \mathbb{R}$. The ODE can be solved analytically by using the ansatz $u(x) = ax^2 + bx + c$.

$$\begin{aligned} \frac{d^2u}{dx^2} = 2a = -1 &\Rightarrow a = -\frac{1}{2}. \\ u(0) = 0 &\Rightarrow c = 0. \\ u(1) = 0 &\Rightarrow a + b = 0 \Rightarrow b = \frac{1}{2}. \end{aligned}$$

We thus see that this two-point boundary problem has a unique solution.

Multiplying the differential equation by a test function v and integrating over the domain gives the weak formulation of the problem.

$$\int_\Omega v \left(\frac{d^2u}{dx^2} + 1 \right) dx = 0 \tag{75}$$

Because v also has to fulfill the BCs, integration by parts yields:

$$\int_\Omega v \frac{d^2u}{dx^2} dx = \left[\frac{du}{dx} v \right]_0^1 - \int_\Omega \frac{du}{dx} \frac{dv}{dx} dx = - \int_\Omega \frac{du}{dx} \frac{dv}{dx} dx.$$

In summary,

$$\begin{aligned} \text{Strong formulation:} \quad & \frac{d^2u}{dx^2} + 1 = 0 \\ \text{Weak formulation:} \quad & - \int_\Omega \frac{du}{dx} \frac{dv}{dx} dx + \int_\Omega v dx = 0. \end{aligned}$$

Discretization of the domain: This is obvious in 1D, but much more demanding in 2D or 3D. In our case we divide the domain into N equal subdomains. Let us denote the basis functions by ϕ. Then we can define the discretized u and v as:

$$\begin{aligned} u_h(x) &= \sum_{i=1}^{N} u_i \phi_i(x) \\ v_h(x) &= \sum_{i=1}^{N} v_i \phi_i(x) \end{aligned}$$

Because in the weak formulation the functions only have to be differentiable once, we can use piecewise linear basis functions:

$$\phi_i(x) = \begin{cases} \frac{x - x_{i-1}}{h} & x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{h} & x \in [x_i, x_{i+1}] \\ 0 & \text{else} \end{cases}$$

Plugging the discretized functions for u and v into the weak formulation yields:

$$\sum_{i,j} v_i u_j \int_0^1 \frac{d\phi_i}{dx} \frac{d\phi_j}{dx} dx = \sum_i v_i \int_0^1 \phi_i dx$$

This formula should be valid for any test function v_h. By choosing $v_i = \delta_{i,j}$ it simplifies to a system of linear equations.

$$\frac{1}{h} \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = h \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Solving the system of linear equations results in:

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 2h^2 \\ 3h^2 \\ 3h^2 \\ 2h^2 \end{pmatrix}$$

5.2 FEM for Two Dimensions

Consider the general Possion equation

$$\begin{aligned} \Delta u(\mathbf{x}) &= -f(\mathbf{x}) \quad , \quad \mathbf{x} \in \Omega \\ u(\mathbf{x}) &= f_D(\mathbf{x}) \quad , \quad \mathbf{x} \in \Gamma_D \\ n \cdot \nabla u(\mathbf{x}) &= f_N(\mathbf{x}) \quad , \quad \mathbf{x} \in \Gamma_N \end{aligned}$$

where Γ_D denotes Dirichlet boundaries and Γ_N von Neumann boundaries. Again we have to multiply the equation with a test function v and integrate over the domain: $\int_\Omega v(\Delta u + f)\, dV = 0$.

$$\begin{aligned} \int_\Omega v(\Delta u)\, dV &= \int_\Omega \nabla \cdot (v \nabla u)\, dV - \int_\Omega \nabla v \cdot \nabla u\, dV & (76) \\ &= \int_{\Gamma_D} v \nabla u\, dS + \int_{\Gamma_N} v \nabla u\, dS - \int_\Omega \nabla v \cdot \nabla u\, dV & (77) \end{aligned}$$

where we used Gauss's divergence theorem. ∇u is only known for Neumann BCs, therefore we set $v(\mathbf{x}) = 0, \mathbf{x} \in \Gamma_D$ at the Dirichlet boundaries. This results in the following weak formulation of the Poisson equation:

$$\int_\Omega \nabla v \cdot \nabla u \, dV = \int_\Omega f v \, dV + \int_{\Gamma_N} f_N v \, dS$$

In comparison to the strong formulation the solutions have to be differentiable only once, but also need to be integrable. Therefore the solutions are in the so-called Sobolev space. Triangles should cover the whole domain, have common edges with their neighbors and each edge has to belong to only one BC. Because the triangles can be very different it is helpful to define all basis functions on a reference triangle and then map it to the real domain. On the reference domain we can, for example, define three different basis functions denoted by N_k.

$$\begin{aligned} N_1 &= 1 - \tilde{x} - \tilde{y} \\ N_2 &= \tilde{x} \\ N_3 &= \tilde{y} \end{aligned}$$

Arranging the basis functions properly will result in the following 2D hat function.

$$\phi_i(x) = \begin{cases} N_3(\mathbf{x}) & \mathbf{x} \in T_1 \bigcup T_4 \\ N_2(\mathbf{x}) & \mathbf{x} \in T_2 \bigcup T_5 \\ N_1(\mathbf{x}) & \mathbf{x} \in T_3 \bigcup T_6 \\ 0 & \text{else} \end{cases}$$

The system of linear equations can now be assembled. A is the so-called stiffness matrix.

$$\begin{aligned} (A)_{i,j} &= \int_\Omega \nabla \phi_i \cdot \nabla \phi_j \, dV \\ (b)_i &= \int_\Omega \phi_i f \, dV + \int_\Omega \phi_i f_N \, dS \end{aligned}$$

We then just have to solve $Au = b$ to get the solution vector u.

5.3 FEM for Time-dependent PDEs

Up until now we only considered time-independent problems, as they are easier to handle. Nevertheless what we are really interested in is solving time-dependent PDE's such as the diffusion equation:

$$\frac{du(\mathbf{x}, t)}{dt} = D\Delta u(\mathbf{x}, t)$$

For temporal discretization the same methods already used in FDM can be applied, i.e.

- Forward Euler
- Backward Euler
- Crank-Nicholson

where the forward Euler method is the easiest but most unstable method. A more recent method, the so-called discontinous Galerkin method, is based on a finite element formulation in time with piecewise polynomials of degree q. With $q = 0$ one obtains the Backward Euler scheme. Discontinous Galerkin method are better in case of variable coefficients and non-zero right hand sides and non-linearities. Precise error estimates are possible such that efficient methods can be developed for automatic time-step control, which are of particular importance for stiff problems.

Let us now look at the diffusion equation in a forward Euler scheme:

$$\begin{aligned} \frac{u^k - u^{k-1}}{\Delta t} &= \nabla^2 u^{k-1} \\ u^k &= \Delta t \nabla^2 u^{k-1} + u^{k-1} \end{aligned}$$

where k denotes the discrete time steps that are Δt apart. ∇^2 is the Laplace operator. We can again easily find a weak formulation for the problem.

$$\int_\Omega u^k v \, dx = -\Delta t \int_\Omega \nabla u^{k-1} \nabla v \, dx + u^{k-1} v \, dx,$$

which can be again written in terms of the basis functions ϕ_i and ϕ_j. In order to have a shorter notation, we will define $M = \int_\Omega \phi_i \phi_j \, dx$ and $K = \int_\Omega \nabla \phi_i \nabla \phi_j \, dx$. This results in:

$$M U^k = (M - \Delta t \, K) \, U^{k-1}$$

This is solved iteratively until all u^k are known.

6 Solving PDEs using finite-element methods (FEM) in MATLAB

MATLAB offers the function *parabolic* to solve parabolic PDEs of the form

$$d \frac{\partial u}{\partial t} - \nabla \cdot (c \nabla u) + a u = f \quad \text{on} \quad \Omega. \tag{78}$$

with FEM. The syntax is

```
u1=parabolic(u0,tlist,b,p,e,t,c,a,f,d,rtol,atol)
```

on a mesh described by p, e, and t, with boundary conditions given by b, and with initial value u0. The function accepts also systems of PDEs. In that case the coefficients are vectors. For the scalar case, each row in the solution matrix u1 is the solution at the coordinates given by the corresponding column in p. Each column in u1 is the solution at the time given by the corresponding item in tlist. For a system of dimension N with np node points, the first np rows of u1 describe the first component of u, the following np rows of u1 describe the second component of u, and so on. Thus, the components of u are placed in the vector u as N blocks of node point rows.b describes the boundary conditions of the PDE problem. b can be either a Boundary Condition matrix or the name of a Boundary M-file. The boundary conditions can depend on t, the time. The formats of the Boundary Condition matrix and Boundary M-file are described in the entries on assemb and pdebound, respectively. The geometry of the PDE problem is given by the mesh data p, e, and t. For details on the mesh data representation, see initmesh. The coefficients c, a, d, and f of the PDE problem can be given in a variety of ways. The coefficients can depend on t, the time. For a complete listing of all options, see initmesh. atol and rtol are absolute and relative tolerances that are passed to the ODE solver.

Exercise 7: Solve the heat equation on a square geometry using FEM
Solve the heat equation

$$\frac{\partial u}{\partial t} = \Delta u \tag{79}$$

on a square geometry $-1 \leq x, y \leq 1$ using the MATLAB function 'squareg'. Choose $u(0) = 1$ on the disk $x^2 + y^2 < 0.4$, and $u(0) = 0$ otherwise. Use Dirichlet boundary conditions $u = 0$ using the MATLAB function 'squareb1'. Compute the solution at times $t = 0 : 0.1 : 20$.

Set the initial conditions to $u(0) = 1$, and set the boundary conditions to $u = 1$ on one boundary, and apply zero-flux boundary conditions otherwise. Solve the equations. Add a linear degradation term with degradation rate $k_deg = 2$. Compare the solutions.

Listing 16: Simple Diffusion Equation with FEM

```
%  Solve the heat equation on a square geometry -1 <= x, y <= 1 (squareg).
%  Choose u(0) = 1 on the disk x2 +y2 < 0.4, and u(0) = 0 otherwise.
%  Use Dirichlet boundary conditions u = 0 (squareb1).
%  Compute the solution at times linspace(0,0.1,20).


function pdeFEM()

% ------------------------------------------------------------------------
% 1) Create the mesh
%
% p,e,t]=initmesh(g) returns a triangular mesh using the geometry
% specification function g. It uses a Delaunay triangulation algorithm.
% The mesh size is determined from the shape of the geometry.
%
% g describes the geometry of the PDE problem. g can either be a
% Decomposed Geometry matrix or the name of a Geometry M-file.
% The formats of the Decomposed Geometry matrix and Geometry M-file are
% described in the entries on decsg and pdegeom, respectively.
%
% The outputs p, e, and t are the mesh data.
%
% In the Point matrix p, the first and second rows contain
% x- and y-coordinates of the points in the mesh.
%
% In the Edge matrix e, the first and second rows contain indices of the
% starting and ending point, the third and fourth rows contain the
% starting and ending parameter values, the fifth row contains the
% edge segment number, and the sixth and seventh row contain the
% left- and right-hand side subdomain numbers.
%
% In the Triangle matrix t, the first three rows contain indices to the
% corner points, given in counter clockwise order, and the fourth row
% contains the subdomain numbe

[p,e,t]=initmesh('squareg');
[p,e,t]=refinemesh('squareg',p,e,t);


% ------------------------------------------------------------------------
% 2.1) Solve the PDE

u0=zeros(size(p,2),1);                  % initial condition
ix=find(sqrt(p(1,:).^2+p(2,:).^2)<0.4); % initial condition
u0(ix)=ones(size(ix));                  % initial condition
```

```
tlist=linspace(0,0.1,20);               % time interval
c = 1;
a = 0;
f = 0;
d = 1;
u1=parabolic(u0,tlist,'squareb1',p,e,t,c,a,f,d);


% ------------------------------------------------------------------------
% 2.2) Plot solution of PDE

figure(1)
for j=1:10:length(tlist)
    plot3(p(1,:), p(2,:),u1(:,j),'.');
    hold on,
end


%------------------------------------------------------------------------
% 3.1) Modify the equations

% Set the initial conditions to $u(0) = 1$ and set the boundary
% conditions to $u = 1$ on one boundary, and use zero-flux boundary
% conditions otherwise. Solve the equations.

u0=ones(size(p,2),1); % initial condition
tlist=linspace(0,1,100);               % time interval
u1=parabolic(u0,tlist,@pdebound,p,e,t,1,0,0,1);

% 3.2) Plot final solution of PDE
figure(2)
plot3(p(1,:), p(2,:),u1(:,length(tlist)),'.');


% ------------------------------------------------------------------------
% 4.1 Add a linear degradation term with degradation rate $k_deg=2$.
u0=ones(size(p,2),1); % initial condition
kdeg =2;                % degradation rate
tlist=linspace(0,1,100);              % time interval
u1=parabolic(u0,tlist,@pdebound,p,e,t,1,kdeg,0,1);

% 4.2) Plot final solution of PDE
figure(3)
plot3(p(1,:), p(2,:),u1(:,length(tlist)),'.');
end


%%------------------------------------------------------------------------
% Boundary function
function [qmatrix,gmatrix,hmatrix,rmatrix] = pdebound(p,e,u,time)

ne = size(e,2); % number of edges
qmatrix = zeros(1,ne);
gmatrix = qmatrix;
hmatrix = zeros(1,2*ne);
rmatrix = hmatrix;

for k = 1:ne
    x1 = p(1,e(1,k)); % x at first point in segment
    x2 = p(1,e(2,k)); % x at second point in segment
    xm = (x1 + x2)/2; % x at segment midpoint
    y1 = p(2,e(1,k)); % y at first point in segment
    y2 = p(2,e(2,k)); % y at second point in segment
    ym = (y1 + y2)/2; % y at segment midpoint
```

```
    switch e(5,k)
        case {1} % pick one boundary
            hmatrix(k) = 1;
            hmatrix(k+ne) = 1;
            rmatrix(k) = 1;
            rmatrix(k+ne) = 1;
        otherwise % other boundaries
            qmatrix(k) = 0;
            gmatrix(k) = 0;
    end
end
end
```

7 Solving PDEs using finite-element methods (FEM) in COMSOL

7.1 Example 1 - A simple diffusion equation (no reactions!) on a 1D domain

In this first example you will learn how to get started with COMSOL, i.e. how to create a model and how to further expand it with more details such as the model geometry and the equations to be solved. Furthermore, you will see how important it is to chose the right mesh.

We will solve the diffusion equation

$$\frac{\partial c}{\partial t} = D\Delta c \tag{80}$$

on a 1D domain of length h_0.

Create the model core

Start COMSOL. We first need to create a core of the model. Here we need to specify the dimension of the model (1, 2 or 3D), the type of equation to be solved as well as type of problem e.g. steady state or time dependent. For this we can use the Model Wizard.

1. Select the 1D button to build a 1 dimensional model.
2. Click Next ($\Rightarrow$).
3. In the Add Physics tree, select Chemical Species Transport and then Transport of Diluted Species. This allows us to solve PDEs of reaction-diffusion type.
4. Add the selected physic (+) and continue ($\Rightarrow$).
5. Choose Time Dependent from Preset Studies for Physics as we will look at the diffusion equation which is a time dependent problem.
6. Create the core of the model by clicking on Finish (F1 flag).
7. Save your file!

Define parameters

As it is a good practice in programming and modeling to use parameters with assigned numerical values instead of entering these values directly, we will now define our model parameters. Right click on the Global Definitions menu to open its submenu and choose Parameters. Create the following parameters (Name, Expression, Description) in the new window:

1. $D0, 1E-10[m^2/s]$, Diffusion coefficient

2. $h0, 1E-4[m]$, Domain length
3. $c0, 1E-6[mol/m^3]$, Concentration of morphogen at the boundary
4. $k0, 1E-8[mol/(s*m^2)]$, Morphogen flux at the boundary
5. $maxt, 100[s]$, Simulation time

Define a geometry
The next steps is to define the geometry. In this example, we want to solve the diffusion equation on a 1D domain with length h_0.

1. Right click Geometry.
2. Select Interval.
3. Select Interval 1.
4. Type in Right endpoint: h0.
5. Click Build to construct this geometry.

Define your equations, BCs and ICs
After having defined our geometry, we need to define the equations together with the BCs and ICs which we want to solve on this domain.

1. Go to the Convection and Diffusion submenu in Transport of Diluted Species to modify your PDE.
2. Type $D0$ in the field for the Diffusion coefficient. As we don't have any reaction terms in this example we don't need to specify anything else here.
3. To define your BC right click Transport of Diluted Species and select Concentration. A new field will be created named Concentration 1.
4. We now need to define the boundary at which this condition should hold true. Select point 1 in the graphics window and add it (+) in the concentration window to apply the constraint to this boundary. (The default BC is no flux, see also the respective menu).
5. Select Species c and type in $c0$ as BC.

You have specified a constant concentration at the boundary in point 1 for species c now. The other boundary condition remains default.

Define your mesh
The last step before we can run our model is to define the mesh. This step is extremely important for the accuracy of the calculations. If the meshing is too crude, solutions can easily become inaccurate or even wrong. However, selection of a very fine mesh increases the time needed to run the simulation and the demand of memory. Therefore, a balanced meshing has to be chosen.
To define your mesh, go the Mesh menu and choose Extra fine as Element size.

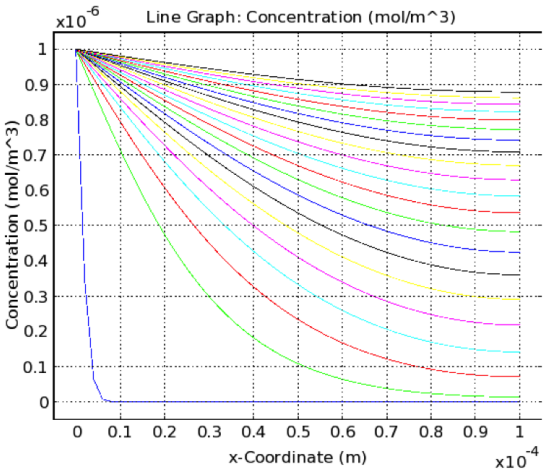


Figure 9: **Output from Example 1:** A simple diffusion equation, without any reaction term, solved on a 1D domain.

Simulation and analysis
The model can now be simulated.

1. Click Step 1: Time Dependent in the Study 1 menu
2. Specify the time interval for the output from the simulation by typing $range(0, maxt/20, maxt)$ in the Times field.
3. Run the simulation (=)

If you followed this tutorial correctly, the output from your model will look like Fig. 9. Save your file and answer the following questions:

1. How does the concentration gradient look like at very early time points?
2. How does it with increasing time?
3. What would you expect for very late time points? Why is this happening?

After having answered these questions, modify your model in the following way:

1. Vary the maximum simulation time (change the respective parameter). Do you get the expected result?
2. Run your simulations with different meshes. What is happening if you select a very crude mesh? (Note: You can also define your own mesh size in the mesh menu).
3. Change the boundary conditions.
   (a) Right click Transport of Diluted Species and select Flux. This will specify a constant flux ($k0$) across the boundary.
   (b) Apply this new constraint on Point 1 and species c ($C_{0,c} = k0$).
   (c) Change $maxt$ to 5 seconds and run the model. What did you expect? How does the profile look like?

7.2 Example 2 - Morphogen binding to receptor, a reaction-diffusion system on a 1D domain

This example is based on a paper published by Lander et al. [1] which was also discussed in the lectures.

As in the previous example, we will consider a 1 dimensional problem. However, you will now learn how to solve systems of coupled PDEs of reaction-diffusion type. You will get insights into formation of morphogen gradients and the importance of degradation of the receptor-morphogen complex for the transduction of positional information.
The scheme of the considered biochemical interactions as well as their formulation in the form of PDEs is taken from [1] (Fig. 10).

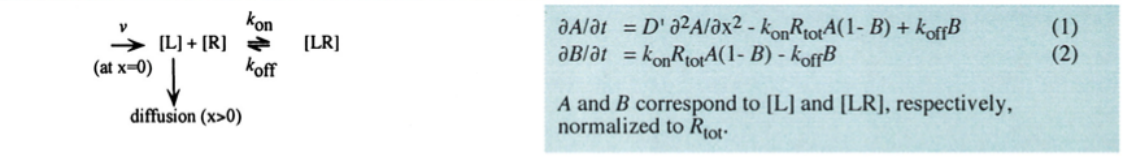


Figure 10: **Example 2:** Scheme of biochemical interactions and PDEs considered in this example. Figures taken from [1]

Create model core and define model parameters
As in the first example (and as always), you need to start with the definition of the core of your model. As we are using the same model core like in example 1, use the same steps to create a 1 dimensional, time dependent model of Transport of Diluted Species. Again, save your file!
Before you go further, open the Preferences of COMSOL. In Model Builder tick the Show More Options box. This will give us the possibility to access options in the process of model building which have been hidden before.
As in example 1, create a Parameters submenu in the Global Definitions and create the following parameters:

1. $D0, 1E-11$, Diffusion coefficient
2. $h0, 1E-4$, Domain length
3. $maxt, 1000$, simulation time
4. $kon, 0.01$, rate constant for complex formation
5. $koff, 1E-6$, rate constant of complex dissociation
6. $c0, 1$, concentration of morphogen at the boundary of the domain
7. $flux0, 1E-6$ Flux of morphogen at the boundary of the domain

Define model geometry
Create a simple 1D domain of length $h0$ as in example 1.

Define your equations, BCs and ICs, and build your mesh

After having created our model core and model geometry as well as having specified the model parameters, we now need to set up our system of PDEs.

1. Click Transport of Diluted Species.
2. As we have multiple species, concentration of morphogen and bound receptor, in this example, open the Dependent Variables field and change Number of species to 2. Name these species c (Morphogen, default) and R (ligand-receptor complex).
3. In this example, diffusion is the sole transport mechanism. Thus, go to the Transport Mechanisms tab and deselect Convection. Migration in electric field should be deselected by default.
4. We need to specify the diffusion of the species now. Change to the Convection and Diffusion menu and specify the diffusion coefficients for C and R as $D0$ and 0, respectively. We are neglecting the diffusion of the receptor, as it is known that receptors usually diffuse 100 - 10000 time slower than morphogens since they are membrane-bound.
5. Make sure that initial concentrations for morphogen and receptor and their derivatives are set to 0 in the Initial Values menu.
6. So far, only diffusion is implemented in the model. To add the reaction terms describing the ligand-receptor interactions right click Transport of Diluted Species and select Reactions.
7. In the new window, we have to define the domain on which the reactions are happening. Select line 1 and add (+) it in the Domains window. Now, specify the reactions happening on this domain (see Fig. 10):
   (a) Type in Rc: $-kon * c * (1 - R) + koff * R$
   (b) Type in Rr: $kon * c * (1 - R) - koff * R$
8. We are still missing the BCs. Right click Transport of Diluted Species and select Flux. Select and add point 1 and apply the BC to species c. Type $flux0$ in kc,A and $c0$ in Cb,c.
9. To mesh your model, go to the Mesh menu and choose Extra fine for the Element size.

Simulation and analysis

1. Set the output times from the simulation to $(0, maxt/20, maxt)$ and run the simulation.
2. Despite the default plots which are generated, you can generate your own plots: Right click Results and select 1D Plot Group. Right click the generated 1D Plot Group and select Line Graph. Now you can specify the parameters you want to be plotted on the x- and y-axis. We want the concentration of bounded receptor to be plotted over the domain length for all time points. Type in the respective parameters and create the plot. You should see a plot like Fig. 11.
3. Obviously, the model we implemented cannot be a representation of the reality so far. The receptors get saturated over the whole domain with time. Thus, a readout of any spatial information would not be possible. As you know from the lectures, degradation of bound receptors helps to maintain spatial information over time. Therefore, we want to include now a degradation term and analyze the effects on the model simulation.
   (a) Go back to menu where you specified the reaction term of the species in the model (Reactions 1 in Transport of Diluted Species).

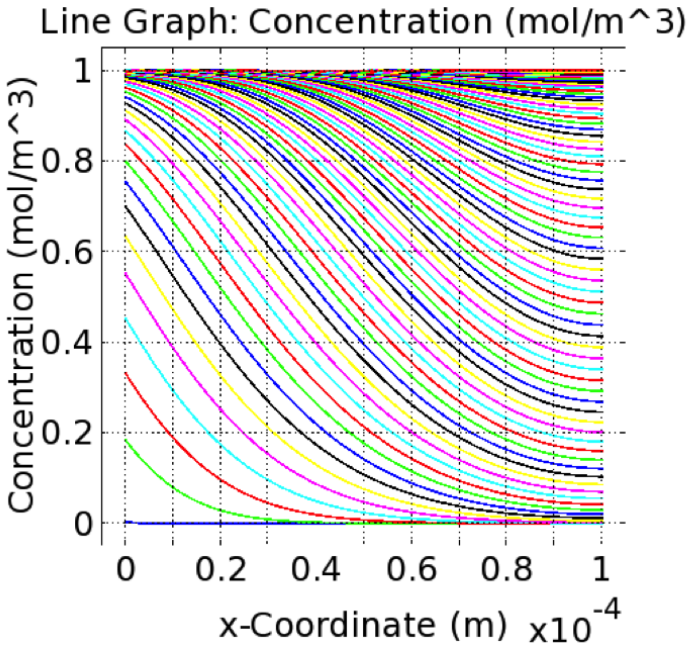


Figure 11: **Example 2:** Output from the simulation. Receptors get saturated with ligand over time. At maximal time, the whole domain is saturated with bound receptors. Thus, cells would not be able to read out any spatial information over time.

(b) To include degradation of the receptor-ligand complex, change Rr to: $kR * c * (1 - R) - koff * R - kdeg * R$.

(c) Add the parameter $kdeg = 0.01$ to your list of Parameters.

(d) Re-run the model. You should get a graph like the one shown in Fig. 3d

(e) Vary the parameter $kdeg$. Which influence does it have on the concentration profile? Try to think about what you would expect before simulating the model

(f) Vary the order of the degradation reaction:

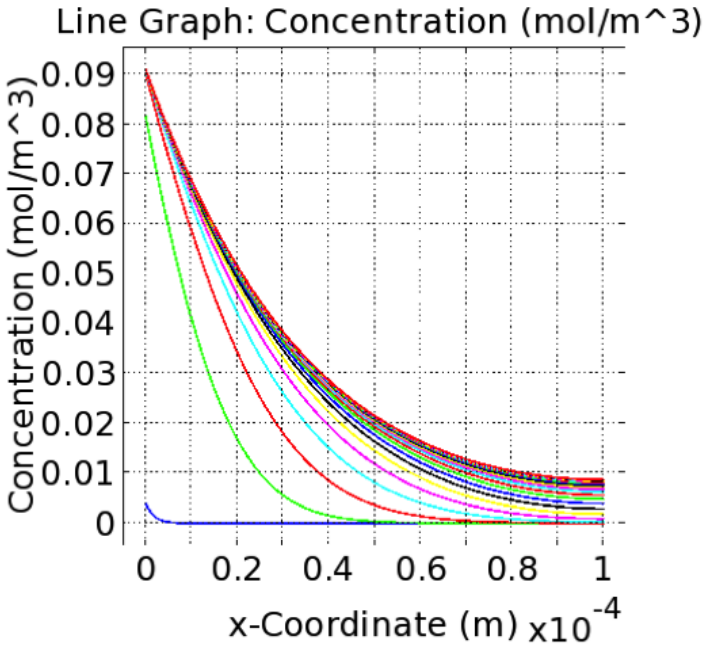
i. Change the reaction term for the receptor-ligand complex Rr to $kR * c * (1 - R) - koff * R - kdeg * R^n$

ii. Set n as a parameter and equal to $0, 1, or 2$. See also Eldar et al. as a paper on the importance of self enhanced degradation [2].

iii. Choose other parameters you expect to have an impact on the morphogen or complex concentration profiles and vary them.

7.3 Example 3 - Cell variability and gradient formation on a 2D domain

This example is based on a paper by Bollenbach et al. [3]
While we have looked at 1 dimensional problems so far, we will now extend our model to two dimensions and study formation of a gradient on this domain.
More important, we will have a look at cell variability with diffusion coefficient, degradation and production rates being a function of spatial coordinates. Furthermore, we will use another interface in COMSOL, the Mathematics interface, which provides us with more flexibility.



[h]

Figure 12: **Example 2:** Output from the simulation with a degradation term included. Spatial information is maintained over all time points.

Create model core and parameters

Similar to the previous examples, we can use the model wizard to create the core of our model. However, select 2D and the General Form PDE from PDE interfaces in the Mathematics field for this example. Once more we will consider a time dependent problem.

Before you continue, go to the preferences of COMSOL and tick the Show More Options in the Model Builder tab.

Create the following parameters (Name, Expression Description). The parameters in this example are dimensionless.

1. $D, 1$, Diffusion coefficient
2. $h0, 1$, Domain length
3. $l0, 1$, Domain width
4. $maxt10$, Simulation time
5. $k, 0.5$, Degradation rate constant
6. $flux, 1$, Morphogen flux at the boundary
7. $na, 0$, Noise amplitude

Furthermore, create the functions we will need to model the cell variability. Right click Global Definitions and select Functions –¿ Random. Name it $noiseD$, Number of arguments is equal to 2, change Distribution to normal and choose Standard deviation as equal to na.

Create two more random functions with the following arguments:

Name: $noisek$, Number of arguments: 2, Standard Deviation: na

Name: $noisef$, Number of arguments: 1, Standard Deviation: na

Geometry
Create a Rectangle geometry with Width $l0$ and Height $h0$.

PDEs and Mesh

1. Double-check that the Number of dependent variables is set to 1 and that the variable is named u in the General Form PDE menu.
2. Go to the General Form PDE 1 menu. Specify the following reaction terms, make sure you understand the nomenclatures used in COMSOL and the meaning of the different terms.
   (a) Conservative Flux x: $-ux * D * (1 + noiseD(x, y))$
   (b) Conservative Flux y: $-uy * D * (1 + noiseD(x, y))$
   (c) Source: $-u * k * (1 + noisek(x, y))$
   (d) Damping of Mass Coefficient: 1
   (e) Mass coefficient: 1
3. Make sure u and it derivative are set to zero in the Initial Values menu.
4. Create a Boundary flux by right clicking General Form PDE and choosing Flux/Source
   (a) Select and add left side of the rectangular
   (b) Set Boundary Flux/Source g to $flux * (1 + noisef(y))$
5. Mesh your domain with an Extra fine mesh. We choose this mesh size as the mesh size should be significantly smaller than any geometrical feature or feature in the solution if accurate results are required.
6. Run the model and output the following times: $(0, maxt/10, maxt)$.

Model analysis
The morphogen gradient should be similar to the once calculated in example 2. Choose a good way to plot your results (you can create 2D and 3D plots similar to the way, you created a 1D plot in example 2).
Now, switch on noise by setting $na = 0.4$. This way we are modeling cell dependent variability in the parameters. Re-run the model and plot the results as cross-section 1D plots:

1. Right click Data Sets and select Cut Line 2D.
2. Go to the new Cut Line 2D submenu and define the cross-section lines
   (a) Point 1: $0, 0.1$
   (b) Point 2: $1, 0.1$
   (c) Tick box Additional parallel lines
   (d) Distance: $range(0, 0.1, 0.5)$
3. Right click Results and select 1D plot group
4. Go to the new menu and select Line graph
5. In 1D plot group in Data set drop down menu select Cut Line 2D
6. Plot the cross-section graph

Before reading the next paragraph, analyze the graphs and try to answer the following questions: Did you expect that? What does it mean for biological systems?
As you can see morphogen gradients are almost independent of the noise level of the model parameters. This shows that even in biosystems with high variability in the parameters gradients provide reliable information for a read-out of spatial information. Now, modify the model:

1. Increase parameter na. Analyze the impact on the morphogen distribution.
2. Restrict cell variability once to the source and once to the sink. Analyze the impact.

8 Image-based Modelling & Simulation

Modern microscopy techniques now provide us with the geometries of biological tissue. In order to be able to solve our models on realistic geometries, we will now segment images, extract boundaries from those images, calculate the displacement fields between two different time points.

8.1 Image Segmentation, Border Extraction, and Smoothing

It is necessary to isolate the portion of the raw images that we find necessary for our work. Segmentation can be done in MATLAB using a number of inbuilt functions and by following these steps.

1. Import the images into MATLAB. Depending on the quality and the brightness of the raw image, the intensity might have to be increased or the contrast changed. The contrast can be increased using the built-in MATLAB function `imadjust`.
2. The image can then be converted to binary scale image using the MATLAB function `im2bw`. This function also allows you to set a threshold such that only those pixels with a value above a specific threshold are selected
3. The threshold filter can assign islands of bright pixels or other objects which are not of our concern as the area of interest. We thus have to separate these area from the ones of our interest. This can be done by first labeling them with the MATLAB function `bwlabel` and then later choosing the one which is the largest.
4. After getting the segmented image, the border can be extracted again using one of the built-in MATLAB functions - `bwboundaries`. The boundary thus obtained has to be smoothed before it could be used for further calculations.
5. The smoothing can be done using the MATLAB function `smooth`. The function offers a number of methods to perform this smoothing and the one of interest or the one which gives the best result can be chosen.
6. After getting the smoothed boundary, the number of points in the boundary can then be adjusted according to our need through interpolation. There are number of interpolation functions available - one of which is interparc [4]

Listing 17: Image Segmentation - Border Extraction - Smoothing

```
1 %% Generating boundaries with n points for all 49 images 0-48 and the coordinates stored in cu.
2
3
4 function segementation
```

```
    load crop_coordinates
    clc
    cur = [];
    n = 500;
    thresh = 0.5;  %% Threshold for isolating points

    cur = [];

    for k = 1:2
        disp(['k=' num2str(k)])
        cur1 = generating_smooth_plots(k,thresh,rect,n);
        cur = [cur cur1];
    end

    figure(10), hold on,
    plot(cur(:,1),cur(:,2))

    clear k n cur1
    save('output_segmentation.mat')
end



function cur1 = generating_smooth_plots(k,thresh,rect,n)
    %% Reading the images
    jpgFilename = strcat('c125_xy1t0', num2str(k), '.tif');

    I = imread(jpgFilename)*10;

    %% Cropping, Contrasting and Isolating biggest area
    I = imadjust(I(:,:,1));
    BW2 = im2bw(I, thresh);
    [I] = imcrop(BW2,rect);

    %% Labeling Areas
    z = bwlabeln(I);

    zmax = max(max(z));


    for i = 1:zmax
        areasize(i) =  length(z(z==i));
    end

    areanumber = find(areasize == max(areasize));

    z(z~= areanumber) = 0;

    %% Generating boundaries
    [B,L] = bwboundaries(z,'noholes');
    for j = 1:length(B)
        contour = B{j};
    end

    %% Smoothing
    cont = contour;

    for lk = 1:25
        cont = [contour(end-lk+1,:);cont;contour(lk,:)];
    end

    xs = smooth(cont(:,1),15); %% moving average of 25 points
    ys = smooth(cont(:,2),15);
```

```
69
70      Xs = xs(lk+1:end-lk,:);
71      Ys = ys(lk+1:end-lk,:);
72      cur1 = interparc(n,Xs,Ys); %% interpolated curve with n points obtained from the piecewise
            linear approximated curve
73  end
```

8.2 Calculation of Displacement field

Depending on the shape of the image and the frequency of time-variant images we have, the displacement between different stages can be calculated as the minimum distance between two consecutive stages. This can be done either using the built-in MATLAB function `pdist2` or using `distance2curve` [5], which use an interpolation method to find out the minimal distance. The values thus obtained for each of the points in the x and y coordinates can then be stored for import into an FEM solver.

Listing 18: Calculating the Displacement field

```
1  %% Generating displacement vectors between the original curve and a new interpolated curve
2  clear
3
4  load output_segmentation.mat
5  clearvars -except cur
6  inicoo = cur(:,1:2);
7  J= [];
8  noofpoints = length(cur(:,1));
9  displacementvals = [];
10
11
12 for k = 1
13      if(k==1)
14          cur1 = cur(:,1:2);
15          curv = cur1;
16      else
17          cur1 = curv(:,2*k-2:2*k);
18      end
19      cur2 = cur(:,2*k+1:2*k+2);
20
21      figure(11), hold on
22      plot(cur(:,1),cur(:,2),'b')
23      plot(cur(:,3),cur(:,4),'r')
24
25      xy = distance2curve(cur2,cur1,'linear');      %% calculating normal distances using
            distance2curve function
26
27      dist2 = xy-cur1;                              %% Displacement
28
29      displacementvals = [displacementvals dist2];
30      %% finding the distance between consecutive points - this is done to get rid of the repeating
             points in the new generated curve - as the interpolation would be impossible in such a
            case
31
32      for m=1:length(xy)-1
33          J(m) = pdist2(xy(m,:),xy(m+1,:));
34      end
35
36      jmean = mean(J);   %%  mean of distance between consecutive points
37
38      %% equating all distances less than mean/30 as 0  - its difficult to get rid of the values in
             the matrix directly due to the varying size everytime - thus equate to zero and get rid
            of the zero values
```

```
39
40      for m = 1:length(xy)-1
41          if (J(m) <= jmean/30)
42              xy(m,:) = [0 0];
43          end
44      end
45
46      %% finding coordinates of these zero values
47
48      [xc,yc] = find(xy(:,1) == 0);
49      coor = [xc yc];
50
51      r= length(xc) - length(find(xy(:,1) == 0));
52      s = 0;
53
54      %% eliminating all the zero values
55
56      for r = 1:length(xc)
57          xy(xc(r)-s,:) = [];
58          s=s+1;
59      end
60
61      xyz = interparc(noofpoints,xy(:,1),xy(:,2));    %% interpolation to 500 points
62
63      curv = [curv xyz];
64
65  end
66
67  clearvars -except cur curv displacementvals
68  save displacement
```

8.3 Simulating on complex domains in MATLAB

The PDE toolbox in MATLAB supports figures with only standard shapes like rectangles, ellipses and polygons. Though it is possible import custom figures using the PDE toolbox, solving for them would be very difficult. Here we will solve the Schnakenberg Turing model in a custom figure which look a little like the kidney epithelium obtained from the pictures.

Creating Geometry, Meshing and specifying Boundary Conditions

1. Open the PDE toolbox in MATLAB by typing `pdetool` and then pressing enter in the command window.

2. Now you can create a custom shape or geometry wherein you want to solve your PDEs. This can be done by selecting one of the options in the toolbar or by going through **Draw** menu and then choosing the desired shape. As we are solving for two variables, select the **Generic System** option on the top right corner or by going through **Options** $->$ **Applications**.

3. After creating the geometry, we need to mesh it. This can be done by clicking on Mesh and followed by **Initialize Mesh** or simply pressing **Ctrl + I**. Once we have a mesh, it can then be further refined or optimized by using the **Refine Mesh** or **Jiggle Mesh** options. Once you have a mesh of desired size you can then export the mesh parameters into the MATLAB workspace by clicking on Mesh $->$ Export Mesh.

4. Now, we need to specify the boundary conditions for the given geometry. For our case, we have Neumann boundary conditions. The boundary conditions can be specified by first going into the boundary mode(**Boundary** $->$ **Boundary Mode**) and then followed by clicking on **Specify Boundary Conditions** under the **Boundary** menu option. Once the boundary conditions are specified they can be

exported to the MATLAB workspace by clicking on **Export Decomposed Geometry, Boundary Cond's** option under the **Boundary** menu.

5. Finally you can save all the parameters in the matlab workspace.

Solving the PDEs

1. Before solving the PDEs, the boundary condition file has to be generated. The general format of the file looks like this

Listing 19: Boundary Conditions

```
function [q,g,h,r]=boundary_conditions1(p,e,u,time)

load mesh_parameters1.mat

bl = b; % Boundary condition saved from PDE toolbox
bl(1:2,:) = [];


if any(size(u))
  [q,g,h,r]=pdeexpd(p,e,u,time,bl);  % q and g are Neumann parameters
else                                 % h and r are Dirichlet parameters
  [q,g,h,r]=pdeexpd(p,e,time,bl);
end
```

2. The initial values need to be specified as well. This can be done using the following function. In this case we are assuming random initial values for both variables

Listing 20: Initial Conditions

```
function [u1initial,u2initial] = lvinitial(x,y)

u1initial = 1+(rand(1,1)-0.5)/10;
u2initial = 1+(rand(1,1)-0.5)/10;

end
```

3. Now the PDEs can be solved by incorporating both the Boundary conditions and the Initial conditions. The MATLAB PDE solver function `parabolic` is used.

Listing 21: Solving the PDEs

```
clear

load mesh_parameters1.mat

a_param = 0.2;
b_param = 1.5;
gamma_param = 0.04;
D = 50;

c = [1 0;0 D];
a = [0 0;0 0];
d = [1 0;0 1];

m=size(p,2);  %% Number of endpoints
n=size(t,2);  %% Number of triangles
```

```
t_final=100;  %% Stop time

M=10;

dt=t_final/M; %% Time-stepping increment (M-file time-stepping)
tlist=linspace(0,dt,2); %% Time vector for MATLAB's time-stepping

%% Rectangular coordinates for plotting
x=linspace(-1.5,1.5,50);
y=linspace(-1.5,1.5,50);
%% Set diffusion
c1 = c(1,1);
c2 = c(2,2);
a1 = a(1,1);
a2 = a(2,2);



%% Initial conditions
for i=1:m %% For each point of the triangular grid
    [u1old(i),u2old(i)]=lvinitial(p(1,i),p(2,i));
end

u1old1 = transpose(u1old);
u2old1 = transpose(u2old);

u1=tri2grid(p,t,u1old1,x,y);
u2=tri2grid(p,t,u2old1,x,y);


figure(1)
imagescwithnan(u1,jet,[0 0 0]);
figure(2)
imagescwithnan(u2,jet,[0 0 0]);

for k=1:M
    %% Nonlinear interaction

    disp(['loop = ' num2str(k)])
    for i=1:m
        f1(i) = gamma_param*(a_param - u1old(i) + u1old(i)*u1old(i)*u2old(i));
        f2(i) = gamma_param*(b_param - u1old(i)*u1old(i)*u2old(i));
    end

    f1_tr = transpose(f1);
    f2_tr = transpose(f2);
    f1center=pdeintrp(p,t,f1_tr);
    f2center=pdeintrp(p,t,f2_tr);

    %% Solve the PDE
    u1new=parabolic(u1old,tlist,'boundary_conditions1',p,e,t,c1,a1,f1center,1);
    u2new=parabolic(u2old,tlist,'boundary_conditions1',p,e,t,c2,a2,f2center,1);

    %% Update u1old and u2old

    u1old=u1new(:,2);
    u2old=u2new(:,2);

    %% Plot each iteration
    u1=tri2grid(p,t,u1old,x,y);
    u2=tri2grid(p,t,u2old,x,y);

    figure(1)
    imagescwithnan(u1,jet,[0 0 0]);
```

```
    figure(2)
    imagescwithnan(u2,jet,[0 0 0]);

end
```

The imagescwithnan is a function to view the images such that the background colour is fixed. It is given as

Listing 22: Solving the PDEs

```
function [h hcb] = imagescwithnan(a,cm,nanclr)

% find minimum and maximum
amin=min(a(:));
amax=max(a(:));
%# size of colormap
n = size(cm,1);
%# color step
dmap=(amax-amin)/n;
%# standard imagesc
him = imagesc(a);
%# add nan color to colormap
colormap([nanclr; cm]);
%# changing color limits
caxis([amin-dmap amax]);
%# place a colorbar
hcb = colorbar;
%# change Y limit for colorbar to avoid showing NaN color
ylim(hcb,[amin amax])

if nargout > 0
    h = him;
end
```

8.4 Simulating on extracted domains in COMSOL

COMSOL Multiphysics is a well established software and several studies have shown that it gives accurate solutions for reaction-diffusion equations both on static [6] as well as growing domain [7–9]. Let us consider solving a traveling wave equation of the form for a static domain

$$\frac{\partial u}{\partial \tau} = D\Delta u + u(1-u) \tag{81}$$

Implementation of the model of interest can be done in COMSOL as follows:

Model Wizard

1. A new COMSOL Multiphysics file needs to be created for the appropriate dimension (in our case 2D).
2. In the **Add Physics** option, you need to select the appropriate system you have to solve. For a **Reaction-Diffusion System** - select the **Coefficient Form PDE** option. If you're working on a growing domain, then select the **Moving Mesh (ale)** option in addition to the previously chosen **Coefficient Form PDE** and then click on **Next**.
3. Under **Preset Studies** choose **Time Dependent** which allows you to solve time dependent problems and click on **Finish**.

A new file would be opened with the selected options. Save your file before continuing further.

Global Definitions

1. Right Click on **Global Parameters** and choose **Parameters**. Here you can define the set of parameters you'll be using in the model - **D**, maximum time for which you want to run the simulation (**maxT**), etc.

2. You can also define a number of functions. The input values for the traveling wave would be in the form of a step function. Thus the **Step Function** can be defined with appropriate location and minimum and maximum values.

Model

The Model parameters are defined here.

1. Under **Geometry**, you can specify the domains on which the equations need to be solved. COMSOL offers a number of options for creating these domains. A Square, Circle, Rectangle, Ellipse and other such simple geometries can be created. Geometries from outside can also be imported using the **Interpolation** function. The file which is imported should contain the coordinates of the required geometry. In addition, Boolean operations can also be performed on a combination of such geometries to obtain the desired geometry. At the end, click on **Build**.

2. Under **Coefficient Form PDE**, you need to define the geometry where you want to solve the set of equations. The number of variables to be used in the equation needs to be defined under **Coefficient Form PDE =>Dependent Variables**

3. Next, the equation needs to be completed. For the travelling wave equation, the value of $c = D$ (Diffusion coefficient defined in Parameters), $a = 0$, $f = u \times (1 - u)$ $e_a = 0$, $d_a = 1$.

4. After defining your equations, the region of **Zero flux** needs to be chosen. This defines the boundary outside which there occurs no diffusion.

5. Finally, the initial value needs to be defined. As mentioned earlier, the initial value for the case of traveling wave equation was in the form of a step function. Thus the function defined earlier under **Global Definitions** can be used here.

6. After defining the set of equations that need to be solved, the computational domains needs to be meshed. Selection of the correct mesh size is very important. If the mesh size is too small, the results observed would be more accurate, but the time for computation as well as the memory usage would be really high. If the mesh size is very crude, the solutions observed might be inaccurate or wrong. Click on Mesh and select **User-controlled mesh**. **Right Click** on **Mesh** and select **Free Triangular**. Again **Right click** on **Free Triangular** and select **Size**. Click on **Size** and select the area that needs to be meshed. Now you can define the maximum size of a mesh element here as well as other parameters like Resolution of curvature, Resolution of narrow regions under **Element Size Parameters**.

Now, the model is ready !

Study

1. Click on **Step 1: Time Dependent** and in Times, define the time range as: **range(0,maxt/20,maxt)**.

2. Save your file and Click on **Run**.

Literature

- Numerical Methods:
  - Press, Teukolsky Vetterling, Flannery, *Numerical Recipes*, Cambridge University Press, 2007
  - LeVeque, *Finite Differece Methods for Ordinary and Partial Differential Equations*
  - Ralf Hiptmayr, *Numerical Methods for Partial Differential Equations*, lecture notes, 2010
  - Gerald W. Recktenwald, *Finite-Difference Approximations to the Heat Equation*, 2011
- MATLAB & COMSOL (Software and Literature used in the examples):
  - Prof. Iber lecture notes and literature therein
  - Read help files for *pdepe* and *parabolic* in the MATLAB help
  - Browse through available COMSOL models at http://www.comsol.com/showroom/
  - COMSOL manuals

References

1. Lander AD, Nie Q, Wan FYM: **Do morphogen gradients arise by diffusion?** *Dev Cell* 2002, **2**(6):785–796.

2. Eldar A, Rosin D, Shilo BZ, Barkai N: **Self-Enhanced Ligand Degradation Underlies Robustness of Morphogen Gradients**. *Developmental Cell* 2003, **5**(4):635–646.

3. Bollenbach T, Pantazis P, Kicheva A, Bökel C, Gonzalez-Gaitan M, Julicher F: **Precision of the Dpp gradient.** *Development (Cambridge, England)* 2008, **135**(6):1137–1146.

4. D'Errico R: **Interparc function** 2012, [http://www.mathworks.in/matlabcentral/fileexchange/34874-interparc].

5. D'Errico R: **Normal Distance function** 2012, [http://www.mathworks.com/matlabcentral/fileexchange/34869-distance2curve].

6. Cutress IJ, Dickinson EJF, Compton RG: **Analysis of commercial general engineering finite element software in electrochemical simulations.** *J Electroanal Chem* 2010, **638**:76–83.

7. Carin M: **Numerical Simulation of Moving Boundary Problems with the ALE Method: Validation in the Case of a Free Surface and a Moving Solidification Front.** *Excert from the Proceedings of the COMSOL Conference* 2006.

8. Thummler V, Weddemann A: **Computation of Space-Time Patterns via ALE Methods.** *Excert from the Proceedings of the COMSOL Conference* 2007.

9. Weddemann A, Thummler V: **Stability Analysis of ALE-Methods for Advection-Diffusion Problems.** *Excert from the Proceedings of the COMSOL Conference* 2008.