Gas Transport In Porous Medium

Members: Moussa Atwi and Surya Narayanan Iyer

Supervisor: Dr. Jürgen Fuhrmann

Group 10: Master in Scientific Computing

ID: 464241 and 464252

Grade: Excellent 1.0

Acknowledgement

We would like to thank Dr. Jürgen Fuhrmann for allowing us to work on this very interesting topic. While researching and writing code for simulation, we learned about many different methods for solving partial differential equations numerically, some of which we hope to use in the future for different projects. Any doubts that we had were cleared in a timely manner by him, and the insights he offered were put to good use.

General Overview

Let $m > 1$ and $\Omega \subset \mathbb{R}^d$ be a polygonal bounded domain (i.e., Lipschitz domain with piecewise smooth boundary). Choose final time $T > 0$, regard functions $u(x,t) \to \mathbb{R}$. Consider the transient parabolic IBV problem $(*)$

- $u_t - \nabla.(D\nabla u) = 0, \quad$ in $\Omega \times [0,T]$
- Initial condition: $u(x,0) = u_0(x), \quad$ in Ω $\qquad (*)$
- Boundary conditions: $D\nabla u \cdot \vec{n} + \alpha u = \beta \quad$ on $\quad \partial\Omega \times [0,T]$

where, $\alpha, \beta \geq 0$ and $D(u) = mu^{m-1}$.

Our project focuses on this **nonlinear diffusion problem** and implements solutions by using the **VoronoiFVM.jl** package. This package provides a solver for coupled nonlinear partial differential equations based on the **Voronoi finite volume method**, which gives us a finite dimensional approximation of our problem.

The first half of this report describes the theory, mathematical and physical background, as well as some topics covered throughout this course. The overview is as follows:

I. Discretization overview

1. Mesh Generation: 1D and 2D discretization grids using **ExtendableGrids.jl**;
2. Perform a FD discretization in time, then perform a FV discretization in space (**Rothe Method**); and finally
3. Time discretization: We choose the $\theta-$Scheme (**Implicit Euler Method**).

II. Handling the nonlinear system of equations

- To figure out how we solve the nonlinear system we look for desirable **matrix properties**.
- For **stability** reasons, we **downgrade** our problem to **Neumann BV problem**.
- As we have a discrete problem with some nonlinearity, the **VoronoiFVM.jl** package uses the **Newton iteration scheme** to solve the nonlinear system of equations.

The second half of this report focuses solely on the numerical results for our problem.

III. Simulation results

We present the following numerical results, each with the three different types of boundary conditions:

1. 1D Case, i.e, in the space-time domain $(-1,1) \times (t_0, t_1)$;
2. 2D Case, i.e, in the space-time domain $(-1,1)^2 \times (t_0, t_1)$

The results include solution plots, error in each case, and the effect of grid spacing on the error of the solution.

IV. Optional results

We investigate performance improvement using the **DifferentialEquations.jl** solver for both the 1D and 2D case.

Introduction

Mathematical Background

The *Porous Medium Equation* is a degenerate nonlinear parabolic PDE of second order

$$u_t - \Delta(u^m) = 0, \quad m > 1. \qquad (1)$$

It is natural to consider the nonnegative solutions from a physics standpoint(as we are dealing with density, temperature, etc.), so if we do not restrict ourselves to the study of nonnegative solutions, then for signed solutions our diffusion equation should be written as

$$u_t - \Delta(|u|^{m-1}u) = 0, \quad m > 1$$

One can write (1) in a more general form:

$$u_t = \nabla.(mu^{m-1}\nabla u), \quad m > 1$$

- For $m > 1$: (1) degenerates at $u = 0$, thus we have slow diffusion.
- For $m < 1$: (1) is singular at $u = 0$ where $\frac{m}{u^{1-m}} \to \infty$ as $u \to 0$, thus we have fast diffusion.
- Assuming $u \geq 0$, equation (1) is formally parabolic, only at points where u is strictly positive.

In particular:

1. For $m = 1$ we have the classical heat equation

$$u_t = \Delta u$$

2. For $m = 2$ we have Boussinesq's equation (groundwater infiltration)

$$u_t = \Delta u^2$$

Solution of Boussinesq's Equation

Boussinesq's equation can be solved in 1D using **separation of variables**. Assume,

$$u(x,t) = v(t)w(x),$$

which implies

$$(v(t)w(x))_t = \Delta\big(v^2(t)w^2(x)\big) \quad \text{and} \quad v'(t)w(x) = v^2(t)\Delta\big(w^2(x)\big).$$

This yields

$$\frac{v'(t)}{v^2(t)} = \frac{\Delta(w^2(x))}{w(x)} = \lambda = \text{constant}.$$

- For t we get: $\left(-\frac{1}{v(t)}\right)' = \lambda$ which implies

$$v(t) = \frac{1}{-\lambda t + c}$$

- For x we get:

$$\Delta(w^2(x)) = \lambda w(x).$$

Let us guess the particular solution, $w(x) = |x|^r$ for some r. Then,

$$0 = \lambda w(x) - \Delta(w^2(x)) = \lambda|x|^r - \Delta(|x|^{2r}),$$

that gives

$$\lambda|x|^r = 2r(2r + d - 2)|x|^{2r-2}.$$

Hence,

$$r = 2 \quad \text{and} \quad \lambda = 4(d+2)$$

Therefore,

$$u(x,t) = v(t)w(x) = \frac{|x|^2}{-\lambda t + c} = \frac{|x|^2}{-4(d+2)t + c}.$$

Barenblatt Solution

The effect of the nonlinearity in (1) is that there are solutions with compact support. Equation (1) has a radially symmetric exact solution in $\mathbb{R}^d \times (0, \infty)$, the so-called $Barenblatt$ solution

$$b(x,t) = \max\left(0, t^{-\alpha}\left(1 - \frac{\beta(m-1)r^2}{4m\,t^{\beta}}\right)^{\frac{1}{m-1}}\right), \qquad (2)$$

Where,

$$r = |x|, \quad \beta = \frac{2\alpha}{d} \text{ and } \alpha = \frac{1}{m - 1 + \frac{2}{d}}$$

.

Barenblatt solution takes as initial data a Dirac mass: as $t \to 0, b(x,t) \to M\delta(x)$, where M is a function of the free constants m and d.

This solution spreads a finite amount of mass over the space domain, thus for any fixed $t \in (0,T),\ b(x,t)$ has a compact support

$$|x|^2 \leq \frac{4m\,t^{\beta}}{\beta(m-1)}$$

which grows as t grows.

In general, nonlinear parabolic equations with degeneracy do not have classical solutions, and thus it is neccessary to generalize the notion of solutions. We can see from (2) that the solution to (1) may contain an interface where the gradient is discontinuous. A precise mathematical treatment involves the notion of $weak\ solutions$. Moreover, the behavior of weak solutions causes many difficulties for a good numerical simulation. For example, the weak solution may lose its classical derivative at some (interface) points, and the sharp interface of the support may propagate with finite speed if the initial data have compact support.

Regularity of solutions

Let $\Omega \subset \mathbb{R}^d$ be a bounded Lipschitz domain and consider the PME

$$u_t - \Delta u^m, \ x \in \Omega, \ 0 < t \leq T.$$

For solvability we need additional conditions and introduce the intial value boundary problem:

- Initial state in the time dependent case: $u(x,t) = u_0(x), \ x \in \mathbb{R}^d$
- Dirichlet homogeneous boundary condition: $u(x,t) = 0$ on $S_T = \partial\Omega \times (0,T)$
- Constant energy: $\int_{\mathbb{R}^d} u(x,t)dx = 1$

Then for $u_0 \geq 0$ we have:

- Uniqueness provided that $u_0 \in L^1(\Omega)$
- Existence and uniqueness for **weak solution** provided that $u_0 \in L^{m+1}(\Omega) \subset L^1(\Omega)$

For d = 1:

Suppose that $u_0 \geq 0$ and is bounded, u_0^m is Lipschitz continuous and $u(x,t)$ is a generalized solution of (1) with $u(x,0) = u_0(x)$. Then

- for any $\tau \in (0,T)$, there exists u $\in C^{k,\frac{k}{2}}(\mathbb{R} \times [\tau, T])$ with $k = \min\{1, \frac{1}{m-1}\}$ and $u \in C^{k,\frac{k}{2}}(\mathbb{R} \times [0,T])$ provided u_0^{m-1} is Lipschitz continuous.
- *Barenblatt solution* shows that $u \in C^{k,\frac{k}{2}}(\mathbb{R} \times [\tau, T])$ is the best possible global result and the Holder exponent $k = \min\{1, \frac{1}{m-1}\}$ cannot be increased.

Physical Background

To numerically study any kind of problem we need a stable and consistent numerical scheme. Besides, a good scheme has to reproduce all of the important features of the original model, which arise from the physical background of the problem. First of all the scheme has to preserve the non negativity of solutions as we deal with densities and concentrations. Then, if we consider bounded domains with no-flux boundary conditions, the numerical approximation has to conserve the total mass.

The nonlinear diffusion equation can be written in a more generalized form as

$$u_t - \nabla \cdot (D(u)\nabla u) = f(x,t) \iff u_t + \nabla \cdot \vec{j} = f(x,t)$$

where

- $f(x,t)$: species sources,
- $u(x,t)$: time-dependent local amount of species,
- $D(u) = mu^{m-1}$: Density-dependent diffusivity, and
- $\vec{j} = -D(u)\nabla u = -mu^{m-1}\nabla u$: vector field of the diffusion species flux.

Where might the Porous Medium Equation arise?

There are a number of physical applications where this simple model appears in a natural way, mainly to describe processes involving fluid flow, heat transfer or diffusion.

The *porous medium equation* owes its name to its use in describing the flow of gas in a porous medium; filtration or diffusion can happen through the pores. The flow is governed by the following laws:

1. Conservation of mass:

$$\epsilon\rho_t + \nabla \cdot (\rho\vec{V}) = 0$$

where, $\epsilon \in [0,1]$ is the porosity of the medium, ρ density of gas and $\vec{V}$ velocity of fluid. If $\epsilon = 1$, we have the classical conservation of mass, that is the flow is every where.

2. Darcy's law:

$$\vec{V} = -\frac{k}{\mu}\nabla P$$

which means that the gradient of the pressure causes the gas to flow. Here P is the pressure, k is the permeability of the medium, μ is the viscosity of the fluid and the minus sign means that the flow goes from low to high pressure.

3. Equation of state, which for perfect gases asserts that:

$$\rho = \rho_0 P^\gamma$$

where γ, is the so-called $polytropic\ exponent$ with $0 < \gamma \leq 1$. and ρ_0 is the reference pressure.

Eliminating $\vec{V}$ and P gives:

$$\epsilon \rho_t = -\frac{k}{\mu} \frac{1}{\rho_0^{\frac{1}{\gamma}}} \frac{1}{1+\gamma} \Delta \rho^{1+\frac{1}{\gamma}} \iff \rho_t = C \Delta \rho^m$$

where $m = 1 + \frac{1}{\gamma} \geq 2$ and

$$C = \frac{k}{\epsilon \mu \rho_0^{\frac{1}{\gamma}} (1+\gamma)}$$

Setting $u = \rho$ and rescaling t by new time $t' = Ct$ we get

$$u_t = \Delta u^m, \quad m > 1.$$

For our problem, we choose m=2.

Finite Volume Space Discretization

Consider a polygonal domain $\Omega \subset \mathbb{R}^d$ of BVP for PDEs. Essentially, we have an infinite number of unknowns (an unknown value in each point of domain), which is not possible to handle on the computer, because of finite computational capabilities and memory. Therefore, we need a finite-dimensional approximation for this PDE. We will use the **Voronoi FVM** for this purpose.

The usual way of defining such an approximation is based on subdividing our computational domain into a finite number of elementary closed subsets (control volumes), which are called "meshes" or "grids". In our case, these elementary shapes are simplexes (triangles or tetrahedra). Therefore, we have

$$\bar{\Omega} = \bigcup_{k \in \mathcal{N}} \bar{\omega_k}$$

such that:

- ω_k are open covex domains i.e. $\omega_k \cap \omega_l = \emptyset \ (k \neq l)$
- $\sigma_{kl} = \bar{\omega_k} \cap \bar{\omega_l}$ are either empty, points or st-lines, if $|\sigma_{kl}| > 0$ then ω_k, ω_l are neighbours.
- $n_{kl} \perp \sigma_{kl}$: normal of $\partial\omega_k$ at σ_{kl}

The idea behind the construction of the FVM is to exploit the divergence form of the equation by integrating it over control volumes, and to use Gauss' theorem to convert the volume integrals into surface integrals across the boundaries, which are then discretized. So instead of integrating the fluxes inside the cells, we integrate these fluxes across the boundary of the cells. This simplifies the PDE substantially.

Moreover, our equation depends on time, so we will use the Rothe method, that consists of discretizing in time first and then in space.

1D Case - Interval Domain

Let $\Omega = (a, b)$ be subdivided into $n - 1$ intervals by:

- $x_1 = a < x_2 < \cdots < x_{n-1} < x_n = b$.

$$\omega_k = \begin{cases} (x_1, \frac{(x_1+x_2)}{2}) & k = a \\ (\frac{(x_{k-1}+x_k)}{2}, \frac{(x_k+x_{k+1})}{2}) & a < k < b \\ (\frac{(x_{n-1}+x_n)}{2}, x_n) & k = n \end{cases}$$

2D Case - Polygonal domain

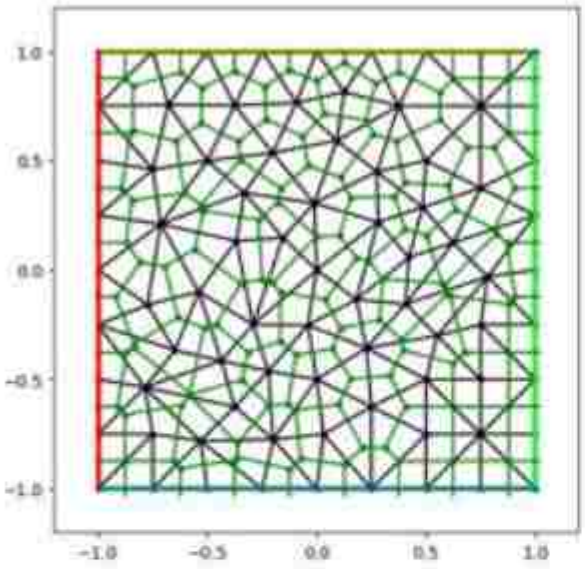
Here, we can use the connection between Voronoi and Delaunay. We essentially need a boundary conforming Delaunay triangulation, allowing us to construct Voronoi cells from it.

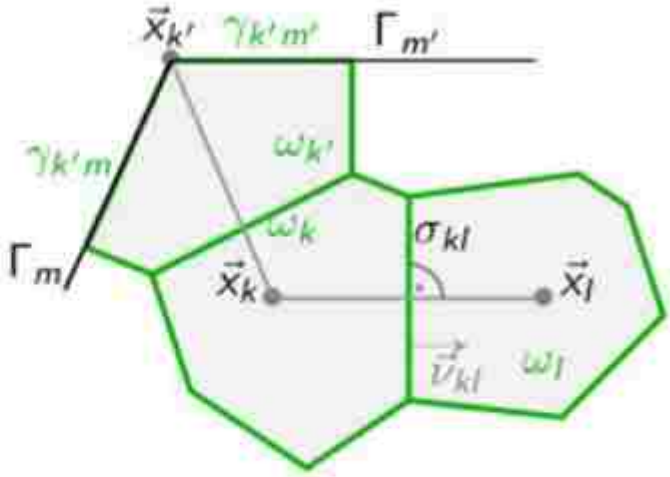
Let $S \subset \mathbb{R}^d$ be given finite set of points. The **Voronoi diagram** of S is the collection of the regions of the points of S and the Voronoi diagram subdivides the whole space into nearest neighbour regions.

In the Delaunay triangulation, there is a construct restriction of Voronoi cells ω_k with vertices $\vec{x_k} \in \omega_k$, where we require

- Delaunay property for each edge, that means for any two triangles sharing common edge, the sum of opposite angles should be less than or equal to π;
- Orthogonality between the interface of two control volumes and the discretization edge which makes the method consistent; and
- For triangles at the boundary that their circumcenters should be located within the domain.

The triangulation circumcenters are connected by lines, which define the Voronoi cells or control volumes. Each control volume ω_k is assigned a collocation point $\vec{x_k}$.



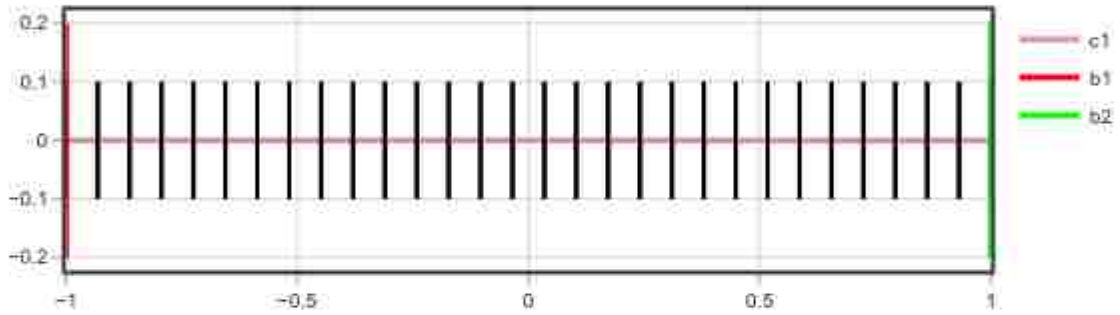


Mesh Generation

This is done by the package **ExtendableGrids.jl**.

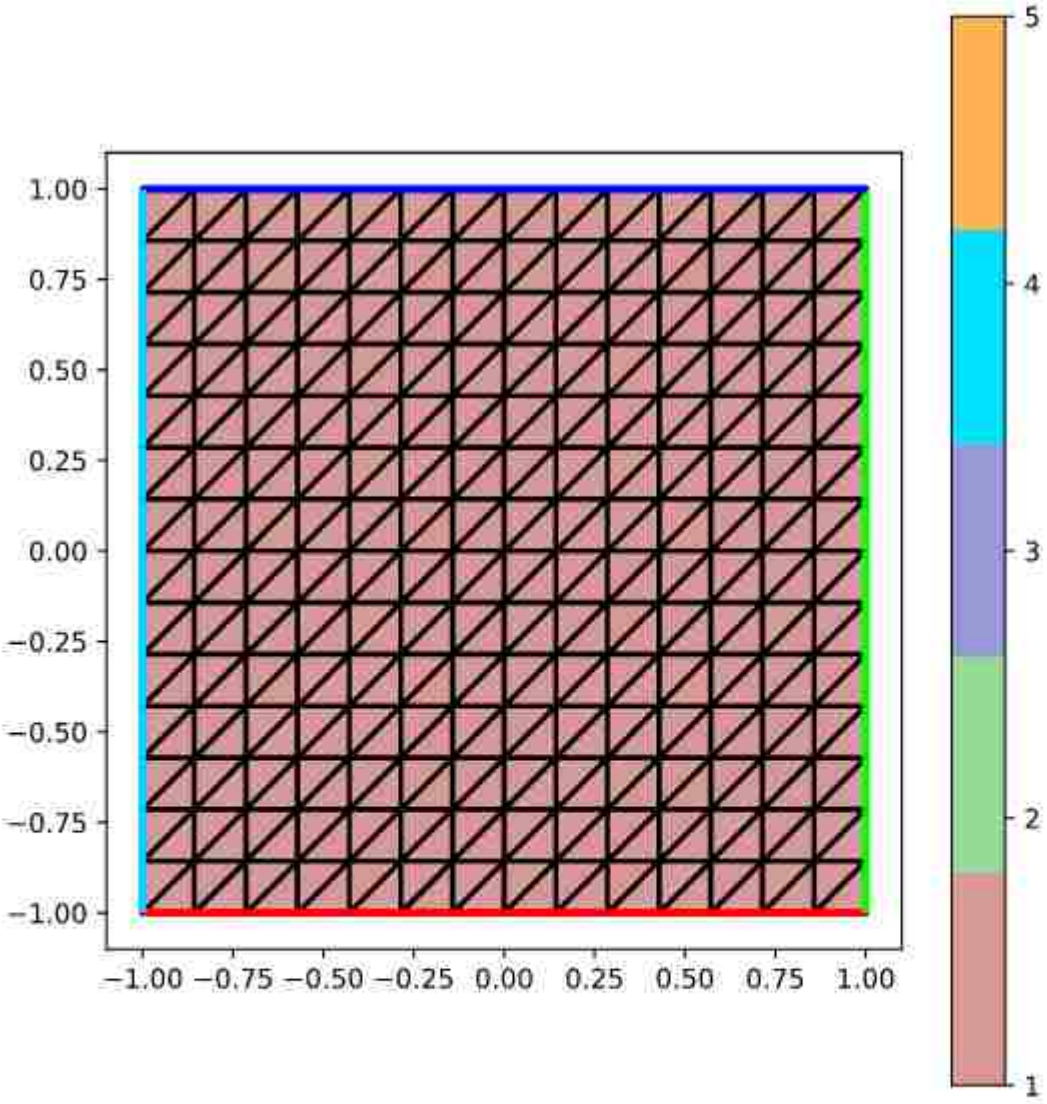
1D Discretization Grid

1D grids are created from vectors of monotonically increasing coordinates using the **simplexgrid** method. Grid in domain $\Omega = (-1, 1)$, consisting of N11=**30** points.



2D Discretization Grid

For 2D case, we just need to replace the 1D grid with a 2D tensor product grids.
The grid is created in the domain $\Omega = (-1, 1) \times (-1, 1)$, consisting of N13=**15** points in each coordinate direction, and is a Delaunay-comforming triangulation.



Second Order PDE With Robin BC.

Regard second order PDE with Robin boundary conditions as a system of two first order equations in a Lipschitz domain Ω.

- Continuity equation in Ω

$$u_t + \nabla \cdot \vec{j} = 0$$

- Flux law in Ω

$$\vec{j} = -D(u)\nabla u$$

- Boundary condition on Γ

$$-\vec{j} \cdot \vec{n} + \alpha u = \beta$$

where, $\alpha, \beta \geq 0$.

Discretization Of Continuity Equation

After subdividing the computational domain into a finite number of REV's, we can now start discretizing our system. We begin by discretizing the continuity equation: Integrating the spatial derivative in the continuity equation over the control volumes ω_k, and using **Gauss' Divergence Theorem** to convert the volume integral over the divergence into a surface integral across the boundaries, we get:

$$\begin{aligned} 0 &= u_t + \int_{\omega_k} \nabla \cdot \overrightarrow{j}\, d\omega \\ &= u_t + \int_{\partial\omega_k} \vec{j} \cdot \vec{n}_\omega \, ds \\ &= u_t + \sum_{l \in \mathcal{N}_k} \int_{\sigma_{kl}} \overrightarrow{j} \cdot \vec{n}_{kl} \, ds + \sum_{m \in \mathcal{G}_k} \int_{\gamma_{km}} \overrightarrow{j} \cdot \overrightarrow{n_m} \, ds \end{aligned}$$

$\mathcal{G}_k$ denotest the set of non-empty boundary parts of $\partial\omega_k$, and $\mathcal{N}_k$ is the set of neighbours of ω_k.

Approximation of flux between control volumes

The finite difference approximation of the normal derivative is

$$\nabla u \cdot \vec{n}_{kl} \approx \frac{u_l - u_k}{h_{kl}},$$

and using this we can approximate the flux between neighboring control volumes:

$$\int_{\sigma_{kl}} \vec{j} \cdot \vec{n}_{kl}\, ds = -\int_{\sigma_{kl}} D(u)\nabla u \cdot \vec{n}_{kl}\, ds \approx \frac{|\sigma_{kl}|}{h_{kl}} D(u_k - u_l) =: \frac{|\sigma_{kl}|}{h_{kl}} g(u_k, u_l)$$

Here, $g(\cdot,\cdot)$ is called the *flux function*, which will be used later in the implementation of our problem. Therefore, we get

$$\sum_{l \in \mathcal{N}_k} \frac{\sigma_{kl}}{h_{kl}} g(u_k, u_l) + \sum_{m \in \mathcal{G}_k} \int_{\gamma_{km}} \vec{j} \cdot \overrightarrow{n_m}\, ds \approx 0,$$

Approximation of boundary fluxes

In order to approximate the RHS integral, we assume that $\alpha|_{\Gamma_m} = \alpha_m$ and $\beta|_{\Gamma_m} = \beta_m$. Then the approximation of $\vec{j} \cdot \vec{n}_m$ at the boundary of ω_k is

$$\vec{j} \cdot \vec{n}_m \approx \alpha_m u_k - \beta_m$$

and the approximation of the flux from ω_k through Γ_m is

$$\int_{\gamma_{km}} \vec{j} \cdot \overrightarrow{n_m}\, ds \approx |\gamma_{km}|(\alpha_m u_k - \beta_m).$$

Discretized system of equations

The discrete system of equations then writes for $k \in \mathcal{N}$:

$$u_t + \sum_{l \in \mathcal{N}_k} \frac{|\sigma_{kl}|}{h_{kl}} D(u_k - u_l) + \sum_{m \in \mathcal{G}_k} |\gamma_{km}|(\alpha_m u_k - \beta_m) = 0,$$

which is equivalent to

$$u_t + u_k \left(D \sum_{l \in \mathcal{N}_k} \frac{|\sigma_{kl}|}{h_{kl}} + \alpha_m \sum_{m \in \mathcal{G}_k} |\gamma_{km}| \right) - D \sum_{l \in \mathcal{N}_k} \frac{|\sigma_{kl}|}{h_{kl}} u_l - \sum_{m \in \mathcal{G}_k} |\gamma_{km}| \beta_m = 0$$

Transient Robin BVP

Choosing final time $T > 0$, we proceed with specifying an initial state of the system, since this is an initial boundary value problem.

$$\begin{aligned} u_t - \nabla \cdot (D\nabla u) &= 0 \quad \text{in } \Omega \times [0,T] \\ u(x,0) &= u_0(x) \quad \text{in } \Omega \\ D\nabla u \cdot \vec{n} + \alpha u &= \beta \quad \text{on } \Gamma \times [0,T] \end{aligned}$$

There are two methods which solve the space-time discretization problem:

1. The **Method of lines**, where we have to first discretize in space, get a huge ODE system, then apply time discretization; and
2. The **Rothe Method**, where we first discretize in time, then in space.

Method of lines vs Rothe Method

Traditionally, for time-dependent problems, we would utilize the (vertical) $method\ of\ lines$, since it results in simple data structures and matrix assembly. However, method of lines has a major disadvantage: the spatial mesh is **fixed** (since we first discretize in space). Since the mesh is fixed initially, it is therefore difficult to represent or compute time-varying features such as certain target functionals (e.g., drag or lift).

On the other hand, the $Rothe\ method$, gives a PDE for each time step that we may choose to discretize independently of the mesh used for the previous time step. Thus, Rothe method allows for **dynamic** spatial mesh adaptation with the price that data structures and matrix assembly are more costly. This is why we will choose the **Rothe Method** over the method of lines.

Rothe Method

We choose time discretization points $0 = t^0 < t^1 < \ldots < t^N = T$. Let $\tau^n = t^n - t^{n-1} = \frac{T}{N}$ where the superscript n indicates the number of a time step and τ^n the difference between two neighboring time discretization points or the length of the present time step. Here we use finite differences and more specifically we introduce the so-called One-Step $\theta-$ scheme. Thus, we approximate the time derivative by a finite difference in time and evaluate the main part of the equation for the value interpolated between the old and new timesteps. We choose θ and substitute u^θ in all the terms concerning boundary conditions.

For $n = 1, 2, 3. \ldots, N$, we have to solve:

$$\frac{u^n - u^{n-1}}{\tau^n} - \nabla \cdot D\nabla u^\theta = 0 \text{ in } \Omega \times [0, T]$$

$$D\nabla u^\theta \cdot \vec{n} + \alpha u^\theta = \beta \text{ on } \Omega \times [0, T]$$

where u^θ is the linear combination of the solution at the new and old time points.

$$u^\theta = \theta u^n + (1 - \theta) u^{n-1}$$

Several methods exist for solving time discretization, and one of them is the $\theta-$ scheme that has three cases:

- Backward Implicit Euler Method, for $\theta = 1$
- Crank Nicolson Method, for $\theta = \frac{1}{2}$
- Forward explicit Euler Method, for $\theta = 0$

From a numerical point of view, implicit $\theta-$ scheme possesses several properties that make it very attractive for practical computations, such as parallelization, adaptivity, and simple treatment of boundary conditions. The most important properties of this method are its strong stability and high-order accuracy. Indeed, to have good stability properties of time-stepping, $\theta-$scheme is important for temporal discretization of PDEs.

Time Discretization

Let us write the derivation of FVM. Given control volume ω_k, integrate equation over space-time control volume $\omega_k \times (t^{n-1} - t^n)$, divide by τ^n :

$$\begin{aligned}
0 &= \int_{\omega_k} \left(\frac{1}{\tau^n}(u^n - u^{n-1}) - \nabla \cdot D\vec{\nabla} u^\theta \right) d\omega \\
&= \frac{1}{\tau^n} \int_{\omega_k} (u^n - u^{n-1}) d\omega + \sum_{l \in N_k} \frac{|\sigma_{kl}|}{h_{kl}} D(u_k - u_l) \\
&+ \sum_{m \in \mathcal{G}_k} |\gamma_{km}|(\alpha_m u_k - \beta_m) \\
&\approx \frac{|\omega_k|}{\tau^n}(u_k^n - u_k^{n-1}) + \sum_{l \in N_k} \frac{|\sigma_{kl}|}{h_{kl}} D(u_k^\theta - u_l^\theta) \\
&+ \sum_{m \in \mathcal{G}_k} |\gamma_{km}|(\alpha_m u_k^\theta - \beta_m)
\end{aligned}$$

Then, we obtain

$$\frac{|\omega_k|}{\tau^n}(u_k^n - u_k^{n-1}) + \sum_{l \in N_k} \frac{|\sigma_{kl}|}{h_{kl}} D(u_k^\theta - u_l^\theta) + \sum_{m \in \mathcal{G}_k} |\gamma_{km}|(\alpha_m u_k^\theta - \beta_m) = 0$$

and

$$\underbrace{\frac{|\omega_k|}{\tau^n}(u_k^n - u_k^{n-1})}_{\to M} + \underbrace{\sum_{l \in N_k} \frac{|\sigma_{kl}|}{h_{kl}} D(u_k^\theta - u_l^\theta) + \sum_{m \in \mathcal{G}_k} |\gamma_{km}| \alpha_m u_k^\theta}_{\to A} = \underbrace{\sum_{m \in \mathcal{G}_k} |\gamma_{km}| \beta_m}_{\to b}$$

The resulting matrix equation then reads

$$\frac{1}{\tau^n}\left(Mu^n - Mu^{n-1}\right) + Au^\theta = b$$

Nonlinear system of equations

We are left with a nonlinear system of equations that must be solved at each time step:

$$u^n + \tau^n M^{-1}\theta A u^n = u^{n-1} + \tau^n M^{-1}(\theta - 1)Au^{n-1} + b\tau^n M^{-1}$$

where $M = (m_{kl})$, $A = (a_{kl})$ and $b = (b_k)$ with coefficients

$$a_{kl} = \begin{cases} \sum_{l' \in \mathcal{N}_k} D\frac{|\sigma_{kl'}|}{h_{kl'}} + \sum_{m \in \mathcal{G}_k} |\gamma_{km}|\alpha_m & l = k \\ -D\frac{\sigma_{kl}}{h_{kl}}, & l \in \mathcal{N}_k \\ 0, & else \end{cases}$$

$$m_{kl} = \begin{cases} |\omega_k| & l = k \\ 0, & else \end{cases}$$

$$b_k = \sum_{m \in \mathcal{G}_k} |\gamma_{km}|\beta_m$$

Matrix Properties

To figure out how we solve the nonlinear system we look for some desirable matrix properties:

- $\theta A + M$ is $strictly\ diagonally\ dominant$

$$\sum_{k \neq l} |(\theta A + M)_{kl}| < |(\theta A + M)_{kk}|$$

- $\theta A + M$ has M$-property$

A and M have M $-property$ (off-diagonal entries are non-positive and all eigenvalues have positive parts) and essentially the matrix is **invertible**.

- $\theta A + M$ is symmetric and therefore it is positive definite.

Lemma

Assume that A has positive main diagonal entries, non-positive off-diagonal entries and row sum zero. Then, $\|(I + A)^{-1}\|_\infty \leq 1$.

Note: In our case: Robin BVP, the matrix does not satisfy row sum zero.

Transient Neumann BVP

Downgrade our problem a little bit. For $\alpha, \beta = 0$ we get the homogeneous neumann problem:

$$\begin{aligned} u_t - \nabla \cdot D\nabla u = 0 \quad &\text{in } \Omega \times [0,T] \\ u(x,0) = u_0(x) \quad &\text{in } \Omega \\ D\nabla u \cdot \vec{n} = 0 \quad &\text{on } \Gamma \times [0,T] \end{aligned}$$

Deriving FVM leads to

$$u^n + \tau^n M^{-1} \theta A u^n = u^{n-1} + \tau^n M^{-1}(\theta - 1) A u^{n-1}$$

where

- $\theta A + M$ is strictly diagonally dominant!
- $\sum_{l=1}^{n} a_{kl} = 0$: A has row sum zero.

Physical Properties

Stability Analysis

For stability, our question is: Under which conditions can we assume the estimation of new time value by the old time value, in the sense of L^∞ norm?

$$\|u^n\|_\infty < \|u^{n-1}\|$$

For the answer, we turn to the matrix equation again:

$$\begin{aligned} u^n + \tau^n M^{-1}\theta A u^n &= u^{n-1} + \tau^n M^{-1}(\theta - 1)Au^{n-1} \\ u^n &= (I + \tau^n M^{-1}\theta A)^{-1}(I + \tau^n M^{-1}(\theta - 1)A)u^{n-1} \end{aligned}$$

From the lemma we have

$$||(I + \tau^n M^{-1}\theta A)^{-1}||_\infty \leq 1,$$

which yields

$$||u^n||_\infty \leq ||B^n u^{n-1}||_\infty.$$

Thus, we have to estimate $||B||_\infty$ where the sufficient condition is that for some $C > 0$,

$$(1 - \theta)\tau^n \leq Ch^2$$

Stability of methods

- Implicit Euler: $\theta = 1 \Rightarrow$ unconditional stability
- Explicit Euler: $\theta = 0 \Rightarrow \tau \leq Ch^2$
- Crank-Nicolson: $\theta = \frac{1}{2} \Rightarrow \tau \leq 2Ch^2$

Implicit Euler method

Implicit Euler methods are **unconditionally stable**: In particular, without any additional condition, we can guarantee that we have no physical oscillation of our solution. They also require more computational efforts, especially for nonlinear equations. In light of this, time accuracy is less important and the number of timesteps is independent of the size of the space discretization.

Obstacles for implementing the explicit Euler method

Explicit Euler methods involve very small timesteps. They are cheaper computationally, but are **conditionally stable**, causing your stepsize in time to be dependent on the numerical method we choose, leading generally to more time steps taken to get to a desired time. Also, they are easy to implement with only simple calculations performed at each timestep. If we have nicely structured system we can readily apply this, for instance on very fast systems we have a very small timestep and high accuracy in time.

Discrete Maximum Principle

If we want to prevent oscillation of the solution, we can look at another interesting physical property: the **Discrete Maximum Principle**.

Let us regard the Implicit Euler method ($\theta = 1$):

$$\frac{1}{\tau^n}\left(Mu^n - Mu^{n-1}\right) + Au^n = 0$$

$$\frac{1}{\tau^n}Mu^n + Au^n = \frac{1}{\tau^n}Mu^{n-1}$$

$$\frac{1}{\tau^n}m_{kk}u_k^n + a_{kk}u_k^n = \frac{1}{\tau^n}m_{kk}u_k^{n-1} + \sum_{k\neq l}(-a_{kl})u_l^n$$

$$u_k^n \leq \max(\{u_k^{n-1}\} \cup \{u_l^n\}_{l\in\mathcal{N}_k})$$

It is important, in this case, that we have a positive diagonal matrix M that makes this estimate possible. Indeed, the value of the solution at a certain discretization point and at the new timestep is estimated by the solution from the old timestep, and by the solution in the neigboring points.

Nonnegativity

Regarding the nonnegativity of the solution in our timestep:

$$u^n + \tau^n M^{-1} A u^n = u^{n-1}$$
$$u^n = (I + \tau^n M^{-1} A)^{-1} u^{n-1}$$

$(I + \tau^n M^{-1} A)$ is an M-Matrix and its inverse has positive entries, which means that if $u_0 > 0$, then $u^n > 0$ for all $n > 0$.

Numerical Solution Methods

We performed space and time discretization of our computational domain. We can now solve the system of our equation. We look at the full discretization which consists of combining the Voronoi finite volume method and backward Euler method:

$$u^n + \tau^n M^{-1} A u^n = u^{n-1}$$
$$u^n = (I + \tau^n M^{-1} A)^{-1} u^{n-1}$$

The cheap **explicit Euler** method only works for very small time steps. That is a compensation with respect to the fact that we don't have to solve a linear system. It is possible to get very accurate iterative solvers for steady-state problems if we use the backward Euler method, but they are difficult to implement and parallelize.

The **Implicit Euler** method is stable over a wide range of time steps. It is for this reason, as well as the reasons above, that we choose it over the explicit Euler method.

Simulation Results

We present in this section the numerical results we obtained for the 1D case. We will look at the 1D results on a 1D grid and then the 1D results on a 2D grid.

1D Case

We use **VoroinoiFVM.jl** to approximate the solution. We do so by creating the physics (flux and storage) and passing it to the `System` constructor. Additionally, by specifying the `bcondition` callback in the constructor, we can include all three kinds of homogeneous boundary conditions (Dirichlet, Neumann, Robin). The solver control uses **Newton method** to solve the nonlinear system of equations.

For the time interval, $t_0 = 0.001$ and $t_1 = 0.01$ were chosen so that the support of the Barenblatt solution, $b(x, t_1)$, lies in $(-1, 1)$. Additionally, we choose $u(x, t_0) = u_0(x) = b(x, t0)$.

Homogeneous Robin BC

```
tsol_robin,err_robin = diffusion1d(m, bcond=robin_bc!);
```

The error is calculated by comparing the numerical solution to the Barenblatt solution at $t_1 = 0.01$.

```
0.07107747166535447
```

```
err_robin
```

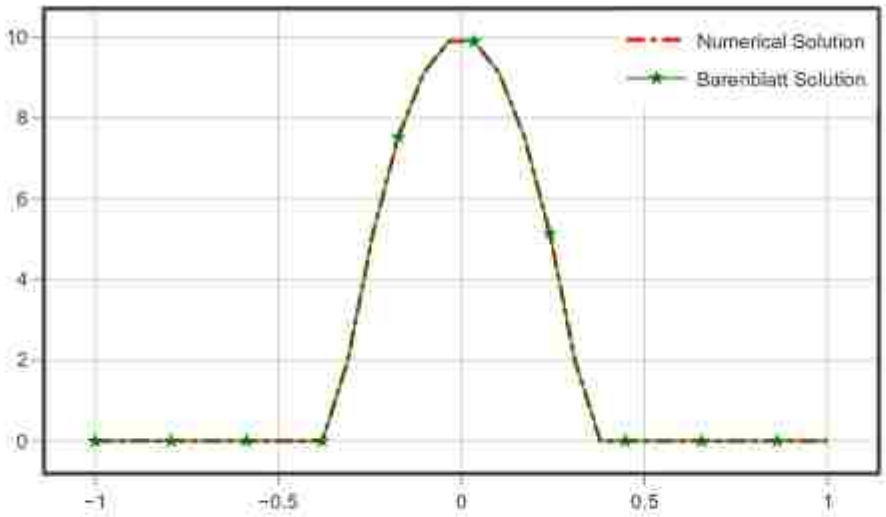
Solution plots

Binding the timesteps to the slider, we can see the numerical solution plotted against our domain $(-1, 1)$ at various points in time. To compare the error, the Barenblatt solution is also plotted on the same graph.

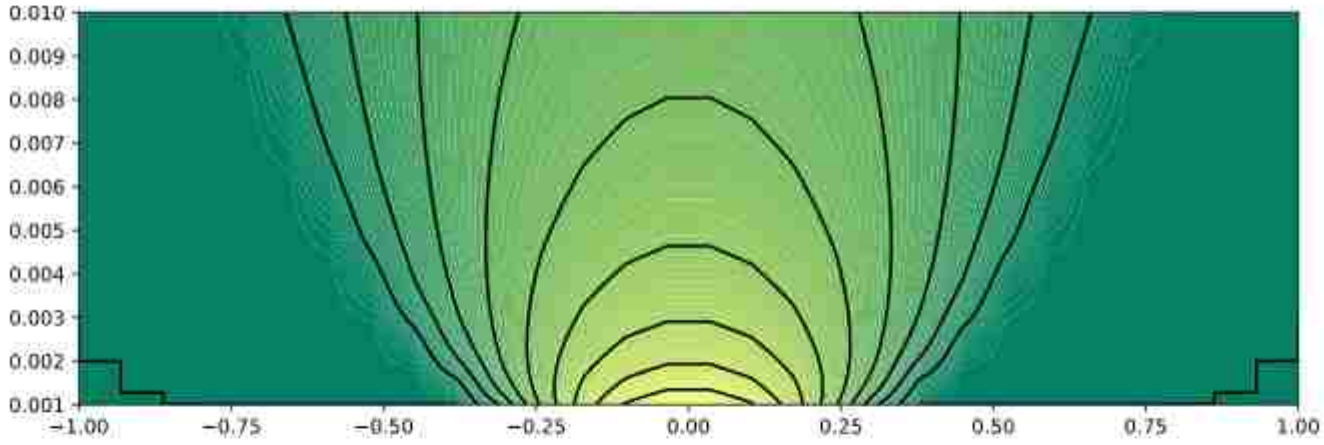
Timestep: 1

Time: 0.001

```
sol_robin_t =
▸ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.97384, 5.14467, 7.52279, 9.1082, 9.9(
```



The following plot shows the evolution of the numerical solution throughout time. We plot our grid on the x-axis and time on the y-axis, thus showing all of the information about the transient solution in one graph. The higher values are represented by lighter colours - and the expected behaviour of diffusion problems can be seen here as the peak value slowly degrades over time.



Homogeneous Neumann BC

By changing the `bcondition` callback, we set the boundary condition to homogeneous Neumann and repeat the above analysis.

```
tsol,err = diffusion1d(m);
```

```
0.07107747166535447
```

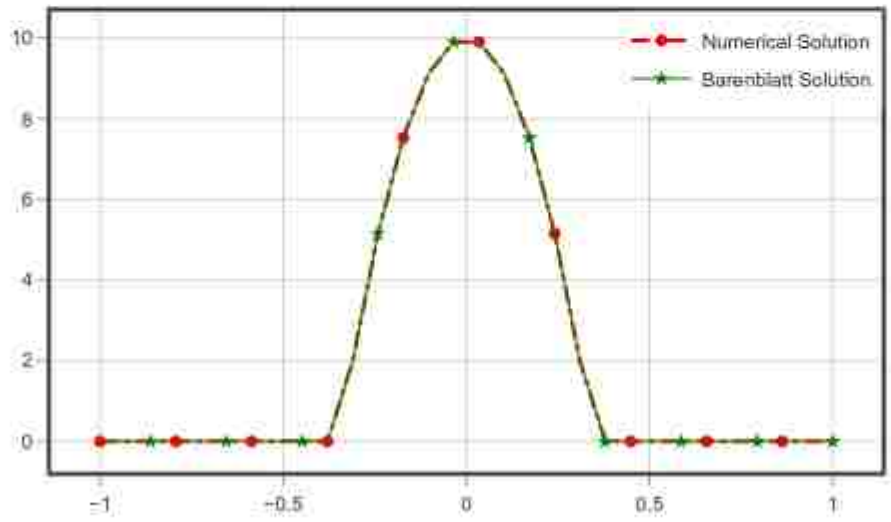
```
err
```

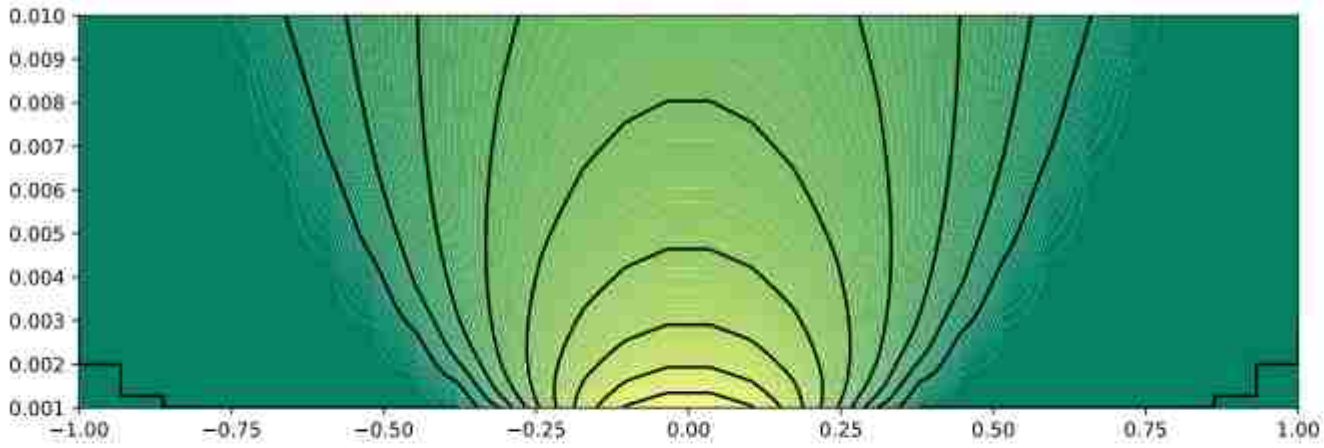
Solution plots

Timestep: 1

Time: 0.001

```
sol_τ =
▸ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.97384, 5.14467, 7.52279, 9.1082, 9.9(
```





Homogeneous Dirichlet BC

And finally, we observe the case of the homegeneous Dirichlet BC.

```
tsol_dirichlet,err_dirichlet=diffusion1d(m, bcond=dirichlet_bc!);
```

```
0.07107747166535447
```

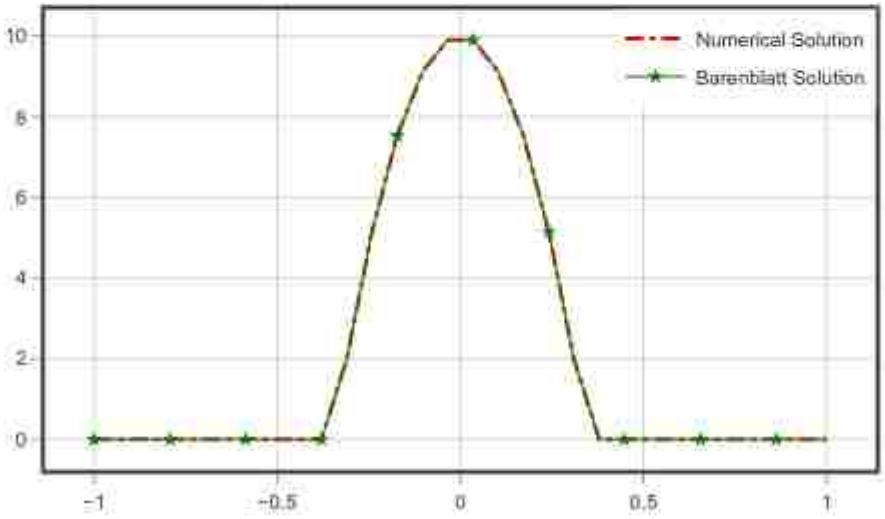
```
err_dirichlet
```

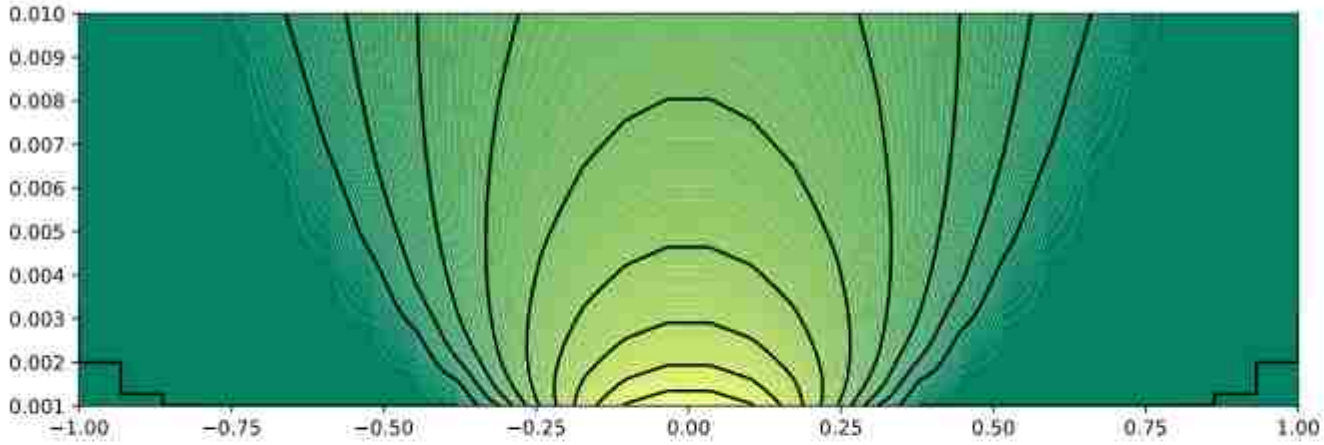
Solution plots

Timestep: 1

Time: 0.001

```
sol_dirichlet_τ =
▸[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.97384, 5.14467, 7.52279, 9.1082, 9.9(
```

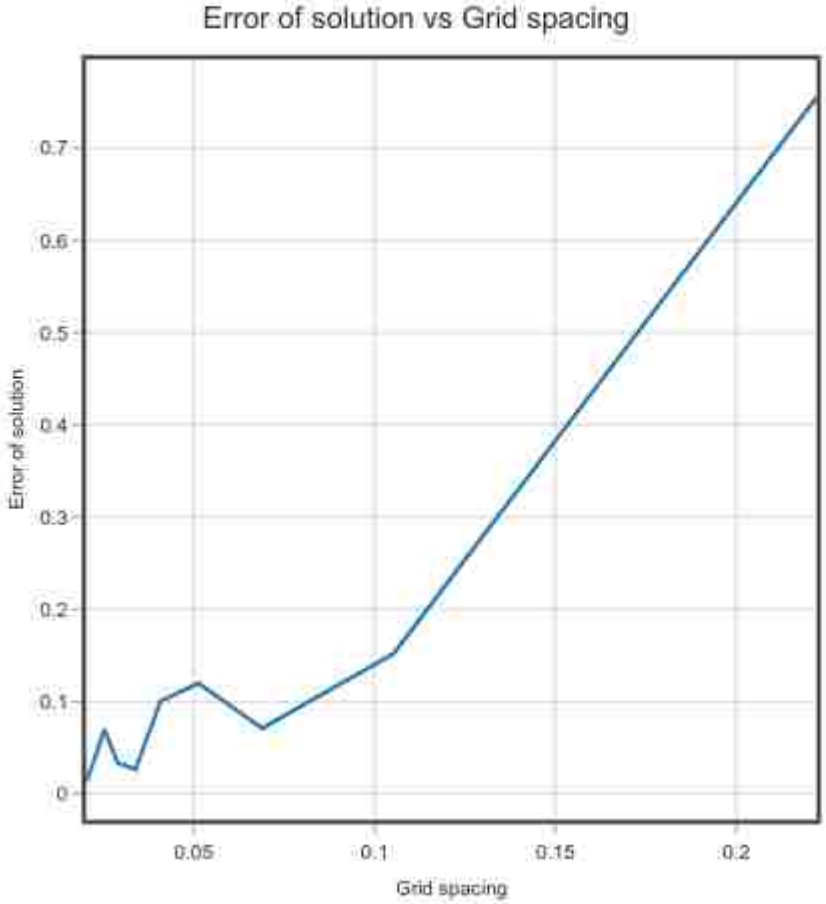




Error of solution vs grid spacing

We manually run our function `diffusion1d` for different values of N, the number of grid points. The error in each case is plotted against the grid spacing. We can see that increasing N, or equivalently, decreasing the grid spacing leads to lower error.

```
▶ ([0.222222, 0.105263, 0.0689655, 0.0512821, 0.0408163, 0.0338983, 0.0289855, 0.0253165, (
  spacinglist,errlist=err_sol_vs_grid_spacing(m)
```



Choosing the time values

We need to choose the time values t_0 and t_1 such that the support of $b(x, t_1)$ is contained in $(-1, 1)$.

i.e.,

$$-1 < -2t^{\frac{\beta}{2}}\sqrt{\frac{m}{\beta(m-1)}} \leq x \leq 2t^{\frac{\beta}{2}}\sqrt{\frac{m}{\beta(m-1)}} < 1$$

which grows as t grows.

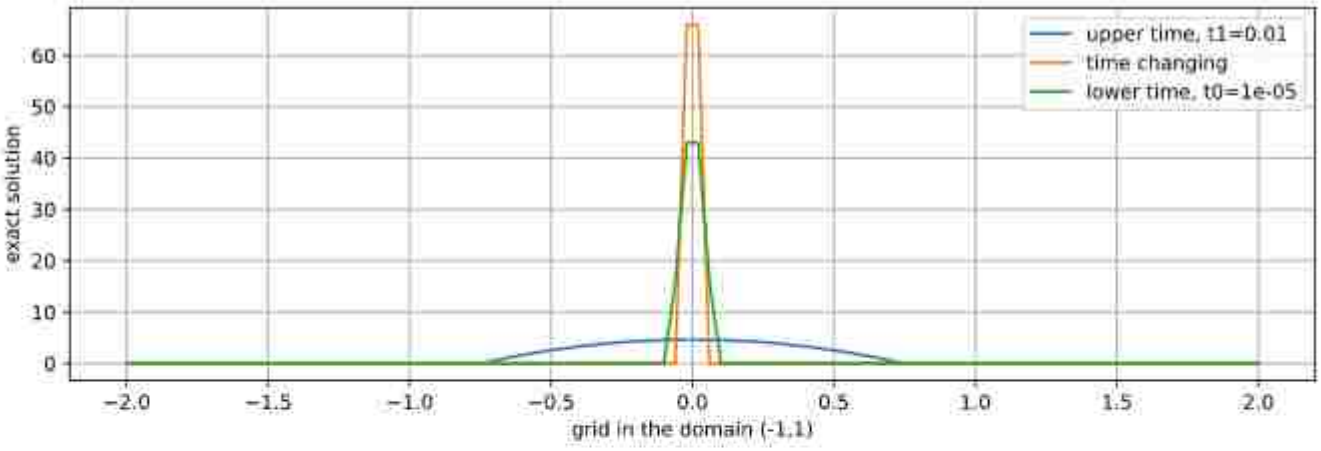
To start with, let us implement the Barenblatt solution as functions.

```
barenblatt (generic function with 1 method)
```

```
barenblatt2d (generic function with 1 method)
```

The support of the Barenblatt solution lies in the domain for $t0 > 0.00001$ and $t1 < 0.01$.

time= 1.0e-6



Note: We have used the same time values for the 2D case as well.

2D Case

We repeat the above analysis for the 2D case.

Homogeneous Robin BC

```
tsol2d_robin,err2d_robin=diffusion2d(m, bcond=robin_bc!);
```

As we did in the 1D case, we measure error by comparing our solution to the exact solution at $t1$.

```
11.275606605012797
```

```
err2d_robin
```

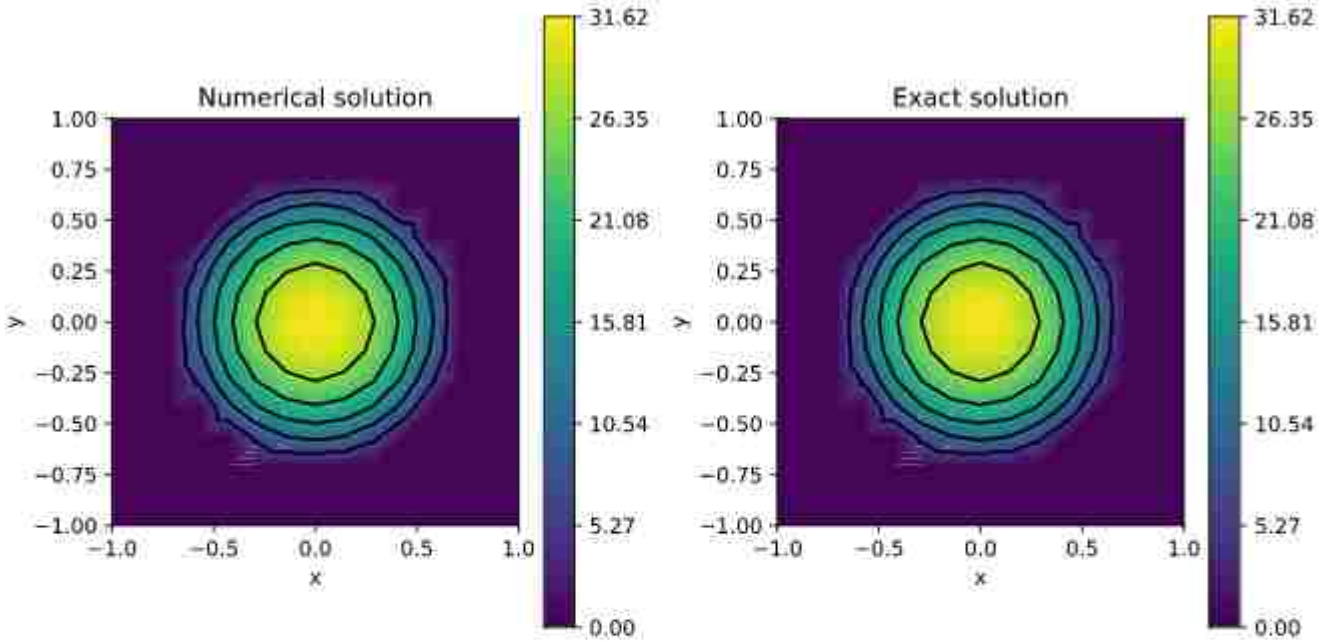
Solution plot

Timestep: 1

Time: 0.001

```
sol_t2d_robin =
▸ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

The following plot shows our solution (on the left) spread out over the domain, and as we can see from the legend, higher values are represented by lighter colours. Binding the timesteps to the slider (similar to the 1D case) will let us see the evolution of our solution over time, and compare it to the exact solution (on the right.)



Homogeneous Neumann BC

```
tsol2d,err2d=diffusion2d(m);
```

In the 1D case, we observed that the error stayed the same no matter which boundary condition we chose - however, we see in the 2D case that the error changes when the boundary condition changes.

```
11.941902635813467
```

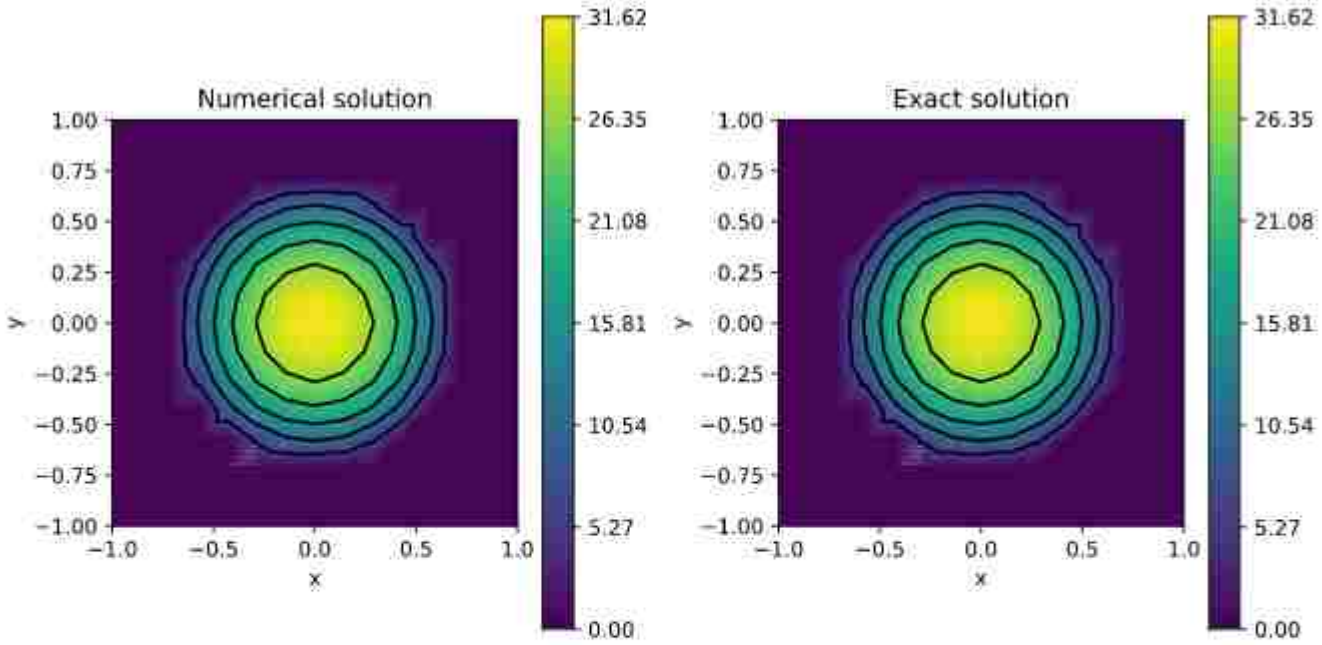
```
err2d
```

Solution plot

Timestep: 1

Time: 0.001

```
sol_t2d =
▸ [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```



Homogeneous Dirichlet BC

```
tsol2d_dirichlet,err2d_dirichlet=diffusion2d(m, bcond=dirichlet_bc!);
```

```
18.511930483412978
```

```
err2d_dirichlet
```

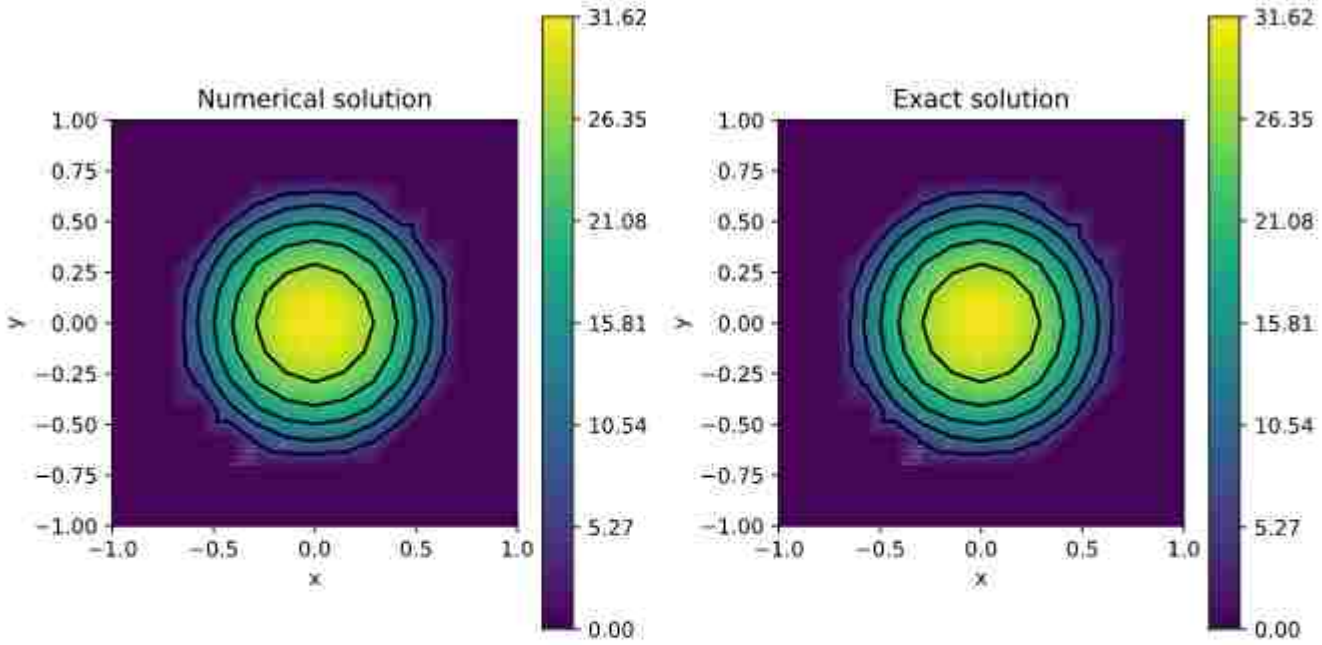
Solution plot

Timestep: 1

Time: 0.001

```
sol_t2d_dirichlet =
```

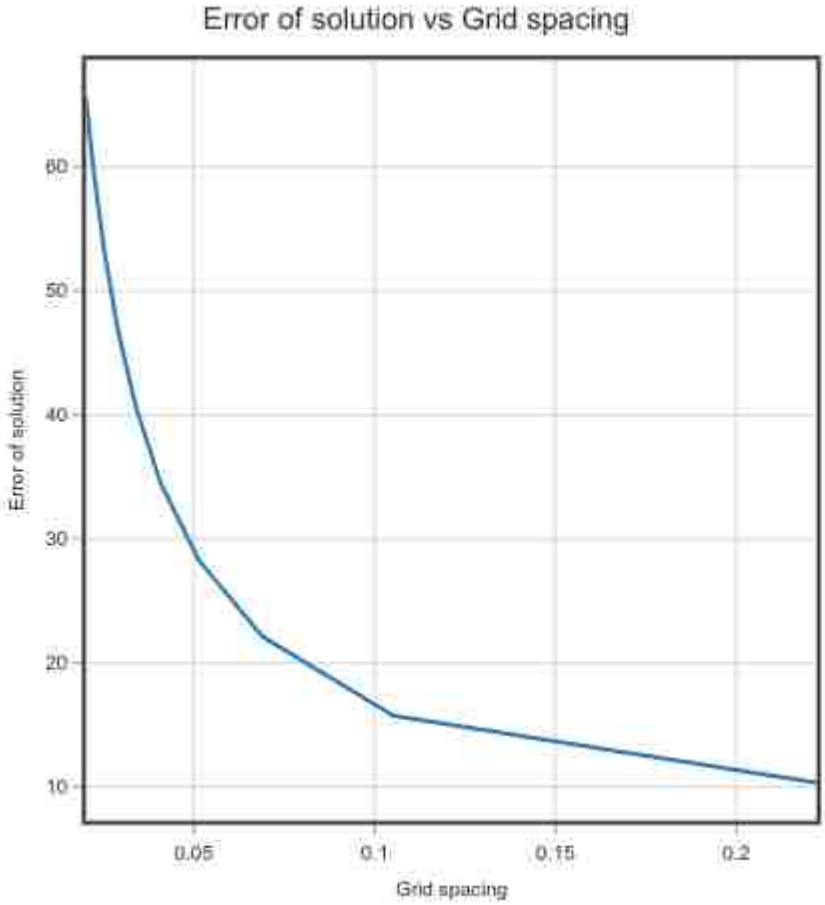

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```



Error of solution vs Grid spacing

This analysis is repeated for the 2D case, however here we observe the opposite effect: The error increases with an increase in the number of grid points N.

```
([0.222222, 0.105263, 0.0689655, 0.0512821, 0.0408163, 0.0338983, 0.0289855, 0.0253165, (
spacinglist2d,errlist2d=err_sol_vs_grid_spacing2d(m)
```



Optional

Improving Performance

Using **DifferentialEquations.jl** for transient problems can yield better performance. We see that this is indeed the case, as the **DifferentialEquations.jl** solver runs faster and also produces less error.

1D Case

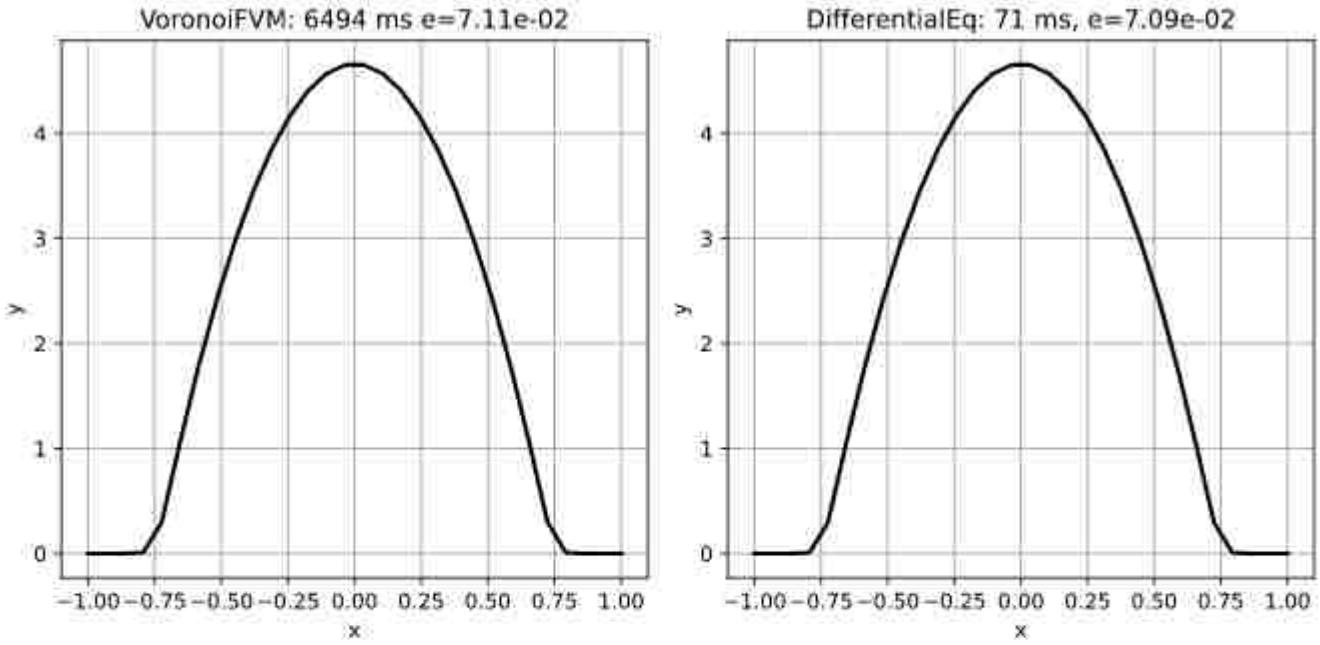
First we measure the time taken by the VoronoiFVM solver:

```
t = 6.493921821
```

Next we see how the **DifferentialEquations.jl** performs.

```
t2 = 0.071393583
```

We summarize the results in the following graph, where the numerical solution at $t1$, the runtimes, and the error are compared.



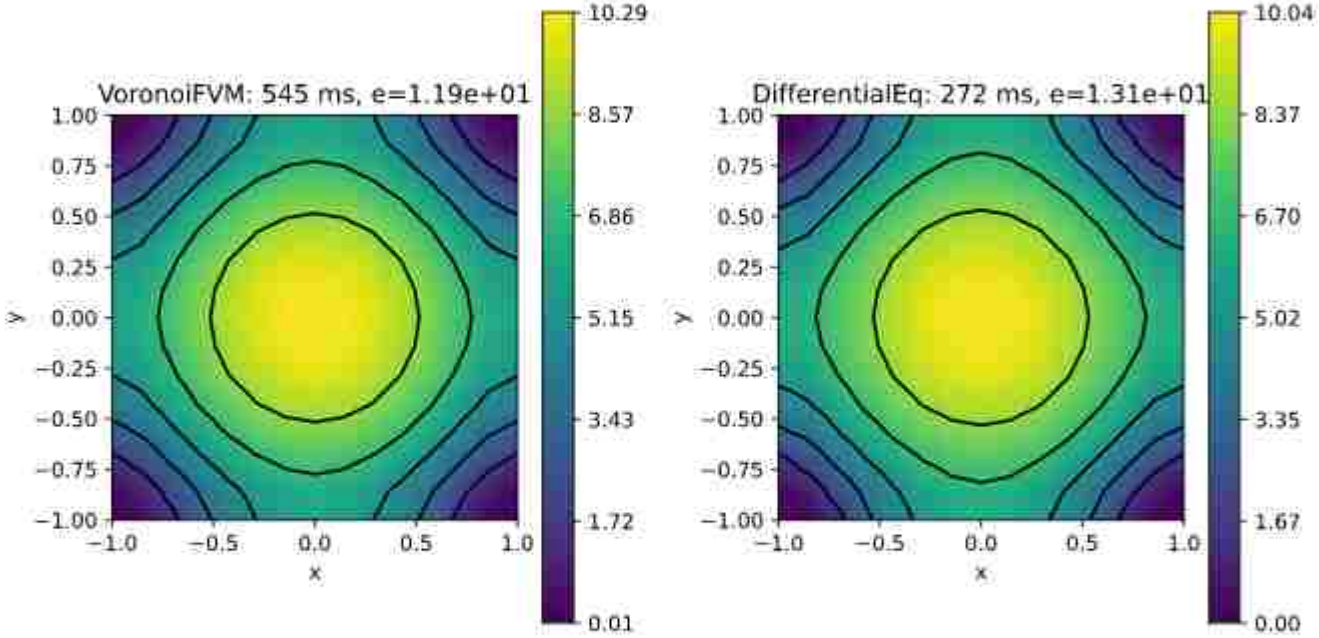
2D Case

For the 2D case we perform the same steps. Here $t3$ measures the time taken by the VoronoiFVM.jl solver, and $t4$ measures the time taken by DifferentialEquations.jl.

```
t3 = 0.544985484
```

```
t4 = 0.27248274
```

As above, we summarize the results in the form of the following graph.



Observation

We observe that, on the initial loading of the notebook, the DiffEq solver runs slower (equivalently, the time measuring block runs slower) than the VoronoiFVM solver. However, after running both cells for a second time, the correct time is shown.

Conclusion

So far, many different approaches have been taken to approximate the exact solution for the porous medium equation (PME). However, it still remains to find an appropriate scheme that can approximate the exact solution when the adiabatic exponent increases monotonically. In this report, we have presented numerical results for which we have used Explicit-Implicit Finite volume Method (EIFVM). Since the PME is a degenerate parabolic equation and analytically the existence and uniqueness occur weakly only in the Sobolev sense, it is very hard to approximate the exact solution numerically.
We have used **VoronoiFVM.jl** to approximate the solution, for both the 1D and 2D case, and compared it to the Barenblatt solution. We then plotted the solutions over time and provided a full spacetime plot. The effect of decreasing the grid spacing on the error was also measured, with an anomaly in the 2D case. Finally, we discussed performance, and how using **DifferentialEquations.jl** leads to faster solutions and lower errors in some cases.

References

Listed below are the references we used for this project (URLs included).

1. Chowdhury, Atiqur & Barmon, Ashish & Alam, Sharmin & Akter, Maria. (2017). Numerical Simulation that Provides Non-oscillatory Solutions for Porous Medium Equation and Error Approximation of Boussinesq's Equation. Universal Journal of Computational Mathematics.
2. Vazquez, Juan Luis. (2007). The Porous Medium Equation: Mathematical Theory.
3. Wu, Zhuoqun & Yin, Jingxue & Li, Huilai & Zhao, Junning. (2001). Nonlinear Diffusion Equations.
4. Knabner, Pter & Angermann, Lutz. (2003). Numerical Methods for Elliptic and Parabolic Partial Differential Equations., Springer-Verlag.

Julia functions

```
diffusion1d (generic function with 5 methods)

robin_bc! (generic function with 1 method)

dirichlet_bc! (generic function with 1 method)

neumann_bc! (generic function with 1 method)

plotsol (generic function with 1 method)

diffusion2d (generic function with 6 methods)

err_sol_vs_grid_spacing (generic function with 2 methods)

err_sol_vs_grid_spacing2d (generic function with 2 methods)

diffusion1d_perf (generic function with 5 methods)

diffusion2d_perf (generic function with 5 methods)
```