Analytical and Numerical Methods for Partial Differential Equations and Transforms

---

Laboratory 1
Convection and Diffusion

1 Convection problem

This task concerns the concentration $u : \Omega \to \mathbb{R}$ of wildfire smoke, satisfying the partial differential equation $v \cdot \nabla u = f$ in a domain Ω. For simplicity, we assume that the wind, $v : \Omega \to \mathbb{R}^2$, and smoke production, $f : \Omega \to \mathbb{R}$, are time-independent in the region $\Omega \subset \mathbb{R}^2$.

To explain it a bit better we have a concentration of wildfire smoke u that describes the amount of wildfire smoke that is represented by the function u. This function u depends on some region Ω that we can think of as some specific geographical area.

The partial differential equation $v \cdot \nabla u = f$ tells us how the concentration of wildfire smoke will change in some geographic region Ω. In order for us to describe this we need 3 components:

- Information about the direction and speed of the wind represented by the vector field v defined over the geographical region Ω.
- Information about the gradient of the concentration u that tells us how the concentration varies over some geographical region.
- Finally we need to know where smoke is being produced ie. where the forest is burning.

NOTE: For simplicity we also assumed that the wind and the smoke production rate do not change. In reality this is not the case since wind patterns change over time and the rate of smoke being generated at a sort point in space has to vary since fuel burns out and different materials burn in at different rates. So by use assuming time-independence it makes the problem more manageable for theoretical analysis.

1.1 Analytical Solution Using the Method of Characteristics.

First, assume that

$$\begin{aligned} u(x_1, 1) &= g(x_1), \quad \Omega := \left\{(x_1, x_2) \in \mathbb{R}^2 : x_1 \in \mathbb{R}, x_2 > 1\right\}, \\ v(x_1, x_2) &= (x_1, x_2), \\ f(x_1, x_2) &= 2u(x_1, x_2), \end{aligned}$$

where $g : \mathbb{R} \to \mathbb{R}$ is a given function.

We can use the method of characteristics to analytically solve the convection problem

$$v(x) \cdot \nabla u(x) = 2u(x).$$

1.1.1 Setup:

We want to solve the equation:

$$v(x) \cdot \nabla u(x) = 2u(x)$$

The **boundary conditions** are given as:

$$u(x_1, 1) = g(x_1)$$

where $g(x)$ is a known function.

The **wind velocity** is given by:

$$v(x_1, x_2) = (x_1, x_2)$$

Our **region of interest** is:

$$\Omega := \left\{(x_1, x_2) \in \mathbb{R}^2 : x_1 \in \mathbb{R}, x_2 > 1\right\}$$

1.1.2 Method of Characteristics:

The change of u along the curve $\gamma(s)$ equals the product of the velocity of the curve $\gamma'(s)$ and the gradient of u at $\gamma(s)$.

This can be expressed as:

$$\begin{aligned} \frac{d}{ds}u(\gamma(s)) &= \nabla u(\gamma(s)) \cdot \gamma'(s) \\ \gamma'(s) &= v(\gamma(s)) \\ \Rightarrow \frac{d}{ds}u(\gamma(s)) &= v(\gamma(s)) \cdot \nabla u(\gamma(s)) \\ \frac{d}{ds}u(\gamma(s)) &= \gamma'(s) \cdot \nabla u(\gamma(s)) = v(\gamma(s)) \cdot \nabla u(\gamma(s)) \end{aligned}$$

Here, $\gamma(s)$ represents the characteristic curve parameterized by s, and $\gamma'(s)$ is its derivative with respect to s. This implies $\gamma'(s) = v(\gamma(s)) = \gamma(s)$, leading to a system of differential equations.

Considering the system of ODEs:

$$\begin{aligned} \gamma_1'(s) &= \gamma_1(s), \\ \gamma_2'(s) &= \gamma_2(s) \end{aligned}$$

To solve each ODE:

$$\begin{aligned} \gamma_1(s) &= \gamma_1(0)e^s, \\ \gamma_2(s) &= \gamma_2(0)e^s \end{aligned}$$

Combining solutions into vector form:

$$\gamma(s) = (\gamma_1(0)e^s, \gamma_2(0)e^s)^T$$

Solving the Differential Equation:

1. Start with the differential equation:
$$\frac{d}{ds}u(\gamma(s)) = 2u(\gamma(s))$$
2. Find the general solution for $y' = ky$ which is $y = Ce^{ks}$:
$$u(\gamma(s)) = Ce^{2s}$$
3. Use the initial condition $u(\gamma(0))$ to find C:
$$u(\gamma(0)) = Ce^{2\cdot 0} \Rightarrow C = u(\gamma(0))$$
4. The final solution is:
$$u(\gamma(s)) = u(\gamma(0))e^{2s}$$

Finding u:

1. Initial function setup:
$$\gamma(0) = (x_1, x_2)$$
2. Evaluating u at different points along the characteristic curve:
$$u\left(x_1 \mathrm{e}^s, x_2 \mathrm{e}^s\right) \mathrm{e}^{-2s} = u\left(x_1, x_2\right) \quad \forall s \in \mathbb{R}$$
3. Applying the substitution $s = -\log(x_2)$:
$$\begin{aligned} u\left(\frac{x_1}{x_2}, 1\right) x_2^2 &= u\left(x_1, x_2\right) \\ u\left(\frac{x_1}{x_2}, 1\right) &= g\left(\frac{x_1}{x_2}\right) \\ u(x_1, x_2) &= g\left(\frac{x_1}{x_2}\right) x_2^2 \end{aligned}$$

Here we have assumed that the function g is differentiable.

1.2 Least Squares Method for Numerical Approximation

1.2.1 Setup

In applications, data are often given at specific data points, for example, when the wind is given only at observation points. In such cases:

- First step is to determine an approximation to the function from the given data.
- Next step is to solve the problem using the constructed function

Therefore, we now assume that the wind and smoke are given at M discrete points $\mathbf{x}_p = (x_p, y_p) \in \mathbb{R}^2$, meaning $v(\mathbf{x}_p) \in \mathbb{R}^2$ is the wind velocity at point $\mathbf{x}_p$, and $f(\mathbf{x}_p) \in \mathbb{R}$ is the smoke production at point $\mathbf{x}_p$, for $p = 1, \ldots, M$.

Select the points $\mathbf{x}_p$ randomly in $\Omega := [0,1] \times [0,1]$ with a uniform distribution, for example, set $M = 1000$.

Use a Fourier basis

$$\left\{ e^{\mathrm{i}\kappa(n_1 x + n_2 y)} : (n_1, n_2) \in \mathbb{Z}^2, |n_1| \leq N \quad \text{and} \ |n_2| \leq N \right\}$$

to interpolate data to Ω using the least squares method

$$\min_{\hat{v}_1 \in \mathbb{C}^{(2N+1)^2}} \sum_{p=1}^{M} | \, v_1(\mathbf{x}_p) - \underbrace{\sum_{|n_2| \leq N} \sum_{|n_1| \leq N} \hat{v}_1(n_1, n_2) \, e^{\mathrm{i}\kappa(n_1 x_p + n_2 y_p)}}_{=: \bar{v}_1(\mathbf{x}_p)} \Big|^2,$$

$$\min_{\hat{v}_2 \in \mathbb{C}^{(2N+1)^2}} \sum_{p=1}^{M} | \, v_2(\mathbf{x}_p) - \underbrace{\sum_{|n_2| \leq N} \sum_{|n_1| \leq N} \hat{v}_2(n_1, n_2) \, e^{\mathrm{i}\kappa(n_1 x_p + n_2 y_p)}}_{=: \bar{v}_2(\mathbf{x}_p)} \Big|^2,$$

$$\min_{\hat{f} \in \mathbb{C}^{(2N+1)^2}} \sum_{p=1}^{M} | \, f(\mathbf{x}_p) - \underbrace{\sum_{|n_2| \leq N} \sum_{|n_1| \leq N} \hat{f}(n_1, n_2) \, e^{\mathrm{i}\kappa(n_1 x_p + n_2 y_p)}}_{=: \tilde{f}(\mathbf{x}_p)} \Big|^2,$$

where $\kappa = 2\pi$ and, for example, $N = 6$.

We assume that the wind velocity v and smoke production f at the measurement points are generated as follows:

$$v(x, y) = (y, 1 - x),$$
$$f(x, y) = \begin{cases} 1, & \text{for } |(x, y) - (0.5, 0.5)| < 0.1 \\ 0, & \text{for } |(x, y) - (0.5, 0.5)| \geq 0.1 \end{cases}$$

We will use the least square solutions $(\bar{v}_1, \bar{v}_2)$ and $\bar{f}$ to approximate a convection problem with characteristics. Since v and f are real-valued, and we are looking for real-valued characteristics, we will, for numerical reasons, use the real part of the functions $\bar{v}$ and $\bar{f}$.

Let $\tilde{v}_i(x)$ be the real part of $\bar{v}_i(\mathrm{x})$, and $\tilde{f}(\mathrm{x})$ be the real part of $\bar{f}$, and let $\tilde{v}(\mathrm{x}) = (\tilde{v}_1(\mathrm{x}), \tilde{v}_2(\mathrm{x}))$.

Now determine an approximation to the concentration $u : \Omega \to \mathbb{R}$ that solves

$$\begin{aligned} \tilde{v}(x) \cdot \nabla u(x) &= \tilde{f}(x), \quad x \in \Omega, \\ u(x_1, 0) &= 0, \quad x_1 \in [0,1], \\ u(0, x_2) &= 0, \quad x_2 \in [0,1], \end{aligned}$$

similarly to task 1 but numerically.

´´´For each point $\left(\frac{k_1}{K}, \frac{k_2}{K}\right) \in \Omega$, where $k_i \in \{1, 2, \ldots, K\}$, choose a characteristic path as follows:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}s}x(s) &= \tilde{v}(x(s)), \quad s < 0,\\ x(0) &= \left(\frac{k_1}{K}, \frac{k_2}{K}\right),\\ \frac{\mathrm{d}}{\mathrm{d}s}u(s) &= \tilde{f}(x(s)),\end{aligned}$$

1.2.2 Solution

To solve the approximation of the function, we solved the system $A\hat{\mathbf{v}} = \mathbf{v}$ where A is a 1000×169 matrix with each component being a random point evaluated in the Fourier form. Along each row is a random point (between 0 and 1) and each column is a unique permutation of n_1 and n_2. Here is an example:

$$A = \begin{bmatrix} e^{i\cdot 2\pi(-6\cdot 0.2 + -6\cdot 0.6)} & e^{i\cdot 2\pi(-6\cdot 0.2 + -5\cdot 0.6)} & \cdots & e^{i\cdot 2\pi(6\cdot 0.2 + 6\cdot 0.6)} \\ e^{i\cdot 2\pi(-6\cdot 0.5 + -6\cdot 0.2)} & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix}$$

We solve the system with backslash, since the system is under-determined MatLab automatically solved the $\hat{\mathbf{v}}$ coefficients with least squares method. We solve three least squares problems as indicated in the problem statement $\hat{\mathbf{v}}_\mathbf{1}$ v's x component, $\hat{\mathbf{v}}_\mathbf{2}$ v's y component and $\hat{\mathbf{f}}$ the initial condition.

We solve for a $K \times K$ matrix where, at each point, we follow the characteristic using the Euler forward method. The computation is, with $\Delta s < 0$:

$$x_{n+1} = x_n + \frac{dx}{ds}(x_n) \cdot \Delta s = x_n + \tilde{v}(x_n) \cdot \Delta s.$$

We compute each x point until reaching the boundary, defined by the lines $(0, y)$ and $(x, 0)$. At these lines, we know that the value of $u(x)$ is 0. We iterate and trace back with the same x values using the Euler backward method:

$$u_{n+1}(x_{n+1}) = u_n - \frac{du}{ds}(x_{n+1}) \cdot \Delta s = u_n - \tilde{f}(x_{n+1}) \cdot \Delta s,$$

Since u_0 is 0 the value of $u(k_1, k_2) = \sum_{i=1}^{n} \tilde{f}(x_i)$

Here are the plots:

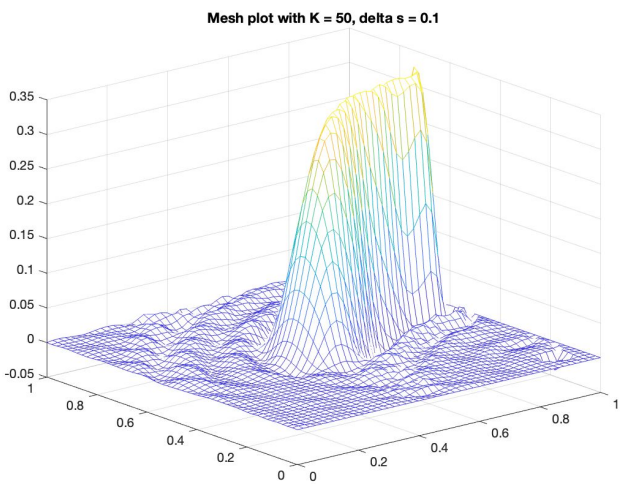

Figure 1: Mesh plot

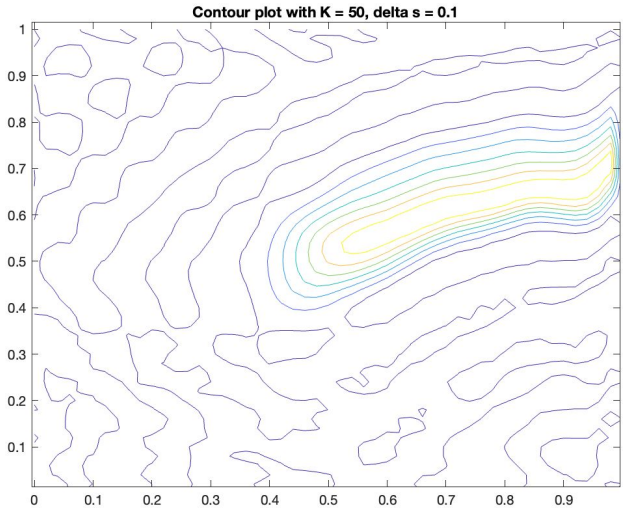

Figure 2: Contour plot

2 Diffusion and Random walk

$$\frac{\partial \rho}{\partial t}(y,t) + \operatorname{div}(v(y)\rho(y,t)) - c\Delta\rho(y,t) = 0, \quad y \in \mathbb{R}^2, t > 0,$$
$$\rho(y,0) = g(y) \geq 0, \quad y \in \mathbb{R}^2,$$

For the density $\rho : \mathbb{R}^2 \times [0,\infty) \to [0,\infty)$, where c is a positive constant and $v : \mathbb{R}^2 \to \mathbb{R}^2$ is a given velocity field, it can be approximated using difference methods. If $v(y) = v$ is constant, the solution can also be approximated using a Fourier series. In the following task, we will compare the difference method with the Fourier series approximation in the case of $v = 0$ (no wind) and one spatial dimension.

2.1 Analytical Solution

We will compare approximations using Fourier Series and Finite Difference Method for:

$$\frac{\partial}{\partial \tau}u(x,\tau) - \frac{\partial^2}{\partial x^2}u(x,\tau) = 0, \quad \tau > 0, x \in [0,\pi],$$

Boundary and initial conditions:

$$u(x,0) = g(x), \quad x \in [0,\pi]$$
$$u(0,\tau) = 0, \quad \tau > 0$$
$$u(\pi,\tau) = 0, \quad \tau > 0$$

for the two cases:

- Case 1: $g(x) = x(\pi - x)$
- Case 2: $g(x) = \begin{cases} x, & x \in [0,\pi/2) \\ 0, & x \in [\pi/2,\pi] \end{cases}$

First we want to find a basis of eigen-functions for the operator $\frac{\partial^2}{\partial x^2}$ that satisfy our boundary conditions. We achieve this by using a set of sine functions:

$$\{\sin(nx)\}_{n \in \mathbb{Z}^+}$$

Our approach to solving the equation involves assuming:

$$u(x,\tau) = \sum_{n=1}^{\infty} v_n(\tau)\sin(nx)$$

Subsequently, for any arbitrary frequency $n \in \mathbb{Z}^+$, we obtain:

1. We start with the equation: $\frac{\partial}{\partial \tau}v_n(\tau)\sin(nx) + n^2 v_n(\tau)\sin(nx) = 0$.
2. By separating variables, we can isolate the τ-dependent part on the left side: $\frac{\partial}{\partial \tau}v_n(\tau) = -n^2 v_n(\tau)$.
3. Solving this first-order ordinary differential equation yields the solution: $v_n(\tau) = v_n(0)\mathrm{e}^{-n^2\tau}$

At this point, the next step is to express the function $g(x)$ using sine functions.

We can do this by computing the following integral:

$$v_n(0) = b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} g(x)\sin(nx)dx$$

2.1.1 Case 1

$$g(x) = x(\pi - x)$$

$$v_n(0) = b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} g(x)\sin(nx)dx$$

$$v_n(0) = b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} x(\pi - x)\sin(nx)dx$$

Here we use integration by parts to eliminate x from the integrand:

$$= \frac{2}{\pi}\left[-x(\pi - x)\frac{\cos(nx)}{n}\right]_0^{\pi} + \frac{2}{\pi}\int_0^{\pi}(\pi - 2x)\frac{\cos(nx)}{n}dx$$

Observe that:

$$\frac{2}{\pi}\left[-x(\pi - x)\frac{\cos(nx)}{n}\right]_0^{\pi} = 0$$

$$\frac{2}{\pi}\left[-x(\pi - x)\frac{\cos(nx)}{n}\right]_0^{\pi} + \frac{2}{\pi}\int_0^{\pi}(\pi - 2x)\frac{\cos(nx)}{n}dx = \frac{2}{\pi}\int_0^{\pi}(\pi - 2x)\frac{\cos(nx)}{n}dx$$

Here again we can use integration by parts to eliminate $(\pi - 2x)$ from the integrand:

$$= \frac{2}{\pi}\left[(\pi - 2x)\frac{\sin(nx)}{n^2}\right]_0^{\pi} + \frac{2}{\pi}\int_0^{\pi} 2\frac{\sin(nx)}{n^2}dx$$

Observe that:

$$\frac{2}{\pi}\left[(\pi - 2x)\frac{\sin(nx)}{n^2}\right]_0^{\pi} = 0$$

$$\frac{2}{\pi}\left[(\pi - 2x)\frac{\sin(nx)}{n^2}\right]_0^{\pi} + \frac{2}{\pi}\int_0^{\pi} 2\frac{\sin(nx)}{n^2}dx = \frac{2}{\pi}\int_0^{\pi} 2\frac{\sin(nx)}{n^2}dx =$$

$$= \frac{4}{\pi}\int_0^{\pi}\frac{\sin(nx)}{n^2}dx = \frac{4}{\pi}\left[\frac{-\cos(nx)}{n^3}\right]_0^{\pi} = \frac{4}{\pi}\left(\frac{1 - (-1)^n}{n^3}\right)$$

NOTE: Symmetry consideration: $g(x) = x(\pi - x)$ so $g(-x) = -g(x)\ \forall x \in [0, \pi]$.

Since:

$$g\left(\frac{x_1}{x_2}\right) = \sum_{n=1}^{\infty} b_n \sin\left(nx_1\right)$$

Where b_n is given by:

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} g\left(\frac{x_1}{x_2}\right)\sin\left(nx_1\right)dx_1$$

Substituting the Fourier series expansion of $g\left(\frac{x_1}{x_2}\right)$ back into the expression for $u(x, \tau)$, we obtain the general solution as:

$$u(x, \tau) = \frac{4}{\pi}\sum_{n=1}^{\infty}\left(\frac{1 - (-1)^n}{n^3}\right)\mathrm{e}^{-n^2\tau}\sin(nx).$$

2.1.2 Case 2

$$g(x) = \begin{cases} x, x \in [0, \pi/2] \\ 0, x \in [\pi/2, \pi] \end{cases}$$

$$v_n(0) = b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} g(x) \sin(nx) dx$$

We have an odd function $g(-x) = -g(x)$ as well as $sin(-x) = -sin(x)$. So w can write:

$$= 2\frac{1}{\pi} \int_{0}^{\pi} g(x) \sin(nx) dx$$

Now since we have a piece wise defined function we have to to split the integral into two parts.

$$2\frac{1}{\pi} \int_{0}^{\pi/2} x \sin(nx) dx + 2\frac{1}{\pi} \int_{\pi/2}^{\pi} 0 \cdot \sin(nx) dx$$

This simplifies the integral to solving:

$$2\frac{1}{\pi} \int_{0}^{\pi/2} x \sin(nx) dx$$

Here again we use integration by parts to eliminate x from the integrand:

$$\frac{2}{\pi} \int_{0}^{\pi/2} x \sin(nx) dx = \frac{2}{\pi} \left[\frac{-x \cos(nx)}{n} \right]_0^{\pi/2} + \frac{2}{\pi} \int_{0}^{\pi/2} \frac{\cos(nx)}{n} dx =$$

$$= \frac{2}{\pi} \frac{-\pi \cos\left(\frac{\pi n}{2}\right)}{2n} + \frac{2}{\pi} \int_{0}^{\pi/2} \frac{\cos(nx)}{n} dx =$$

$$= -\frac{1}{n} \cos\left(\frac{\pi n}{2}\right) + \frac{2}{\pi} \left[\frac{\sin(nx)}{n^2} \right]_0^{\pi/2} =$$

$$= -\frac{1}{n} \cos\left(\frac{\pi n}{2}\right) + \frac{2}{\pi n^2} \sin\left(\frac{n\pi}{2}\right)$$

The General solution is again given by the:

$$u(x, \tau) = \sum_{n=1}^{\infty} b_n e^{-n^2 \tau} \sin(nx)$$

$$u(x, \tau) = \sum_{n=1}^{\infty} (-\frac{1}{n} \cos\left(\frac{\pi n}{2}\right) + \frac{2}{\pi n^2} \sin\left(\frac{n\pi}{2}\right)) e^{-n^2 \tau} \sin(nx)$$

Observe that for different n we have 4 cases that simplify the expression:

$$\text{if } n \bmod 4 = 0 \Rightarrow -\frac{1}{n}$$

$$\text{if } n \bmod 4 = 1 \Rightarrow \frac{2}{\pi n^2}$$

$$\text{if } n \bmod 4 = 2 \Rightarrow \frac{1}{n}$$

$$\text{if } n \bmod 4 = 3 \Rightarrow -\frac{2}{\pi n^2}$$

2.2 Comparing Numerical Methods: Fourier Series vs. Finite Difference Method

Forward Difference for Time Derivative:

$$\frac{u\left(x_n, \tau_{m+1}\right) - u\left(x_n, \tau_m\right)}{\Delta\tau}$$

Central Difference for Spatial Second Derivative:

$$\frac{u\left(x_{n+1}, \tau_m\right) - 2u\left(x_n, \tau_m\right) + u\left(x_{n-1}, \tau_m\right)}{\Delta x^2}$$

Thus we can approximate $\frac{\partial}{\partial\tau}u(x,\tau) - \frac{\partial^2}{\partial x^2}u(x,\tau) = 0$ as:

$$\frac{u\left(x_n, \tau_{m+1}\right) - u\left(x_n, \tau_{mn}\right)}{\Delta\tau} - \frac{u\left(x_{n+1}, \tau_m\right) + u\left(x_{n-1}, \tau_m\right) - 2u\left(x_n, \tau_m\right)}{\Delta x^2} = 0$$

$$\frac{u\left(x_n, \tau_{m+1}\right) - u\left(x_n, \tau_m\right)}{\Delta\tau} = \frac{u\left(x_{n+1}, \tau_m\right) - 2u\left(x_n, \tau_m\right) + u\left(x_{n-1}, \tau_m\right)}{\Delta x^2}$$

$$u\left(x_n, \tau_{m+1}\right) - u\left(x_n, \tau_m\right) = \Delta\tau\frac{u\left(x_{n+1}, \tau_m\right) - 2u\left(x_n, \tau_m\right) + u\left(x_{n-1}, \tau_m\right)}{\Delta x^2}$$

$$u\left(x_n, \tau_{m+1}\right) = u\left(x_n, \tau_m\right) + \Delta\tau\frac{u\left(x_{n+1}, \tau_m\right) + u\left(x_{n-1}, \tau_m\right) - 2u\left(x_n, \tau_m\right)}{\Delta x^2}$$

We remember our conditions:

$$\begin{array}{lll} u(x,0) & = g(x), & x \in [0,\pi] \\ u(0,\tau) & = 0, & \tau > 0 \\ u(\pi,\tau) & = 0, & \tau > 0. \end{array}$$

2.2.1 Fourier Series

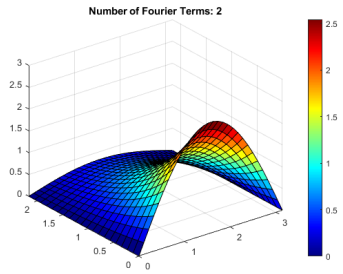

Figure 3: Fourier series with 2 terms

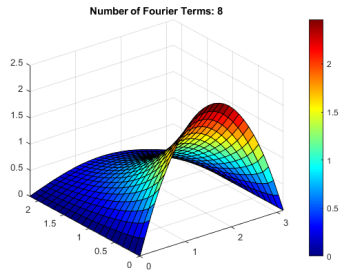

Figure 4: Fourier series with 8 terms

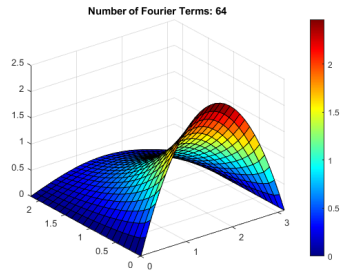

Figure 5: Fourier series with 64 terms

Analysis of the Continuous Function:

- Observe that the difference in the Fourier series approximation is not substantial when a small versus a large number of terms are used.
- Half of the Fourier coefficients for this function are zero, which is characteristic of the function's symmetry and smoothness.
- The remaining non-zero coefficients, have a rapid decay pattern caused by two factors in the equation namely: $\frac{1}{n^3}$ and $\exp\left(-n^2\tau\right)$, indicating that they decrease in magnitude rapidly as the term number n increases.
- We observe the presence of the exponential decay factor $\exp\left(-n^2\tau\right)$.
- The magnitude of the second nonzero coefficient in the series is approximately $\frac{8}{27\pi} \approx 0.09$.
- This rapid decay of coefficients is a significant factor in the Fourier series accurately capturing the function's behavior, even with a relatively low number of terms.
- Thus terms for larger values of n become negligible more swiftly over time τ. As a result, the series converges quickly, and the function's essential characteristics are captured effectively with fewer terms.
- This quick convergence is especially pronounced for smoother functions, as is the case with $g(x) = x(\pi - x)$.

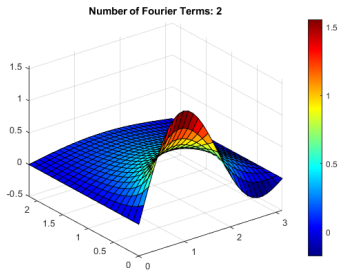

Figure 6: Fourier series with 2 terms

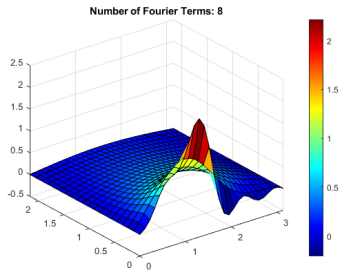

Figure 7: Fourier series with 8 terms

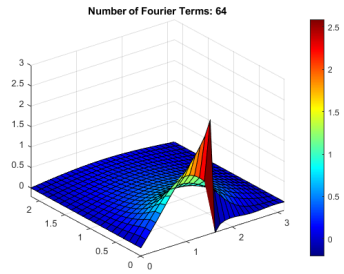

Figure 8: Fourier series with 64 terms

Analysis of the Discontinuous Function:

- Using 2 and 8 terms did not provide an accurate approximation.
- Improved the approximation significantly by using 64 terms.
- Coefficients decrease more slowly.
- Larger number of terms is necessary for an accurate approximation compared to a continuous function.

2.2.2 Finite Difference Method

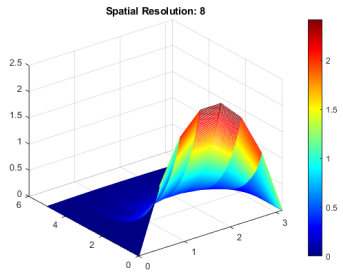

Figure 9: $\Delta x = \frac{\pi}{7}$, $\Delta \tau \approx \frac{5}{400}$

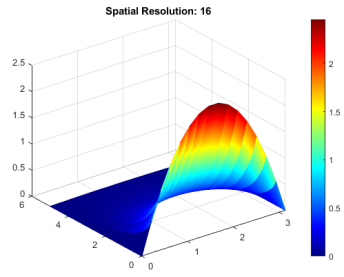

Figure 10: $\Delta x = \frac{\pi}{15}$, $\Delta \tau \approx \frac{5}{800}$

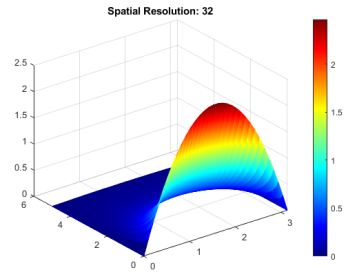

Figure 11: $\Delta x = \frac{\pi}{31}$, $\Delta \tau \approx \frac{5}{1600}$

Analysis of the Continuous Function:

- The Finite Difference method effectively captures the behavior of the continuous function $g(x) = x(\pi - x)$, especially at higher spatial resolutions.
- The results demonstrate convergence towards the true solution as the spatial resolution increases (from 8 to 32).
- The method provides a good approximation across the entire interval $[0, \pi]$, illustrating its suitability for smooth and continuous functions.
- The impact of changing the spatial resolution (Δx) is more pronounced at lower resolutions but diminishes as the resolution increases.
- This method's effectiveness highlights its utility for problems involving smooth initial conditions and diffusion-like phenomena.

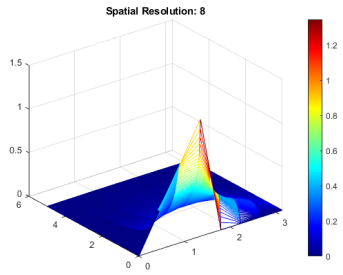

Figure 12: $\Delta x = \frac{\pi}{7}$, $\Delta \tau \approx \frac{5}{400}$

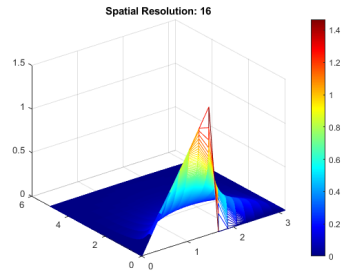

Figure 13: $\Delta x = \frac{\pi}{15}$, $\Delta \tau \approx \frac{5}{800}$

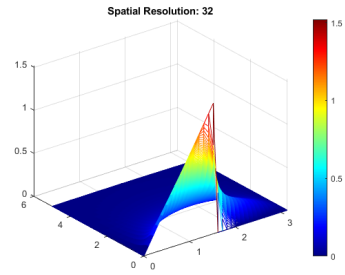

Figure 14: $\Delta x = \frac{\pi}{31}$, $\Delta \tau \approx \frac{5}{1600}$

Analysis of the Discontinuous Function:

- The Finite Difference approximation is particularly effective for the piecewise linear discontinuous function $g(x) = x$ on $[0, \pi/2]$ and 0 on $[\pi/2, \pi]$.
- The linear nature of the function within each piece allows for an almost exact approximation across the interval, except near the point of discontinuity.
- The method shows higher accuracy at higher spatial resolutions, but the difference between various resolutions is less significant compared to the continuous function case.
- The discontinuity poses a challenge, as expected, but the overall approximation quality remains high, especially away from the discontinuity.
- These observations underscore the Finite Difference method's capability in handling functions with sharp transitions, typical in many physical and engineering applications.

Illustration of Instability in Finite Difference Method

- Instability in the Finite Difference method is demonstrated with a scenario where $\Delta\tau/(\Delta x)^2 > \frac{1}{2}$. Until now in all of our scenarios we had a $\Delta\tau/(\Delta x)^2 = 0.2699$
- This instability manifests as significant oscillations, which can grow so intense that the initial condition is obscured, particularly after a certain duration.

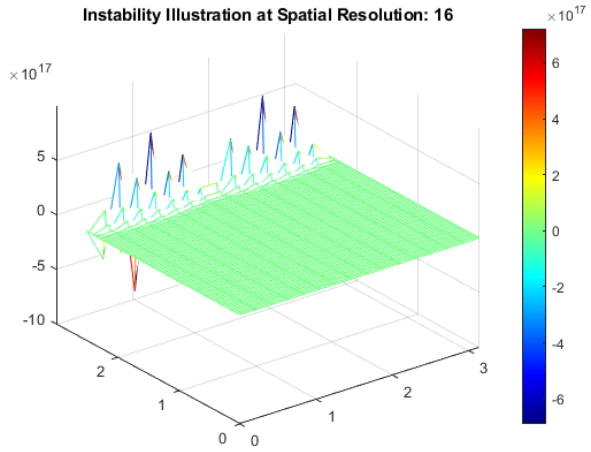

Figure 15: Demonstration where $\Delta\tau/(\Delta x)^2 = 0.75 > \frac{1}{2}$

2.2.3 Summary of Analysis

Continuous vs. Discontinuous Initial Conditions:

- Fourier series is more suitable for continuous and smooth initial conditions
- Finate difference method is more suitable for piece wise linear and discontinuous initial conditions

Efficiency and Computational Complexity:

- The Fourier Series method proved it has greater computational efficiency, requiring fewer computational steps compared to the Finite Difference Method. This is great for large values of τ where we can capture functions behavior with small number of terms!
- On the other hand the Finite Difference method demanded $\mathrm{O}\left(n_x \cdot n_\tau\right)$ computational steps. This means its far less efficient for large values of τ.

Spatial and Temporal Resolution:

- We observe better accuracy as the spatial resolution (Δx) increased. Higher spatial resolution was especially beneficial for capturing fine details, such as the behavior of $g(x)$ near discontinuities.
- In the Finite Difference method, high resolution in τ was crucial for stable solutions since we needed to observe that $\Delta\tau/(\Delta x)^2 \leq \frac{1}{2}$. One must be aware that suboptimal choices of $\Delta\tau$ could lead to instability that could potentially obscure the true solution.

Convergence and Decay Patterns:

- The Fourier Series method showed rapid convergence, even with a small number of terms. The decay pattern of coefficients for smooth functions, leading to quick convergence over time τ while piece-wise had converged slower.

2.3 Random walk

2.3.1 Plots

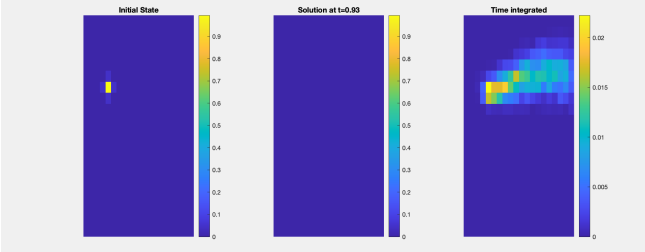

Figure 16: c = 0.02

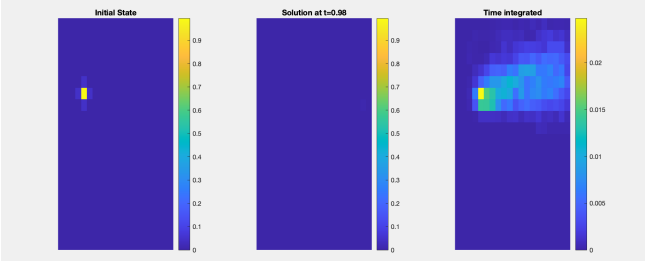

Figure 17: c = 0.04

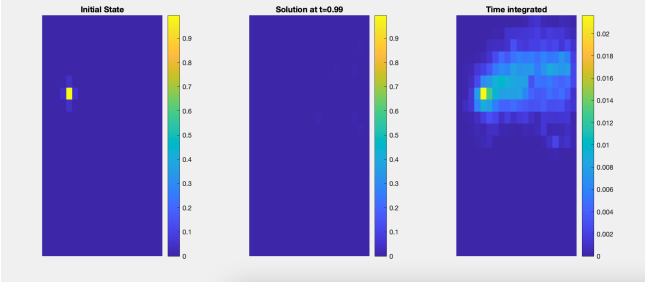

Figure 18: c = 0.06

$$\frac{\partial \rho}{\partial t}(y,t) + \text{div}(v(y)\rho(y,t)) - c\Delta\rho(y,t) = 0$$

2.3.2 Discussion

In the plots we see that when we increase the diffusion coefficient c the concentration of smoke is more spread out. This phenomenon is particularly noticeable for times $t > 0.2$. Higher value of c implies a greater contribution of the random term in the final outcome, aligning with the physical interpretation of the diffusion constant. Essentially, the diffusion constant is indicative of the extent to which the density spreads out. From a physical standpoint, this makes sense as the term $\delta\rho$ in our analysis represents the rate of change of diffusion relative to convection. Thus, if we increase this coefficient results will be enhanced diffusion, causing the smoke to spread over larger area.

3 Appendix

3.1 Problem 2

3.1.1 generate$_{f}ourier_{r}ow.m$

```
function fourier_row = generate_fourier_row(x, n)
    fourier_row = zeros(1,n);
    col_index = 1;
    for n1 = -6:6
        for n2 = -6:6
            fourier_row(col_index) = exp(2*pi*1i*(n1*x(1) + n2*x(2)));
            col_index = col_index + 1;
        end
    end
end
```

3.1.2 main.m

```
fourierMatrix = zeros(1000, 169);

randomx = rand(1000, 2); % 1000 random points in [0,1]x[0,1]

index = 1;

for n1 = -6:6
    for n2 = -6:6
        for i = 1:1000
            x1 = randomx(i, 1);
            x2 = randomx(i, 2);
            fourierMatrix(i, index) = exp(2*pi*1i*(n1*x1 + n2*x2));
        end
        index = index + 1;
    end
end

f = zeros(1000, 1);

for i = 1:1000
    if norm(randomx(i, :) - [0.5, 0.5]) < 0.1
        f(i) = 1;
    else
        f(i) = 0;
    end
end

vx = randomx(:, 2);
vy = 1-randomx(:, 1);

v_x_hat = fourierMatrix \ vx;
v_y_hat = fourierMatrix \ vy;
f_hat = fourierMatrix \ f;


K = 50;
kh = 1/K; %step size for the respective value in the domain
h = 0.1;

U_matrix = getUVValues(K, h, [v_x_hat v_y_hat], f_hat); %calculate the u values
```

```
[x, y] = meshgrid(0:kh:1);

mesh(x, y, U_matrix'); %U_matrix is transposed to get the correct orientation was wrong without
title('Mesh plot with K = 50');
hold on;

figure;
contour(x, y, U_matrix');
title('Contour plot with K = 50');
```

3.1.3 getUValues.m

```
function uValues = getUValues(K, h, v_hat, f_hat)
    uValues = zeros(K, K);
    v_bar = @(x) generate_fourier_row(x, 169) * v_hat;
    euler_forward = @(x) x - h * real(v_bar(x));   % real to make it v_tilda
    f_bar = @(x) generate_fourier_row(x, 169) * f_hat;
    % check lab notes
    for k1 = 1:K
        for k2 = 1:K
            xn = [k1/K, k2/K];
            temp_xn = xn;

            while (xn(1)> 0 && xn(2)> 0)
                xn = euler_forward(xn);  % if xn in the rand its 0
                temp_xn = [temp_xn; xn]; % Save all the values of xn to calculate it back
            end

            temp_u = 0;
            for l = 1:size(temp_xn, 1)
                temp_u = temp_u + real(f_bar(temp_xn(l, :)));  %real to make it tilda
            end                                                % order doesnt matter since its just a sum
            temp_u = temp_u * h;
            uValues(k1, k2) = temp_u;
        end
    end
uValues = [uValues; zeros(1, K)];
uValues = [zeros(K+1, 1), uValues]; %BVP
end
```

3.2 Problem 3

3.2.1 FourierCoefficients.m

```
function coefficients = CalculateFourierCoefficients(numberOfCoefficients, conditionType)
    % CalculateFourierCoefficients
    % Calculates Fourier coefficients based on the specified initial condition type.
    %
    % Arguments:
    % numberOfCoefficients - Number of Fourier coefficients to compute
    % conditionType - Type of initial condition (0 for g(x) = x(pi-x), otherwise g(x) = x on [0,pi/2],
    %
    % Returns:
    % coefficients - Array of calculated Fourier coefficients

    coefficients = zeros(1, numberOfCoefficients);
    if conditionType == 0
        for termIndex = 1:numberOfCoefficients
            if mod(termIndex, 2) == 0
                coefficients(termIndex) = 0;
            else
```

```
                coefficients(termIndex) = 8/(pi * termIndex^3);
            end
        end
    else
        for termIndex = 1:numberOfCoefficients
            switch mod(termIndex, 4)
                case 0
                    coefficients(termIndex) = -pi/(2 * termIndex);
                case 1
                    coefficients(termIndex) = 1/(termIndex^2);
                case 2
                    coefficients(termIndex) = pi/(2 * termIndex);
                case 3
                    coefficients(termIndex) = -1/(termIndex^2);
            end
        end
    end
end
```

3.2.2 PlotFourierSeries.m

```
function PlotFourierSeries(initialConditionType)
    % PlotFourierSeries
    % Plots the Fourier series approximation for different numbers of terms.
    %
    % Arguments:
    % initialConditionType - Flag for initial condition type
    % (0 for g(x) = x(pi-x), otherwise g(x) = x on [0,pi/2], 0 on [pi/2, pi])
    % Test 123
    termsArray = [2, 8, 64]; % Number of Fourier terms
    spatialPoints = 30;        % Number of spatial points
    timePoints = 20;           % Number of time points
    spatialStep = pi/(spatialPoints - 1);  % Spatial step size
    timeStep = spatialStep;                % Time step size

    xSpace = linspace(0, pi, spatialPoints);
    tSpace = linspace(0, timeStep*timePoints, timePoints);
    [XGrid, TGrid] = meshgrid(xSpace, tSpace);

    for numTerms = termsArray
        fourierCoefficients = FourierCoefficients(numTerms, initialConditionType);
        seriesValues = zeros(spatialPoints, timePoints);
        for timeIndex = 1:timePoints
            currentTime = (timeIndex - 1)*timeStep;
            for spatialIndex = 1:spatialPoints
                currentX = (spatialIndex - 1)*spatialStep;
                for termIndex = 1:numTerms
                    seriesValues(spatialIndex, timeIndex) = seriesValues(spatialIndex, timeIndex) + ...
                        fourierCoefficients(termIndex)*sin(termIndex*currentX)*exp(-1*termIndex^2*curre
                end
            end
        end

        figure; % Create a new figure window
        colormap jet; % Set colormap
        surf(XGrid', TGrid', seriesValues); % Create surface plot

        title(sprintf('Number of Fourier Terms: %d', numTerms)); % Add title displaying number of Fouri
        colorbar; % Add colorbar to display the heat legend
    end
end
```

3.2.3 FiniteDifferenceSolver.m

```
function solutionMatrix = SolveDiffusionEquation(spatialPoints, timePoints, simulationEndTime, conditio
    % SolveDiffusionEquation
    % Solves the diffusion equation using the finite difference method.
    %
    % Arguments:
    % spatialPoints   - Number of spatial points
    % timePoints      - Number of time points
    % simulationEndTime - End time for the simulation
    % conditionType    - Flag for initial condition type
    %                    (0 for g(x) = x(pi-x), otherwise g(x) = x on [0,pi/2], 0 on [pi/2, pi])
    %
    % Returns:
    % solutionMatrix - Matrix of solution points

    solutionMatrix = zeros(spatialPoints, timePoints);
    spatialStep = pi / (spatialPoints - 1);
    timeStep = simulationEndTime / (timePoints - 1);

    % Set initial conditions
    if conditionType == 0
        for spatialIndex = 1:spatialPoints
            xPosition = spatialStep * (spatialIndex - 1);
            solutionMatrix(spatialIndex, 1) = xPosition * (pi - xPosition);
        end
    else
        for spatialIndex = 1:spatialPoints
            xPosition = spatialStep * (spatialIndex - 1);
            if xPosition < pi / 2
                solutionMatrix(spatialIndex, 1) = xPosition;
            else
                solutionMatrix(spatialIndex, 1) = 0;
            end
        end
    end

    % Compute finite difference
    for timeIndex = 2:timePoints
        for spatialIndex = 2:spatialPoints - 1
            solutionMatrix(spatialIndex, timeIndex) = solutionMatrix(spatialIndex, timeIndex - 1) + ...
                timeStep / (spatialStep^2) * (solutionMatrix(spatialIndex + 1, timeIndex - 1) - ...
                2 * solutionMatrix(spatialIndex, timeIndex - 1) + solutionMatrix(spatialIndex - 1, time
        end
    end
end
```

3.2.4 PlotFiniteDifference.m

```
function PlotFiniteDifference(initialConditionType)
    % PlotFiniteDifference
    % Plots the finite difference solution for different spatial resolutions.
    %
    % Arguments:
    % initialConditionType - Flag for initial condition type
    % (0 for g(x) = x(pi-x), otherwise g(x) = x on [0,pi/2], 0 on [pi/2, pi])

    resolutionLevels = [8, 16, 32]; % Array of spatial resolutions to plot
    for currentResolution = resolutionLevels
        solutionPoints = FiniteDifferenceSolver(currentResolution, 50*currentResolution, 5, initialCond
        spatialAxis = linspace(0, pi, currentResolution);
        [~, timeSteps] = size(solutionPoints);
        timeAxis = linspace(0, 5, timeSteps);
        [XGrid, TGrid] = meshgrid(spatialAxis, timeAxis);

        figure; % Create a new figure window
        colormap jet; % Set colormap
        mesh(XGrid', TGrid', solutionPoints); % Create mesh plot

        title(sprintf('Spatial Resolution: %d', currentResolution)); % Add title displaying spatial res
        colorbar; % Add colorbar to display the heat legend
    end
end
```