Finite element based simulation of transport limited chemical pattern formation

Tanmoy Sanyal

December 10, 2015

1 Introduction

In this project, I want to apply finite element methods to study transient transport limited pattern formation. It is well known that strong coupling between transport and reaction rate processes in chemical reactions give rise to spatio-temporal distributions of concentrations. These distributions often contain adjacent regions of high and low concentrations and hence form local structures known as *chemical patterns* in the literature. Alan Turing in his famous work,[1] studied systems where diffusion introduces instability in an otherwise steady state to give rise to patterned states, which may then be stabilized by the interplay between diffusion and reaction processes in the system. However, another class of patterns that are of paramount importance, especially in industrial chemical reactors, are the so-called *transport-limited* patterns. In this scenario, reaction rates are disparately huge compared to the speed of diffusion. So, once a steady state loses its stability to perturbations, reactants may be depleted locally at a speed much greater than diffusion gets to replenish that region. In the case of isothermal, autocatalytic reactions, such difference in diffusion and reaction rates often give rise to spotted structures. It may be noted that transport limited patterns are detrimental to product purity in chemical industries and for non-isothermal cases, may often cause very high local hot-spots that may cause meltdown of reactors.

Much of this system is based on my masters' thesis from my integrated BS-MS degree during undergraduate studies. The relevant dynamic simulation studies[2–4] of 3D transport limited patterns had been done previously by my undergrad advisor's group. I will however, consider a much simpler 2D system with no convective flows. Consider the 2D coupled reaction-diffusion equation on a domain Ω:

$$\frac{\partial c_1}{\partial t} = D\nabla^2 c_1 - R(c_1, c_2) \qquad \frac{\partial c_2}{\partial t} = D\nabla^2 c_2 + R(c_1, c_2) \tag{1}$$

$$\partial_n c_1|_{\partial\Omega} = 0 \qquad \partial_n c_2|_{\partial\Omega} = 0 \tag{2}$$

$$c_1(\Omega, 0) = c_{10} \qquad c_2(\Omega, 0) = c_{20} \tag{3}$$

where, c_1, c_2 are the concentration of species A and B, D is the diffusion coefficient for both components and R is the kinetic rate equation. The domain is a disk of unit radius, $\Omega = \{ (x, y) \in [-1, 1] \mid x^2 + y^2 <= 1 \}$ and in this context it actually represents the cross-section of a tubular reactor. Some things to immediately note are:

- Reactant A is depleted in the process while B has a net replenishment given by the same rate expression R
- The diffusion constant is same for both A, B to root out the possibility of any Turing instabilities.

In this project, I consider a simple isothermal, autocatalytic reaction mechanism:

$$A + 2B \rightarrow P + 3B \tag{4}$$

The autocatalysis arises from the fact that B is both a reactant and a product, so more production of B favors the reaction to form more product P. For this reaction, the rate expression can be immediately written down to be:

$$R(c_1, c_2) = k c_1 c_2^2 \tag{5}$$

Finally, the boundary conditions given by Eq. (2) are homogeneous Neumann conditions which basically state that the walls of the reactor are impermeable and do not allow any mass flux through them. In the subsequent sections, I develop a finite element formulation for this pde and apply it to some test problems, before proceeding to study transport-limited patterned states.

2 Variational formulation

The problem discussed in section 1 is essentially a system of coupled diffusion-reaction equations. The general form of such equations can be written as:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{D}\nabla^2\mathbf{u} + \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \qquad \mathbf{x} \in \Omega \text{ x } (0, \infty) \tag{6}$$

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{g_0}(\mathbf{x}, t) \qquad \mathbf{x} \in \Gamma_D \tag{7}$$

$$\partial_n \mathbf{u} = \mathbf{g_1}(\mathbf{x}, t) \qquad \mathbf{x} \in \Gamma_N \tag{8}$$

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x}) \tag{9}$$

where, $\mathbf{u}(\mathbf{x}, t)$ is the vector of species concentrations and $\mathbf{D}$ are their diffusion coefficients. $\mathbf{f}$ is the vector of source functions, which introduces coupling in this system of PDEs. $\mathbf{g_0}$ and $\mathbf{g_1}$ represent spatio-temporally varying Dirichlet and Neumann boundary conditions, defined respectively on the Dirichlet boundary points Γ_D and the Neumann boundary edges Γ_N. The symbol ∂_n denotes the

exterior normal derivative, *i.e.*, $\partial_n \mathbf{u} = \nabla \mathbf{u} \cdot \mathbf{n}$. To keep things simple, I do not incorporate the case of mixed boundary conditions. Note that the source terms in their most general form can be functions of $\mathbf{x}$, $\mathbf{u}$ and t to allow for full non-linearity. In the context of reaction-diffusion problems, $\mathbf{f}$ would represent the reaction rate kinetics.

To avoid running into erroneous notation about the function spaces involved, I present the weak formulation of the scalar version of Eq. (6), which is exactly similar but with a scalar field u, scalar diffusion coefficient D and scalar functions f, g_0 and g_1 for source, Dirichlet and Neumann boundary terms respectively. The variational or weak form of the scalar problem is:

$$\frac{\partial}{\partial t}\int_\Omega d\mathbf{x}\; u \cdot v = -D\int_\Omega d\mathbf{x}\; \nabla u \cdot \nabla v + \int_\Omega d\mathbf{x}\; f v + \int_{\Gamma_N} d\mathbf{x}\; g_1 v \tag{10}$$

$$u(\mathbf{x}, t) = g_0(\mathbf{x}, t) \qquad \mathbf{x} \in \Gamma_D \tag{11}$$

$$v(\mathbf{x}, t) = 0 \qquad \mathbf{x} \in \Gamma_D \tag{12}$$

where, $v(\mathbf{x}, t)$ is a test function. The relevant delimiting spaces for each of the functions in Eq. (2) are:

$$f, g_1 \in \text{Ł}^2(\Omega)\text{ x }(0, \infty) \qquad\qquad v, g_0 \in H^1_{\Gamma_D}(\Omega)\text{ x }(0, \infty) \tag{13}$$

where, $L^2(\Omega) = \left\{ f \colon \Omega \to \mathbb{R} \;\middle|\; \int_\Omega d\mathbf{x}\, |f|^2 < \infty \right\}$, is the space of Lebesgue square-integrable functions.
$H^1_{\Gamma_D}(\Omega) = \{\, v \in H^1(\Omega) \mid v = 0,\ \text{on}\ \Gamma_D \,\}$, where $H^1(\Omega)$ belongs to the family of Sobolev spaces with first order weak derivatives, so that it can be written as: $H^1(\Omega) = \left\{ u \colon \Omega \to \mathbb{R} \;\middle|\; \left(\int_\Omega d\mathbf{x}\, \{|u|^2 + |\nabla u|^2\} \right)^{\frac{1}{2}} < \infty \right\}$.

3 FEM discretization and assembly

I discretize the domain Ω into a mesh of triangles $\mathbb{T}_h$ and use continuous linear polynomials $\mathbb{P}_1 = \{\, a_0 + a_1 x_1 + a_2 x_2 \mid a_0, a_1, a_2 \in \mathbb{R} \,\}$as my shape space. By this, I mean that a function $p(x_1, x_2) \in \mathbb{P}_1$ is uniquely determined by its values on these points. We know that this will give me a nodal basis of *hat* shaped functions, given by:

$$\phi(\mathbf{p}_j) = \delta_{ij} \tag{14}$$

where, $\mathbf{p}_j$ is the j^{th} node of the triangulation. Note that this definition limits the support of $\phi(\mathbf{p}_j)$ to all the triangles that share the vertex $\mathbf{p}_j$. Fig. 1 presents an illustration. The discrete solution u_h can thus be expanded in the nodal basis ϕ as, $u_h = \sum_{j=1}^{N} u_j \phi_j$.

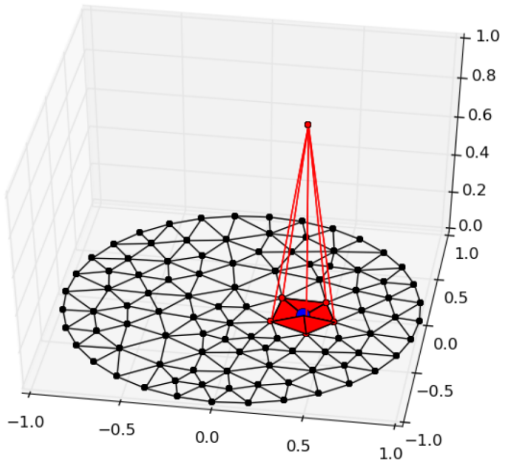

Figure 1: Support of a linear *hat* function on a triangular mesh

Plugging this converts the equation system in Eq. (10) to the linear system:

$$\mathbf{M}\frac{\mathbf{dU}}{dt} = -D\mathbf{WU} + \mathbf{F} + \mathbf{G},\ \ \mathbf{U}^D(\mathbf{p}_j^D) = g_0(\mathbf{p}_j^D, t) \tag{15}$$

where, $\mathbf{U}$ is the vector of solution values at mesh nodes $\{\mathbf{p}\}$ while $\mathbf{U}^D$ are the known values at Dirichlet nodes $\mathbf{p}_j^D$. The other terms appearing in this equation are the mass matrix $\mathbf{M}$, the stiffness matrix $\mathbf{W}$, the source vector $\mathbf{F}$ and the Neumann boundary value vector $\mathbf{G}$. These are basically matrix form of the integrals that appear in Eq. (10). They are given by:

$$\mathbf{M}_{ij} = \int_\Omega d\mathbf{x}\ \phi_j\phi_i \qquad = \sum_{K\in\mathbb{T}_h}\int_K d\mathbf{x}\ \phi_j\phi_i \tag{16}$$

$$\mathbf{W}_{ij} = \int_\Omega d\mathbf{x}\ \nabla\phi_j\nabla\phi_i \qquad = \sum_{K\in\mathbb{T}_h}\int_K d\mathbf{x}\ \nabla\phi_j\nabla\phi_i \tag{17}$$

$$\mathbf{F}_j = \int_\Omega d\mathbf{x}\ \phi_j \qquad = \sum_{K\in\mathbb{T}_h}\int_K d\mathbf{x}\ f\phi_j \tag{18}$$

$$\mathbf{G}_j = \int_{\Gamma_N} d\mathbf{x}\ g_1\phi_j \qquad = \sum_{L\in\Gamma_N}\int_L d\mathbf{x}\ g_1\phi_j \tag{19}$$

These matrices are global integrals over the entire domain Ω and its Neumann boundaries Γ_N. However, as shown here, they can be decomposed into local contributions from individual triangles K and summed up. The process of summing up is formally referred to as *assembly* and often constitutes the most time consuming step in finite element based solutions of PDEs. For a triangular element K, let $\mathbf{p}_1^K = (x_1, y_1)^K, \mathbf{p}_2^K = (x_2, y_2)^K, \mathbf{p}_3^K = (x_3, y_3)^K$ be the vertices and $\phi_1^K, \phi_2^K, \phi_3^K$ be the corresponding nodal basis functions. Then from Eq. (1), I require that, $\phi_j^K(x_k^K, y_k^K) = \delta_{jk}$, $j, k = 1, 2, 3$. The linear function $\phi(x, y)$ that satisfies this requirement can be immediately constructed as:

$$\phi_j^K(x, y) = \frac{1}{2|K|} \begin{bmatrix} 1 & x & y \\ 1 & x_{j+1}^K & y_{j+1}^K \\ 1 & x_{j+2}^K & y_{j+2}^K \end{bmatrix} \tag{20}$$

where, $|K|$ is the area of the triangle K. Further, to calculate the integrals on their respective domains (triangle element or Neumann boundary edge), I resort to some simple approximate quadratures, namely:

$$\int_K d\mathbf{x}\, \phi \approx \frac{|K|}{3} \left(\phi(\mathbf{p}_1^K) + \phi(\mathbf{p}_2^K) + \phi(\mathbf{p}_3^K) \right) \tag{21}$$

$$\int_L d\mathbf{x}\, \phi \approx \frac{|L|}{2} \left(\phi(\mathbf{p}_1^L) + \phi(\mathbf{p}_2^L) \right) \tag{22}$$

where, $|L|$ is the length of the Neumann edge L. Using $\phi(x, y)$ from Eq. (20) and the two quadrature rules, the final form of the local matrices over the individual elements in Eqs. (16) - (19) works out to be:

$$\mathbf{W}_K = \frac{|K|}{2} \cdot GG^T \ , G = \begin{bmatrix} 1 & 1 & 1 \\ x_1^K & x_2^K & x_3^K \\ y_1^K & y_2^K & y_3^K \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{23}$$

$$\mathbf{M}_K = \frac{|K|}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \tag{24}$$

$$\mathbf{F}_K = \left[\frac{|K|}{9} \sum_{j=1}^{3} f\left(\mathbf{p}_j^K\right) \right] \cdot \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T \tag{25}$$

$$\mathbf{G}_L = \frac{|L|}{4} \left(g_1(\mathbf{p}_1^L + \mathbf{p}_2^L \right) \tag{26}$$

The element matrices are added for each triangle K in the mesh to form the global matrices. A simple pseudocode illustrates the process for the stiffness matrix.

Algorithm 1 *Assemble stiffness matrix* $\mathbf{W}$ *over mesh*

$\mathbf{W} \leftarrow$ *empty matrix*(N, N)
$\mathbf{W}_K \leftarrow$ *empty matrix*$(3, 3)$
for $K \in \mathbb{T}_h$ **do**
  $\mathbf{W}_K \leftarrow$ Eq. (23)
  for $\mathbf{p}_i \in K$ **do**
    $N_I \leftarrow T(\mathbf{p}_i)$
    for $\mathbf{p}_j \in K$ **do**
      $N_J \leftarrow T(\mathbf{p}_j)$
      $\mathbf{W}(N_I, N_J) \leftarrow \mathbf{W}(N_I, N_J) + \mathbf{W}_K(i, j)$
    end for
  end for
end for

end

where, N_I, N_J are the global node numbers of the nodes $\mathbf{p}_i, \mathbf{p}_j$. So far, I have used the scalar form of the coupled multicomponent PDE system in Eq. (6) to illustrate the various calculation steps. However, it is elementary to extend the linear system in Eq. (15) to incorporate a vector field $\mathbf{u}$. It would look like:

$$\mathbf{M}_{all}\frac{d}{dt}\mathbf{U}_{all} = -\mathbf{W}_{all}\mathbf{U}_{all} + \mathbf{F}_{all} + \mathbf{G}_{all},\ \mathbf{U}^D_{all}(\mathbf{p}^D_j) = \mathbf{g}_0(\mathbf{p}^D_j, t) \tag{27}$$

where, $\mathbf{M}_{all}, \mathbf{W}_{all}$ are block matrices formed out of the individual matrices $\mathbf{M}, \mathbf{W}$ respectively, while $\mathbf{X}_{all}$, $(\mathbf{X} = \mathbf{U}, \mathbf{F}, \mathbf{G})$ are vectors formed by joining end to end the corresponding vectors for the individual components. Thus, *e.g.*, in a system with 2 components (similar to the one I am interested in studying), the overall matrices look like:

$$\mathbf{M}_{all} = \begin{bmatrix} \mathbf{M}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_2 \end{bmatrix} \qquad \mathbf{W}_{all} = \begin{bmatrix} D_1\mathbf{W}_1 & \mathbf{0} \\ \mathbf{0} & D_2\mathbf{D}_2 \end{bmatrix} \qquad \mathbf{X}_{all} = \begin{bmatrix} \mathbf{X}_1 \mathbf{X}_2 \end{bmatrix}^T$$

4 Time integration

To keep the notation simple, I shall be writing Eq. (27) without the suffix *all*. This is a first order ODE that can be solved using a variety of schemes, perhaps using fast ODE solver packages like the Sundials suite of solvers. For the purposes of this project, however, I shall use a very simple forward Euler scheme for the time integration of Eq. (27). The choice of the method is motivated by its explicit formulation. Implicit solvers like Backward Euler or the Runge-Kutta family may provide superior accuracy but also come at a much larger computational cost. Moreover, if I time-discretize the equation with the forward Euler approach, $y(t+1) = y(t) + \Delta t f(y,t)$, it can be seen that the resulting system is not entirely explicit.

$$\mathbf{M}^{n+1}\mathbf{U}^{n+1} = \mathbf{M}^n\mathbf{U}^n - \Delta t_n \mathbf{W}^n\mathbf{U}^n + \Delta t_n \mathbf{F}^n + \Delta t_n \mathbf{G}^n,\ \mathbf{U}^{n+1,D}(\mathbf{p}_j^D) = \mathbf{g}_0(\mathbf{p}_j^D, n+1) \tag{28}$$

At every time step Δt_n, I still need to solve a linear system and that imparts some sort of *implicit-ness* to the system. However, for the forward Euler method, stability of the numeric solution is of concern. The forward Euler method is not stable for any arbitrary choice of Δt_n and it can be shown that for this system, the stability is a function of stepsizes in both space and time. I present a simplified analysis here.

Consider a coupled PDE system like the one in Eq. (6) with zero source term and homogeneous Neumann boundary conditions and no Dirichlet conditions. Further assume that the mass and stiffness matrix do not change (this would be the case when there is no re-meshing during the time integration process) across time iterations and the time-step is uniform *i.e.*, $\Delta t_n = \Delta t\ \forall n$. These assumptions reduce Eq. (27) to:

$$\mathbf{M}\mathbf{U}^{n+1} = \mathbf{M}\mathbf{U}^n - \Delta t \mathbf{W}\mathbf{U}^n \tag{29}$$

The stiffness and mass matrices can be related through the eigenvalues of the stiffness matrix as, $\mathbf{W}\phi_k = \lambda_{h,k}\mathbf{M}\phi_k$. Plugging this into Eq. (29) and solving the consequent recursion gives us

$$\mathbf{u}^n = \sum_{k=1}^{N(h)} c_k \phi_k \left(1 - \lambda_{h,k}\Delta t\right)^n$$

where, c_k are the coefficients for expanding the solution u in the basis of the nodal values of the discrete eigenvectors ϕ_k of the stiffness matrix $\mathbf{W}$. For no sources, diffusion will dissipate any initial condition so that after sufficiently long time $\mathbf{u} \to 0$. This can only happen when,

$$|1 - \lambda_{h,k}\Delta t| < 1 \tag{30}$$

$$\implies \Delta t\ Max(\lambda_{h,k}) < 2 \tag{31}$$

Thus, the size of the mesh and the time step are not independent of each other but must be chosen in accordance with this condition. In this project, I work with meshes of fixed size, so that my time step is adjusted to meet the stability criterion.

Finally, note that since I already know the solution values at the Dirichlet nodes, I need solve only for the other nodes. This can be done by maintaining lists or similar data structures of Dirichlet nodes where the solution is constrained to the known values and free nodes where the linear system needs to be solved. Eq. (28) can then be written as one single linear system:

$$\mathbf{M}^{n+1}_{free}\mathbf{U}^{n+1}_{free} = \mathbf{M}^{n}_{free}\mathbf{U}^{n} - \Delta t_n \mathbf{W}^{n}_{free}\mathbf{U}^{n} + \Delta t_n \mathbf{F}^{n}_{free} + \Delta t_n \mathbf{G}^{n}_{free} - \mathbf{M}^{n}_{Dir}\mathbf{U}^{n}_{D} \tag{32}$$

where, the suffix *free* refers to the non-Dirichlet or unconstrained or free nodes and the *Dir* refers to the Dirichlet nodes. Thus if there are I free nodes and D Dirichlet nodes, then, for a single component, the matrices and vectors in the above equation have shapes: – $\mathbf{M}_{free} : I\text{x}I, \mathbf{W}_{free} : I\text{x}I, (\mathbf{U}_{free}, \mathbf{F}_{free}, \mathbf{G}_{free}) : I\text{x}1$ and $\mathbf{M}_{Dir} : I\text{x}D, \mathbf{U}^{D} : D\text{x}1$.

5 A brief walk through the code

The code is written to be general and modular, allowing for easy future extensions and portability. It is written almost entirely in python with the exception of the solution of linear algebra systems which is bundled through Fortran and makes use of the efficient linear solver `DGSEV()` from the Linear Algebra Package (LAPACK). I have developed two main libraries - **fem**, that manages the meshing and calculation and assembly of local (or element wise) matrices using the $\mathbb{P}_1$ class of polynomials, and **pde**, that contains class definitions and method prototypes for global assembly of source terms and boundary conditions for multicomponent systems.

The major definitions in the **fem** module are:

- Mesh class - Contains methods to set Dirichlet and Neumann boundary conditions and to refine the mesh based on a simple recursive bisection of all the triangles. My mesh data structure consists of 2D arrays that carry the node co-ordinates and lists of node indices that constitute the triangles. These lists serve as connections between local and global numbering of nodes. I further maintain lists of node indices corresponding to Dirichlet nodes or Neumann edges.
- ShapeFn class - Contains methods to calculate the local stiffness and mass matrices and source and Neumann edge vectors based on the expressions in Eqs. (23) - (22).
- Assemb class - Contains methods to assemble the global stiffness and mass matrices over the entire mesh, for a single component, using algorithm 1. The stiffness and mass matrices are usually sparse because of the limited support of individual nodal basis functions ϕ^K and so sparse matrix based storage schemes can be leveraged for faster computation. In this first version of the code, I have not resorted to either using sparse storage or vectorizing the assembly algorithm. However, I have confirmed that for the relatively coarse meshes that I work with in this project, these naive unoptimized approaches work well.

- Plot class - Contains functions for plotting the mesh and nodal basis functions. Also there are utilities for plotting 2D filled contour plots of the solution $\mathbf{u}$ and generating time-lapse animations of the time varying solution $\mathbf{u}(\mathbf{x}, t)$.

The composition of the **pde** module is:

- PDE class - This serves as the base class that encapsulates methods for assembling the source, Dirichlet and Neumann boundary conditions on the entire mesh, for a single component. Further it defines methods for constructing block matrices like $\mathbf{W}_{all}$ and $\mathbf{M}_{all}$ as discussed in Eq. (27) and for constructing the final LHS of the linear algebra problem that should be solved. Since both these operations are highly problem specific, only their prototypes are defined and can be written by the user to suit his need.
- Elliptic class - This inherits from PDE and defines its own assembly method in which it calls the various methods of PDE to assemble the final linear algebra problem.
- Parabolic class - Similar in flavor to Elliptic; defines its own assembly.

It may be noted that the initial coarse mesh is generated using the `initmesh` tool from MATLAB, which is then encapsulated into the data structures provided by the fem.Mesh class and refined. The entire code can be found online on my github repository `https://github.com/tanmoy7989/FEM_course_project.git`

6 Tests

I conducted several tests on different parts of the code while developing it. The complete test bench can be found in `femlib/test.py` in the github repository. The two major tests that I conducted with the code was to extract error scaling for a known solution of a simple Poisson's equation on a disk and a single component diffusion only problem with no sources or sinks.

6.1 Poisson's equation test

I ran a preliminary test for a Poisson equation on a disk with constant source term and homogeneous Dirichlet boundary conditions, so that the error could be compared to the analytically available solution. I chose the exact solution to be:

$$u(x, y) = \frac{1 - x^3 - y^3}{6}$$

so that the PDE becomes,

$$\nabla^2 u = -(x + y), \ , \Omega = \left\{ \ (x, y) \in [-1, 1] \ \middle| \ x^2 + y^2 <= 1 \ \right\}$$

This was solved using a triangular mesh of diameter 0.1 (generated in MATLAB) with 541 nodes and 1016 triangles. The result:

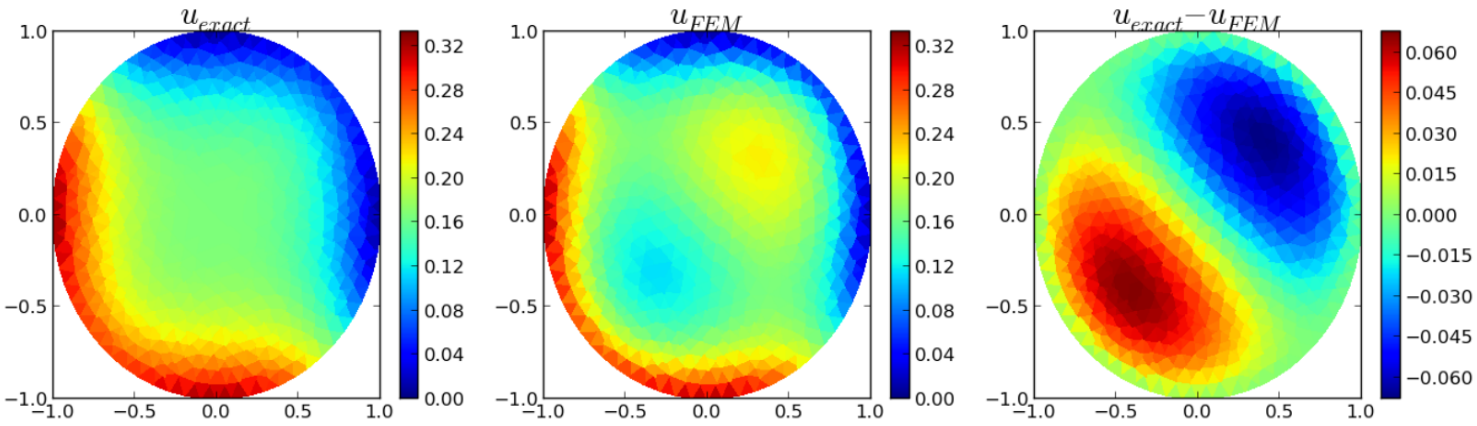

Figure 2: FEM solution of the test Poisson problem $\nabla^2 u = -(x + y)$

To find out the error scaling, I solved this problem on meshes of different diameters $0.04, 0.06, 0.08, 0.1, 0.4, 0.5, 0.8$ and fitted the error data by linear regression. The scaling however comes out to be of order 1 which is not quite right, since linear polynomial shape functions on triangular elements are known to be of order 2. **This part needs some re-working, to find out where exactly I**

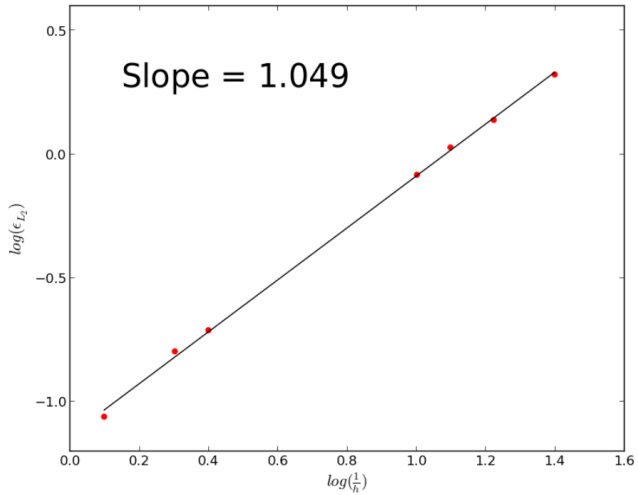

Figure 3: Error scaling for the test Poisson problem. $log(\in_{L2})$ is plotted against $log(h^{-1})$

am going wrong. However, the exact and FEM solutions are in reasonably OK agreement, as seen in Fig. 2. So, before moving on,

I performed one more test for Poisson problem, $\nabla^2 u = 0$ with homogeneous Dirichlet conditions. The exact solution for this problem is $u(x, y) = \frac{1-x-y}{4}$. The error magnitude from the colorbars in Fig. 4 shows that the exact and FEM solutions are in extremely good agreement. The Poisson test is contained in two function `testPoisson()` and `testPoissonErrorScaling()` in the test bench.

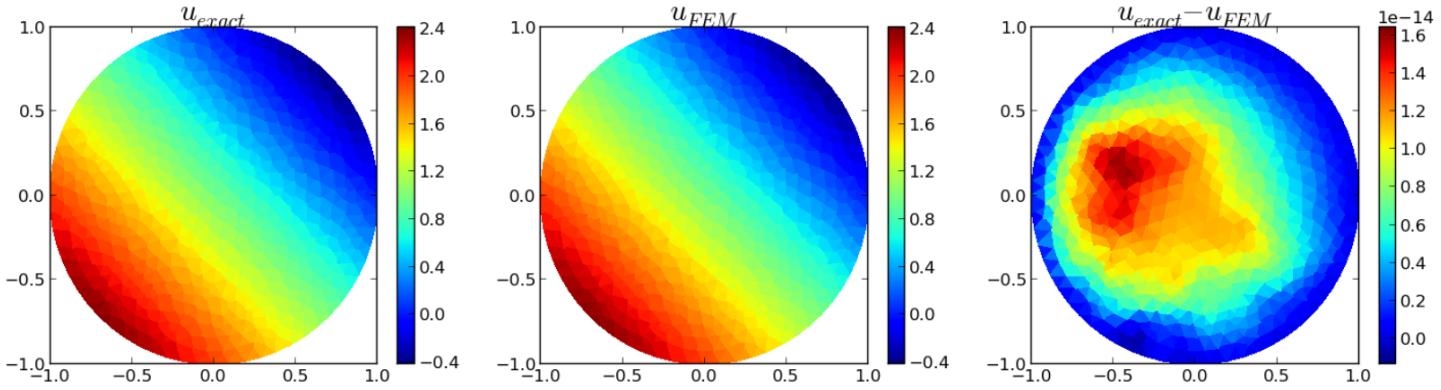

Figure 4: FEM solution of the test Poisson problem $\nabla^2 u = 0$

6.2 Diffusion equation test

My second test consists of a simple 1 component diffusion equation with no sources and sinks and homogenous boundary conditions, *i.e*,

$$\begin{aligned}
&\frac{\partial u}{\partial t} = -D\nabla^2 u \\
&\nabla u \cdot \mathbf{n} = 0, \quad \{\, (x, y) \mid x^2 + y^2 = 1 \,\} \\
&u(x, y, 0) = u_0 \delta(x, y)
\end{aligned}$$

The initial condition is chosen to be a simple point source described by the Dirac Delta function at the origin. In practice, I simulated the Dirac Delta with a step function:

$$u(x, y) = \begin{cases} u_0 & x^2 + y^2 <= 0.01 \\ 0 & x^2 + y^2 \in (0.1, 1] \end{cases}$$

To be able to visualize the time varying solution $u(x, y, t)$, I bumped up the diffusion coefficient to a high value $D = 100$, while keeping the initial concentration low, $u_0 = 1$. Also, since I worked with the same mesh as the Poisson test problem, which is reasonably

fine grained, the time-step for forward Euler integration, in accordance with the stability requirement presented in section 4 was maintained at a low value of $dt = 0.0005$. After running a total of 100 time iterations, the snapshots of $u(x, y, t)$ at different time instants look like:

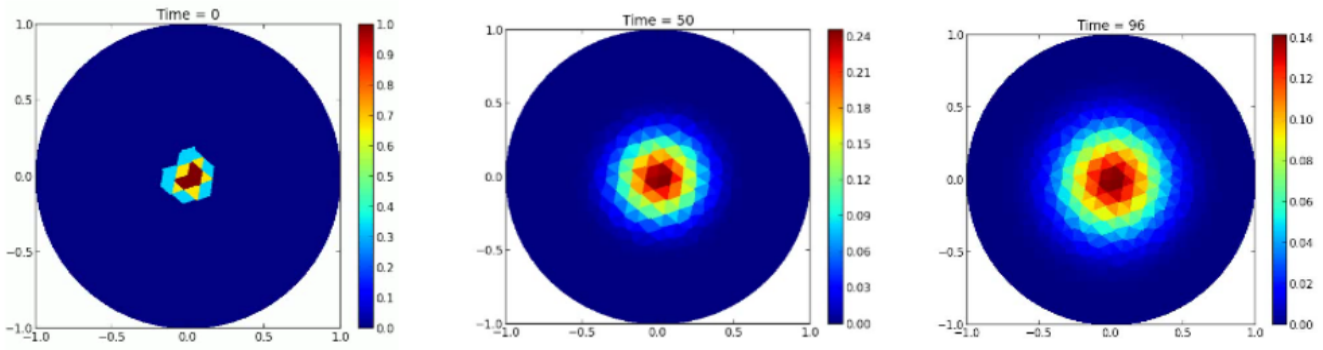

Figure 5: Time varying FEM solution of the test diffusion problem $u_t = D\nabla^2 u$

The classic problem of homogeneous diffusion of a point source leads to a steady state solution which is uniform across the domain, since diffusion forces spread away the entire initial intensity concentrated at the origin. This means that if I kept running this simulation for a very long time, I would eventually observe $u = 0$ everywhere. To make sure that my time integration scheme captures this phenomenon, I plotted $\|u^{n+1} - u^n\|$ for different time instants. Fig. 6 shows that this indeed is the case, and the forward Euler scheme, with proper Δt is reliable.

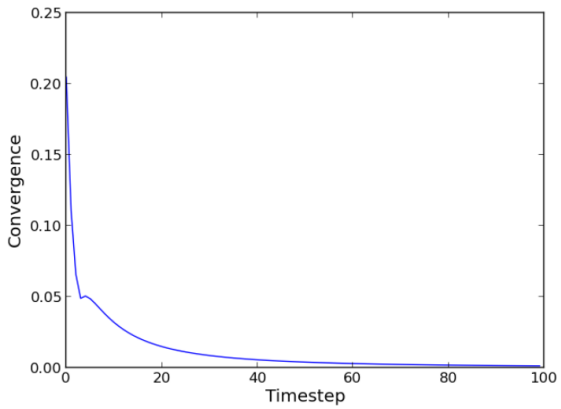

Figure 6: Convergence towards steady state, captured by difference in norms of the solution between successive time points

It is known that the forward Euler method has order $O(\Delta t)$. However, I am not entirely sure of how to test this first order error scaling for my test problem. One possible approach can be to compute the exact solution (which will involve Bessel functions, if done in plane polar co-ordinates) and then compute the error between the exact solution and FEM solution at a particular instant, and repeat this process for several time-steps. However, I am not convinced that presenting such an analysis for a particular instant is sufficient. Moreover, how does this procedure translate to nonlinear systems where exact solutions cannot be computed ? In light of the reliable *approach-to-steady-state* behavior exhibited in Fig. 6, I decided to move on to the application of these methods to the dynamic simulation of patterns. The diffusion test is contained in `testLinDiff()` in the test bench.

7 Dynamic simulation of patterns

Before applying the finite element method to the dynamic simulation of transport limited patterns, it is important to know possible values of the parameters involved. Consider Eq. (1) again. The relevant parameters here are D, k, c_{10}, c_{20}. These parameters cannot have arbitrary values if we want to observe pattern formation. I shall develop a very elementary linear stability analysis here to calculate the possible values. Ideally what is needed, is a bifurcation theory based treatment, but it is too involved for the purposes of this course project and so I shall not go into it. I rewrite the equations here:

$$\frac{\partial c_1}{\partial t} = D\nabla^2 c_1 - k c_1 c_2^2 \qquad\qquad \frac{\partial c_2}{\partial t} = D\nabla^2 c_2 + k c_1 c_2^2$$

with initial conditions $c_1(x, y, 0) = c_{10}$ an $c_2(x, y, 0) = c_{20}$, which are uniform *i.e.*, non-patterned. Patterned states can start developing if these initial conditions are perturbed. Consider small perturbations of the form:

$$c_i = c_{i0} + f_i \tag{33}$$

$$f_i = e^{\lambda t}\, e^{\iota m\theta} J_m\left(k_{mn} r\right), \quad i = 1, 2 \tag{34}$$

where, λ is some characteristic time constant that captures the time decay of the perturbation, J_m is the Bessel function of order m. To maintain homogeneous Neumann boundary conditions, k_{mn} can be calculated from the condition that:

$$\frac{d}{dr} J_m\left(k_{mn} r\right)|_{r=1} = 0$$

The choice of Bessel functions in plane polar co-ordinates (r, θ) is motivated by the fact that they are the natural eigenfunctions in cylindrical co-ordinates. (m, n) is formally called the *mode* of the perturbation. Linear stability analysis then consists of linearizing the non-linear reaction rate terms given by Eq. (5) about their uniform initial values as:

$$R = k c_1 c_2^2 \approx k c_{10} c_{20}^2 + k c_{20}^2 (c_1 - c_{10}) + 2k c_{10} c_{20} (c_2 - c_{20})$$

$$\implies R \approx k c_{10} c_{20}^2 + k c_{20}^2 f_1 + 2k c_{10} c_{20} f_2$$

and plugging it into Eq. (1). The corresponding equations can be collected and written in matrix form as:

$$\begin{bmatrix} \lambda + k_{mn}^2 D + k c_{20}^2 & 2k c_{10} c_{20} \\ -k c_{20}^2 & \lambda + k_{mn}^2 - 2k c_{10} c_{20} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = k c_{10} c_{20}^2 \begin{bmatrix} - & 1 \\ & 1 \end{bmatrix}$$

The *neutral stability* condition then demands non-trivial solutions of this system for $\lambda = 0$. That basically translates to the determinant of this matrix being 0 and working through the algebra gives the criterion:

$$k_{mn}^2 D + (c_{20}^2 - 2k c_{20} c_{10}) k = 0 \tag{35}$$

Thus, with a given set of uniform initial conditions, reaction rate and diffusion coefficient for the system must satisfy Eq. (35) to be on the verge of forming patterned states. Fig. 7 illustrates this fact. For the perturbation mode $(m, n) = (0, 1)$ (eigenvalues were

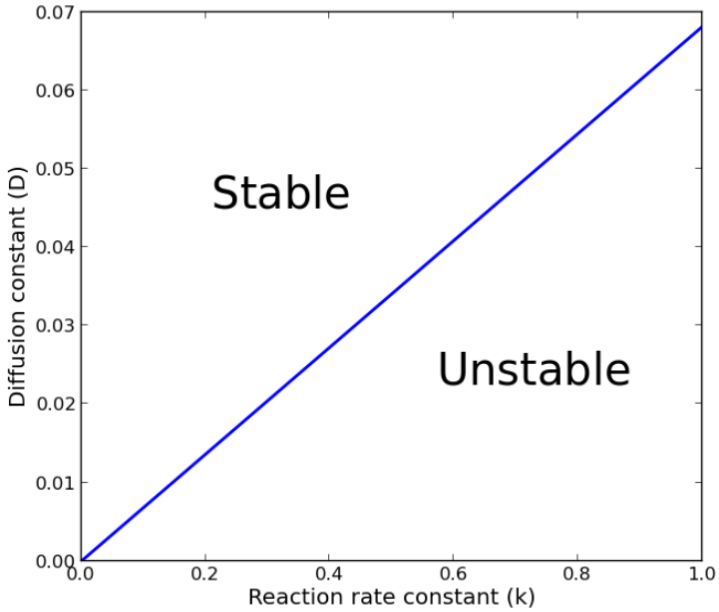

Figure 7: Neutral stability line for the perturbation mode $(m, n) = (0, 1)$

taken from the online resource Wolfram Alpha[5]), the only values of (k, D) that will lead to pattern formation lie *below* the neutral stability line in Fig 7. This is interesting because it reinforces the idea that transport limited patterns are associated with high rates of reaction. For low rate constant k, even a moderately high diffusion coefficient will result in the "Stable" marked region in 7 and

lead to quick wash-out of any patterns that might form. For higher rates, the allowed spectrum of D values becomes larger. The implications of this in an industrial setting is exactly the reverse: for moderately low rate kinetics, diffusion instabilities in the system may still fizz out leading to little or no pattern formation in the reactor, but for large rates, even the slightest disturbance can quickly lead to pattern formation and degrade product.

With the parameters (k, D) sorted out, I now proceed to the actual simulation. The relevant equation system is once again Eq. (1) with boundary conditions given by Eq. (2). The initial conditions however, represent Bessel perturbations given by Eq. (33). The recipe I follow is:

- Fix initial uniform values for c_{10}, c_{20} and reaction rate constant k
- Pick a Bessel mode (m, n) with eigenvalue k_{mn}
- Use the neutral stability line (Eq. (35)) to pick a suitable value for D
- Set up a run with k, D, and initial conditions given by f_i from Eq. (33)

The script that performs the simulations is `simpattern.py` on the github repository. For my pattern simulations, I choose the parameters:

- $c_{10} = 1.0$, $c_{20} = 0.05$. This reflects the case that A is our major reactant while B is an autocatalyst that regenerates itself and so need only be present in very small quantities, initially
- $k = 10$, a reasonably high rate constant. The diffusion coefficient D is first calculated from Eq. (35), and then brought down by a factor of 10, because D close to the neutral stability line will make the initial perturbation static and extremely resilient to wash-out so that there would effectively be no animation.
- Bessel modes $(m, n) = (0, 1), (1, 1), (3, 3)$. The first two are low asymmetric modes I chose at random, while the choice for the third mode is motivated by the fact that $J_3(k_{33})$ will have a lot of cusps and hence extensive local structure across the pattern. It will be interesting to see the interplay between diffusion and reaction on these local structures within the pattern. The value of the initial perturbation is further watered down to 3% just to make sure that it is not large enough to cause pattern formation by itself.
- Triangular mesh of diameter 0.1 with 541 nodes and 1016 triangles.
- Integration time-step $\Delta t = 0.0005$, which was chosen by trial and error and gives reasonable stability to the forward Euler method.

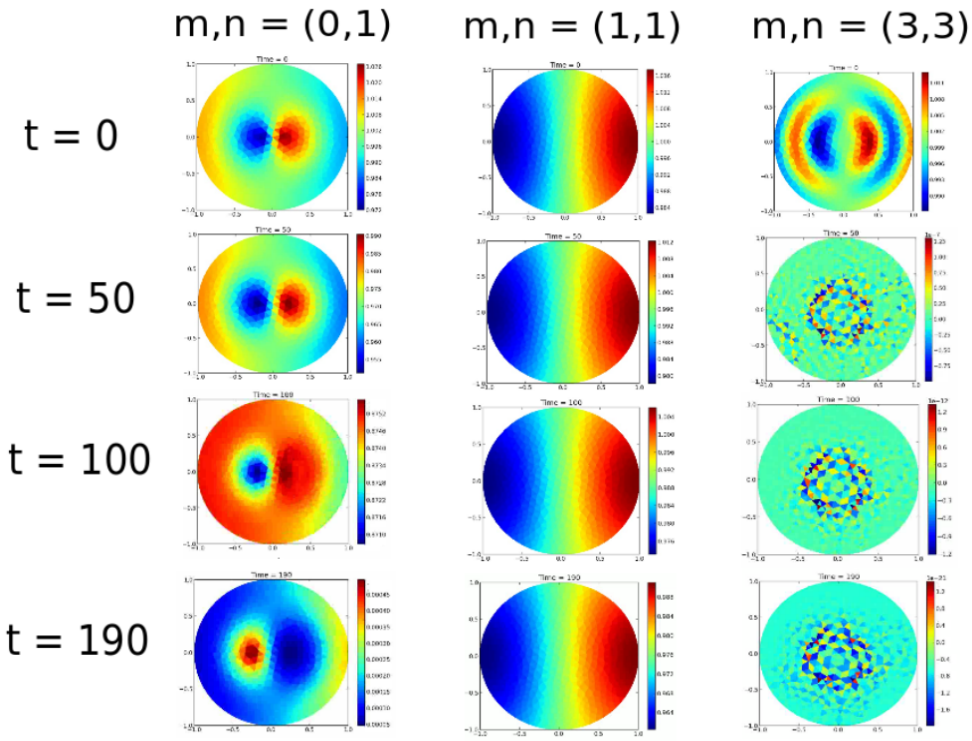

Figure 8: Dynamic simulations of transport limited chemical patterns for perturbation modes $(m, n) = (0, 1), (2, 1), (3, 3)$

Running for 200 time steps on a Intel 8-core system (my laptop) gives me Fig. 8 Here, it shows that the $(1, 1)$ mode is somewhat resistant to diffusion forces, $(0, 1)$ is moderately more pliable while $(3, 3)$ completely washes out in less than 50 time steps. Careful observation shows that the $(1, 1)$ has lesser structure while the other two perturbations have significant local structuring (high and low regions of concentrations) within the pattern. In my masters' thesis, I had conjectured that, in patterns with more structure, regions of alternate high and low concentrations are closer and hence it is easier for diffusion forces to wash them away. However, I shall not go into those details in this project.

8 Conclusion and future prospects

In this project, I have built a library that uses triangular *Linear Lagrange* finite elements and forward Euler time integration schemes to tackle transient diffusion reaction problems. However, as I mentioned earlier in section 5, the code is extremely modular and amenable to easy extension and addition of higher order shape-spaces or higher order time-integration schemes with little or no change to the basic framework. Immediate areas of improvement are:

- Using sparse storage for the stiffness and mass matrices to be easy on memory resources.
- Vectorizing and possibly, parallelizing the assembly process for speedup.
- Using a higher order time-integration scheme like RK4. In fact, it would be ideal to use Fortran or C based fast ODE solver packages like Sundials. That may ensure reliability in a lot of very real life applications of this library.
- Exploring the possibility of solving hyperbolic PDEs like the 2D wave equation or Schrodinger's equation. Such systems can be difficult to tackle without adaptive re-meshing across time integration steps. The speedup schemes discussed above would then play an important role.

References

[1] Alan M. Turing. *The chemical basis of morphogenesis*. Philos. T. R. Soc B. (1952) 237: 37-72

[2] Ankur Gupta, Saikat Chakraborty. *Linear stability analysis of high- and low-dimensional models for describing mixing-limited pattern formation in homogeneous autocatalytic reactors*. Chem. Eng. J. (2009) 145: 399-411

[3] Anwesha Chaudhury, Saikat Chakraborty. *Dynamic simulation of mixing-limited symmetric and asymmetric patterns in homogeneous autocatalytic reactors*. Ind. Eng. Chem. Res. (2010) 49(**21**): 10517-23

[4] Tanmoy Sanyal and Saikat Chakraborty. *Stability of mixing limited patterns in homogeneous autocatalytic reactions*. Poster presented at AICHE 2013, San Francisco, CA.

[5] `http://mathworld.wolfram.com/BesselFunctionZeros.html`