Homework 1

MATH 578: Intro to Numerical PDEs

Liam Pohlmann[1]

[1]University of New Mexico, Department of Nuclear Engineering

Due: 11 February 2024

Contents

Premise

In this assignment, we look to analyze the consistency and stability of three finite difference schemes, which are:

1. Forward Time Backward Space:

$$v_i^{n+1} = (1-\lambda)v_i^n + \lambda v_{i-1}^n \tag{A}$$

2. Lax-Friedrichs:

$$v_i^{n+1} = \frac{1}{2}(1-\lambda)v_{i+1}^n + \frac{1}{2}(1+\lambda)v_{i-1}^n \tag{B}$$

3. Leap-Frog:

$$v_1^{n+1} = v_i^{n-1} - \lambda v_{i+1}^n + \lambda v_{i-1}^n \tag{C}$$

These schemes are implemented as solutions to the diffusion equation:

$$u_t + u_x = 0 \quad t \geq 0, \quad x \in [-1, 1] \tag{1}$$

with the initial condition:

$$u(0,x) = f(x) = \begin{cases} f_1(x) = \begin{cases} 1 & -0.5 \leq x \leq 0.5 \\ 0 & \text{elsewhere} \end{cases} \\ f_2(x) = \sin(2\pi x) \end{cases} \qquad t = 0, \quad x \in [-1, 1]$$

and *periodic* boundary conditions:

$$u(t,-1) = u(t,1), \quad t > 0$$

The solutions to this equation are simply:

$$u(t,x) = f(x-t) \tag{2}$$

where f is the initial condition as stated above.

Additionally, we specify *uniform* spacing in both x and t such that $h = \Delta x$, $k = \Delta t$ and $\lambda = k/h$. Lastly, we note that $v_i^n \approx u(ih, nk)$ for $i = -M, -M+1, \ldots, M-1, M$ and $n = 0, 1, \ldots, N$.

For convenience, we rewrite Equations (A) to (C) as:

$$0 = (1-\lambda)v_i^n + \lambda v_{i-1}^n - v_i^{n+1} \tag{A'}$$

$$0 = \frac{1}{2}(1-\lambda)v_{i+1}^n + \frac{1}{2}(1+\lambda)v_{i-1}^n - v_i^{n+1} \tag{B'}$$

$$0 = v_i^{n-1} - \lambda v_{i+1}^n + \lambda v_{i-1}^n - v_i^{n+1} \tag{C'}$$

Part 1: *Show that all three schemes are consistent with the advection equation.*

Definition 1.1 (Consistency)**.** A finite difference scheme is *consistent* if

$$P\phi - P_{k,i}\phi \xrightarrow{h,k\to 0} 0. \tag{3}$$

Using Taylor's expansion, the following are derived:

$$\phi_i^{n+1} = \phi_i^n + k\phi_t + \mathcal{O}(k^2) \tag{4}$$

$$\phi_i^{n+1} = \phi_i^n - h\phi_x + \mathcal{O}(h^2) \tag{5}$$

and

$$\phi_i^{n+1} = \phi_i^n + h\phi_x + \mathcal{O}(h^2) \tag{6}$$

Scheme A:

$$\begin{aligned} P_{k,i}\phi &= (1-\lambda)\phi_i^n + \lambda\phi_{i-1}^n - \phi_i^{n+1} \\ &= (1-\lambda)\phi_i^n + \lambda\left(\phi_i^n - h\phi_x + \mathcal{O}(h^2)\right) - \left(\phi_i^n + k\phi_t + \mathcal{O}(k^2)\right) \\ &= -h\lambda\phi_x + \lambda\mathcal{O}(h^2) - k\phi_t + \mathcal{O}(k^2) \\ &= -k\phi_x + k\mathcal{O}(h) - k\phi_t + \mathcal{O}(k^2) = 0 \end{aligned} \tag{7}$$

By dividing Equation (7) by k and substituting into Definition 1.1, we find:

$$(\phi_t + \phi_x) - (\phi_t + \phi_x + \mathcal{O}(h) + \mathcal{O}(k)) = \mathcal{O}(h) + \mathcal{O}(k) \xrightarrow{h,k\to 0} 0 \quad \blacksquare$$

Scheme B:

$$\begin{aligned} P_{k,i}\phi &= \frac{1}{2}(1-\lambda)\,\phi_{i+1}^n + \frac{1}{2}(1+\lambda)\,\phi_{i-1}^n - \phi_i^{n+1} \\ &= \frac{1}{2}(1-\lambda)\left(\phi_i^n + h\phi_x + \mathcal{O}(h^2)\right) + \frac{1}{2}(1+\lambda)\left(\phi_i^n - h\phi_x + (O)(h^2)\right) + \\ &\quad - \left(\phi_i^n + k\phi_t + (O)(k^2)\right) \\ &= \phi_i^n - \lambda h\phi_x + \mathcal{O}(h^2) - \phi_i^n - k\phi_t + \mathcal{O}(k^2) \\ &= -k\phi_x - k\phi_t + \mathcal{O}(h^2) + \mathcal{O}(k^2) = 0 \end{aligned} \tag{8}$$

Dividing Equation (8) by k, then applying Definition 1.1, we find:

$$\phi_x + \phi_t - \left(\phi_x + \phi_t + \mathcal{O}\left(\frac{h^2}{k}\right) + \mathcal{O}(k)\right) = \mathcal{O}\left(\frac{h^2}{k}\right) + \mathcal{O}(k) \xrightarrow{h,k\to 0} 0 \quad \blacksquare$$

Although k remains in the denominator, we argue that as both h and k approach 0, the numerator decreases more rapidly due to its quadratic nature. Thus, all error terms will go to 0, showing that the scheme is consistent by Definition 1.1.

Scheme C

Note: The Taylor expansions about $\phi_i^{n\pm1}$ have been expanded further to include the quadratic. This adds $\frac{k^2}{2}\phi_{tt}$ to each, changing the final term to $\mathcal{O}(k^3)$.

$$\begin{aligned} P_{k,i}\phi &= \phi_i^{n-1} - \lambda\phi_{i+1}^n + \lambda\phi_{i-1} - \phi_i^{n+1} \\ &= \phi_i^n - k\phi_t + \frac{k^2}{2}\phi_{tt} + \mathcal{O}(k^3) - \lambda\left(\phi_i^n + h\phi_x + \mathcal{O}(h^2)\right) + \\ &\quad + \lambda\left(\phi_i^n - h\phi_x + \mathcal{O}(h^2)\right) - \left(\phi_i^n + k\phi_t + \frac{k^2}{2}\phi_{tt} + \mathcal{O}(k^3)\right) \\ &= -2k\phi_t - 2\lambda h\phi_x + \mathcal{O}(k^3) + k\mathcal{O}(h) + \mathcal{O}(h^2) = 0 \end{aligned} \tag{9}$$

By dividing Equation (9) by $-2k$ and substituting into Definition 1.1, we find:

$$\phi_t + \phi_x - \left(\phi_t + \phi_x + \mathcal{O}(k^2) + \mathcal{O}(h) + \mathcal{O}\left(\frac{h^2}{k}\right)\right) = \mathcal{O}(k^2) + \mathcal{O}(h) + \mathcal{O}\left(\frac{h^2}{k}\right) \xrightarrow{h,k\to 0} 0 \quad \blacksquare$$

Part 2: *State the CFL condition, and show that **A** and **B** are stable if the CFL condition is satisfied.*

In analyzing the stability of a finite-difference scheme, the Fourier Transform lends itself as a valuable tool. Circumventing lengthy derivation, we will simply replace the approximated solution, v_j^n with $g^n \cdot e^{ij\theta}$, where g is the growth factor and $i = \sqrt{-1}$. j is the index in space. The CFL condition is then greatly simplified to:

$$\|g(\theta, h, k)\|^2 \leq (1 + Ck)^2 \tag{10}$$

where C is a constant.

Scheme A:

Beginning with Equation (A'), we replace v_j^n with $g^n \cdot e^{ij\theta}$, then simplify to find:

$$\begin{aligned} 0 &= (1-\lambda)g^n e^{ij\theta} + \lambda g^n e^{i(j-1)\theta} - \underbrace{g^{n+1}}_{=g^n\cdot g} e^{ij\theta} \\ &= g^n e^{ij\theta} \underbrace{\left(1 - \lambda + \lambda e^{-i\theta} - g\right)}_{\text{This must equal 0}} \end{aligned}$$

Solving for g in the braced expression gives:

$$\begin{aligned} g &= 1 - \lambda + \lambda e^{-i\theta} \\ &= 1 - \lambda + \lambda(\cos\theta - i\sin\theta) \\ &= 1 - \lambda + \lambda\cos\theta - \lambda i \sin\theta \end{aligned}$$

Applying Equation (10):

$$\begin{aligned} \|g\|^2 &= (1 - \lambda + \lambda\cos\theta)^2 + (\lambda i \sin\theta)^2 \\ &= 1 - 2\lambda + 2\lambda\cos\theta + \lambda^2 - 2\lambda^2\cos\theta + \lambda^2 \underbrace{\left(\cos^2\theta + \sin^2\theta\right)}_{=1} \end{aligned}$$

Using the fact that $\cos\theta \leq 1$, all terms above cancel, leaving:

$$\|g\|^2 = 1 \leq 1 \quad \blacksquare$$

Scheme B:

Following a similar procedure to Scheme B, we find:

$$\begin{aligned} 0 &= \frac{1}{2}(1-\lambda)g^n e^{i(j+1)\theta} + \frac{1}{2}(1+\lambda)g^n e^{i(j-1)\theta} - g^n \cdot g e^{ij\theta} \\ &= g^n e^{ij\theta} \underbrace{\left[\frac{1}{2}(1-\lambda)e^{i\theta} + \frac{1}{2}(1+\lambda)e^{-i\theta} - g\right]}_{\text{must equal } 0} \end{aligned}$$

Solving for g yields:

$$\begin{aligned} g &= \frac{1}{2}\left(e^{i\theta} + e^{-i\theta}\right) - \frac{1}{2}\lambda\left(e^{i\theta} - e^{-i\theta}\right) \\ &= \cos\theta - \lambda i \sin\theta \end{aligned}$$

Applying Equation (10) gives:

$$\begin{aligned} \|g\|^2 &= \cos^2\theta + \lambda^2\sin^2\theta \\ &= 1 - \sin^2\theta\left(1 - \lambda^2\right), \quad \sin^2\theta \leq 1 \end{aligned}$$

$$\text{so:} \quad \|g\|^2 \leq \lambda^2 \leq (1 + Ck)^2 \quad \blacksquare$$

Part 3: *State a necessary condition for* ***C*** *to be stable.*

To determine the stability of Equation (C), we will follow the same procedure as seen in Section 2. Replacing v_i^n with $g^n e^{ij\theta}$ for Equation (C′), we find:

$$\begin{aligned} 0 &= g^{n-1}e^{ij\theta} - \lambda g^n e^{i(j+1)\theta} + \lambda g^{ne^{i(j-1)\theta}} - g \cdot g^n e^{ij\theta} \\ \text{Write in terms of } g^{n-1}: \quad &= g^{n-1}e^{ij\theta} - \lambda g \cdot g^{n-1} e^{i(j+1)\theta} + \lambda g \cdot g^{n-1} e^{i(j-1)\theta} - g \cdot g^2 \cdot g^{n-1} e^{ij\theta} \\ &= g^{n-1}e^{ij\theta} \underbrace{\left(1 - \lambda g e^{i\theta} + \lambda g^{-i\theta} - g^2\right)}_{\text{must equal } 0} \end{aligned}$$

Solving for g gives:

$$\begin{aligned} 0 &= 1 - g^2 - \lambda g \left(e^{i\theta} - e^{-i\theta}\right) \\ &= g^2 + 2\lambda i g \sin\theta - 1 \end{aligned}$$

To solve, we complete the square and simplify.

$$\begin{aligned} g^2 + 2\lambda i g \sin\theta + (\lambda i \sin\theta)^2 &= 1 + (\lambda i \sin\theta)^2 \\ (g + \lambda i \sin\theta)^2 &= 1 - \lambda^2 \sin^2\theta \\ g &= -\lambda i \sin\theta \pm \sqrt{1 - \lambda^2 \sin^2\theta} \end{aligned}$$

The sign of the discriminant is quite important to the analysis, as it determines how much we can say regarding the stability. If it is negative, we observe:

$$\begin{aligned} \|g\|^2 &= \left(-\lambda \sin\theta \pm \sqrt{\lambda^2 \sin^2\theta - 1}\right)^2 \\ &= 2\lambda^2 \sin^2\theta \mp 2\sqrt{\lambda^2 \sin^2\theta - 1} - 1 \end{aligned}$$

which bounds both θ and λ. However, if we demand the discriminant to be strictly positive, then we find:

$$\begin{aligned} \|g\|^2 &= 1 - \lambda^2 \sin^2\theta + \lambda^2 \sin^2\theta \\ &= 1 \leq (1 + Ck)^2 \end{aligned}$$

showing that we are *unconditionally* stable. This condition can be simplified by realizing $\sin^2\theta \leq 1$, giving:

$$\begin{aligned} 1 - \lambda^2 \sin^2\theta &> 0 \\ \Rightarrow \quad 1 &> \lambda^2 \sin^2\theta \\ &> \lambda^2 \end{aligned}$$

Therefore, for Scheme C to be guaranteed stable, we demand that $\lambda^2 < 1$. Because $\lambda > 0$, it is synonymous to demand $\lambda < 1$.

Part 4: *Implementation*

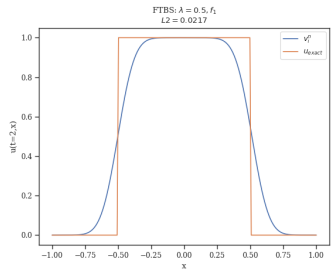
All three schemes (Equations (A) to (C)) were implemented with $\lambda = [0.5, 0.8, 1, 2]$, and the solutions at $t = 2$ are reported for f_1 in Figure 1 and f_2 in Figure 2.

For all cases, the solutions matched well with the theory derived above. That is, all schemes were proven to be stable. Although the error[1] grew quite large for $\lambda = 2$ in *all* cases, the solutions did *not* go to $\pm\infty$ in finite time. In this case, we deduce that not all λ values are applicable to solving a PDE.

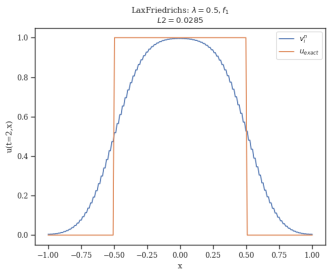
Generally, the error reduced for each scheme as $\lambda \to 1$, aside from Figure 2i, indicating a convergence of error as $k \to h$. However, the error grows rapidly once $k > h$. Figure 2i can be explained by the results in Section 3, where the scheme may become unstable when $\lambda \geq 1$ (this point can be thought of as a bifurcation in stability analysis), which is supported by the many orders of magnitude difference between Scheme **C** and Schemes **A** and **B** once $\lambda = 2$. Overall, it appears clear that Forward-Time-Backward-Space (FTBS) and Leap-Frog were the best methods in terms of accuracy and convergence to their respective best numerical answer. However, it appears that FTBS performed better overall, as it avoided much of the "wobbling" observed by the Leap-Frog scheme. That said, Leap-Frog appears much better suited for solving continuous, smooth schemes, even at lower λ values, implying that a less-fine spatial discretization could be used, which would speed up the computation.

Note: Because Scheme **C** depends on the previous *two* time steps, the first time step taken after the initial conditions is found using Scheme **A**. Additionally, the code was created such that the periodic boundary condition was satisfied for f_1 by demanding $x - t \in [-1, 1]$. This was achieved by creating a `while` loop which shifts the input by the length of the interval (in this case, by adding 2). Python code is available upon request by contacting `lipohlmann@unm.edu`.

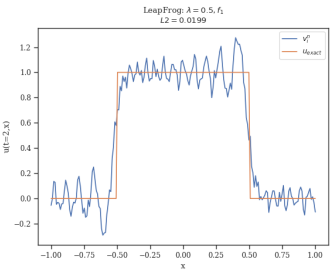
[1] The discrete L2 norm was reported on each subfigure for reference, calculated using the Trapezoid Rule.



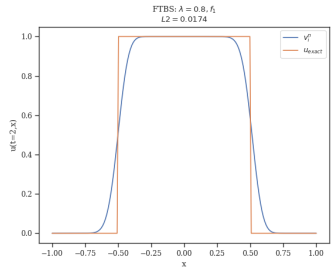
(**a**) FTBS, $\lambda = 0.5$



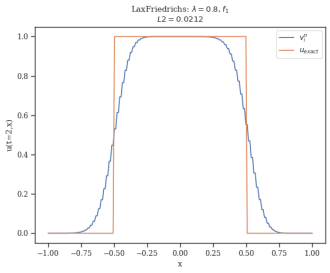
(**b**) Lax-Friedrichs, $\lambda = 0.5$



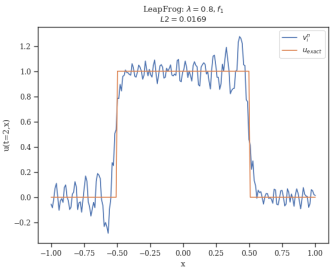
(**c**) Leap-Frog, $\lambda = 0.5$



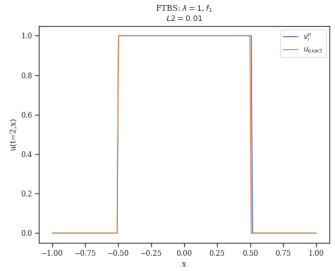
(**d**) FTBS, $\lambda = 0.8$



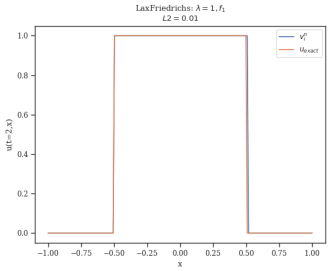
(**e**) Lax-Friedrichs, $\lambda = 0.8$



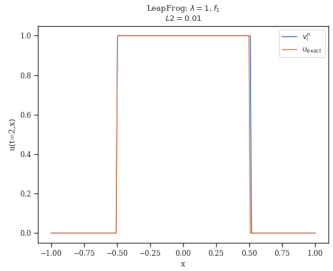
(**f**) Leap-Frog, $\lambda = 0.8$



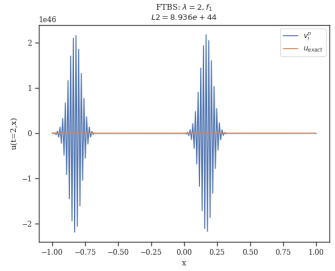
(**g**) FTBS, $\lambda = 1$



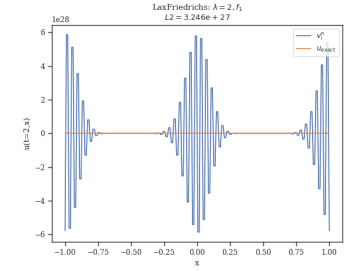
(**h**) Lax-Friedrichs, $\lambda = 1$



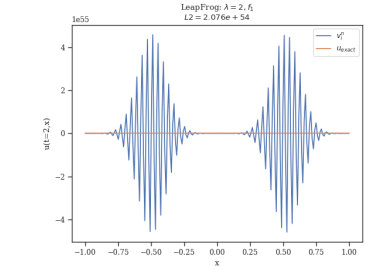
(**i**) Leap-Frog, $\lambda = 1$



(**j**) FTBS, $\lambda = 2$

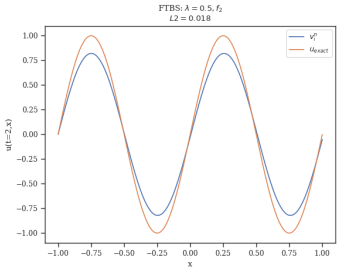


(**k**) Lax-Friedrichs, $\lambda = 2$

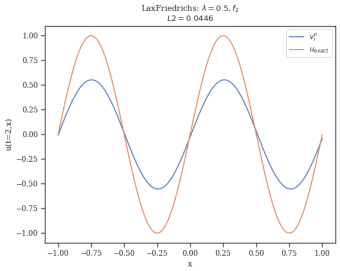


(**l**) Leap-Frog, $\lambda = 2$

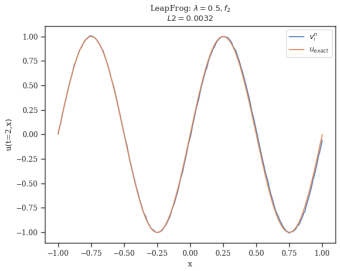
Figure 1: Solutions of Varying λ for f_1



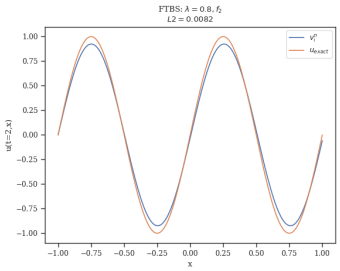
(**a**) FTBS, $\lambda = 0.5$



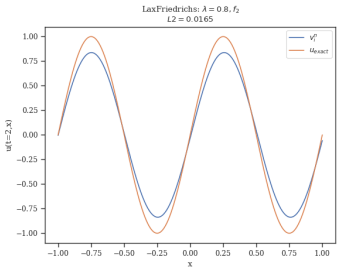
(**b**) Lax-Friedrichs, $\lambda = 0.5$



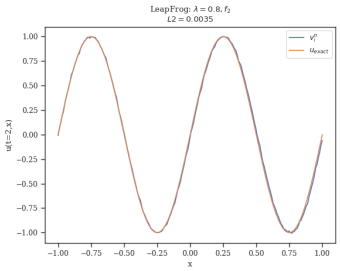
(**c**) Leap-Frog, $\lambda = 0.5$



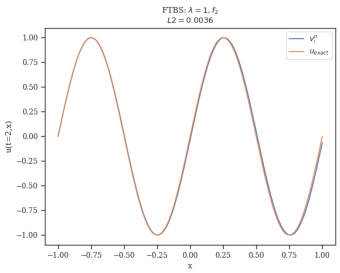
(**d**) FTBS, $\lambda = 0.8$



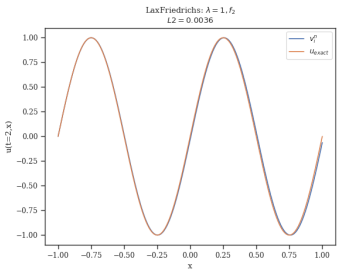
(**e**) Lax-Friedrichs, $\lambda = 0.8$



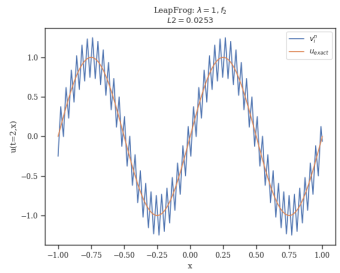
(**f**) Leap-Frog, $\lambda = 0.8$



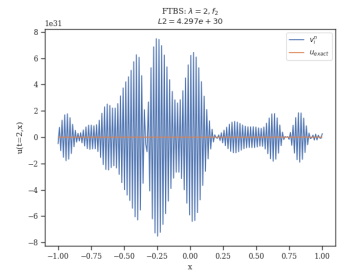
(**g**) FTBS, $\lambda = 1$



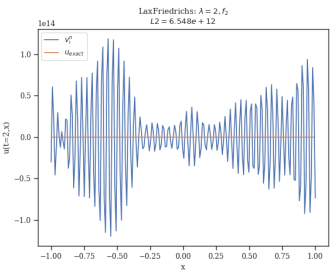
(**h**) Lax-Friedrichs, $\lambda = 1$



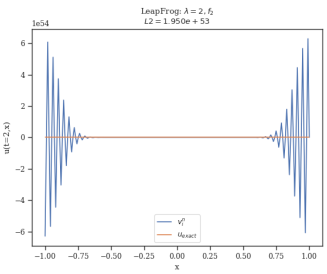
(**i**) Leap-Frog, $\lambda = 1$



(**j**) FTBS, $\lambda = 2$



(**k**) Lax-Friedrichs, $\lambda = 2$



(**l**) Leap-Frog, $\lambda = 2$

Figure 2: Solutions of Varying λ for f_2

Appendix A: Code

```
import numpy as np
from scipy import constants as sc
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

sns.set_theme(
    context="paper",
    style="ticks",
    palette="deep",
    font="serif",
    font_scale=1,
    color_codes=True,
    rc=None,
)
sns.despine()


def f1(x):
    """
    Function returns the pulse function PERIODICALLY
    :param x: input value
    :return:
    """
    while x < -1:
        x += 2
    if -0.5 <= x <= 0.5:
        return 1
    else:
        return 0


def f2(x):
    return np.sin(2 * sc.pi * x)


def calc_u_exact(f, x, t):
```

```
    """
    Returns the exact value of the PDE solution at x,t
    :param f: forcing function
    :param x: location
    :param t: time
    :return:
    """
    return f(x - t)


def FTBS_one_step(Lambda, v0, v1):
    vnp1 = (1 - Lambda) * v1 + Lambda * v0
    return vnp1


def LaxF_one_step(Lambda, v0, v2):
    vnp1 = 1 / 2 * (1 - Lambda) * v2 + 1 / 2 * (1 + Lambda) * v0
    return vnp1


def LeapFrog_one_step(Lambda, v0, vnm1, v2):
    """

    :param Lambda: Lambda
    :param v0: v_(i-1)
    :param vnm1: v_i at previous time step
    :param v2: v_(i+1)
    :return: one step of Leap Frog
    """
    vnp1 = vnm1 - Lambda * v2 + Lambda * v0
    return vnp1


def L2norm(e, h):
    """
    Take the discrete L2 norm of e. Function uses Trapezoid method.
    :param e: error array
    :param h: grid size
    :return:
```

```
    """
    # ensure e has a compatible shape for taking a dot-product
    e = e.reshape(-1, )

    # Task:
    # Return the L2-norm, i.e., the square root of the integral of e^2
    # Assume a uniform grid in x and y, and apply the midpoint rule.
    # Assume that each grid point represents the midpoint of an equally sized region

    l2_norm_squared = np.sum(e ** 2)

    l2_norm = h * np.sqrt(l2_norm_squared)

    if l2_norm > 1000:
        return np.format_float_scientific(l2_norm, precision=3)
    else:
        return round(l2_norm, 4)


h = 0.01
lamb = [0.5, 0.8, 1, 2]

for i, f in enumerate([f1, f2]):
    for L in lamb:
        # we are now going to *attempt* to solve all the schemes at the same time for each
        # lambda value:

        k = L * h

        # create answer arrays:

        # we are just solving for t=2 and the total range of [-1,1]
        FTBS_ans = np.zeros((int(2 / k) + 1,
                             int(2 / h)))
        LaxF_ans = np.zeros_like(FTBS_ans)
        LeapFrog_ans = np.zeros_like(FTBS_ans)

        # create initial data. this will be at the 'top' of the array
        for j in range(FTBS_ans.shape[1]):
```

```
    FTBS_ans[0, j] = f(j * h - 1)
    LaxF_ans[0, j] = f(j * h - 1)
    LeapFrog_ans[0, j] = f(j * h - 1)

# Loop over time:
for s in range(1, FTBS_ans.shape[0]):
    current_time = s * k

    # Loop over space:
    for j in range(FTBS_ans.shape[1]):
        if j == 0:  # Handles the case of needing one point before
            FTBS_ans[s, j] = FTBS_one_step(L, FTBS_ans[s - 1, -1],
                                           FTBS_ans[s - 1, j])
            LaxF_ans[s, j] = LaxF_one_step(L, LaxF_ans[s - 1, -1],
                                           LaxF_ans[s - 1, j + 1])
            if current_time == k:
                LeapFrog_ans[s, j] = FTBS_one_step(L, FTBS_ans[s - 1, -1],
                                                   FTBS_ans[s - 1, j])  # Use FTBS
            else:
                LeapFrog_ans[s, j] = LeapFrog_one_step(L, LeapFrog_ans[s - 1, -1],
                                                       LeapFrog_ans[s - 2, j],
                                                       LeapFrog_ans[s - 1, j + 1])

        elif j == FTBS_ans.shape[
            1] - 1:  # Handles the case of needing one extra point beyond
            FTBS_ans[s, j] = FTBS_one_step(L, FTBS_ans[s - 1, j - 1],
                                           FTBS_ans[s - 1, j])
            LaxF_ans[s, j] = LaxF_one_step(L, LaxF_ans[s - 1, j - 1],
                                           LaxF_ans[s - 1, 0])
            if current_time != k:  # this handles only when LeapFrom can be used
                LeapFrog_ans[s, j] = LeapFrog_one_step(L,
                                                       LeapFrog_ans[s - 1, j - 1],
                                                       LeapFrog_ans[s - 2, j],
                                                       LeapFrog_ans[s - 1, 0])

        else:
            FTBS_ans[s, j] = FTBS_one_step(L, FTBS_ans[s - 1, j - 1],
                                           FTBS_ans[s - 1, j])
            LaxF_ans[s, j] = LaxF_one_step(L, LaxF_ans[s - 1, j - 1],
```

```
                                        LaxF_ans[s - 1, j + 1])
            if current_time == k:
                LeapFrog_ans[s, j] = FTBS_one_step(L, FTBS_ans[s - 1, j - 1],
                                                   FTBS_ans[s - 1, j])
            else:
                LeapFrog_ans[s, j] = LeapFrog_one_step(L,
                                                       LeapFrog_ans[s - 1, j - 1],
                                                       LeapFrog_ans[s - 2, j],
                                                       LeapFrog_ans[s - 1, j + 1])

# Plot figures for current Lambda and initial condition. Need to export each
# figure once done plotting
x_vals = np.linspace(-1, 1, int(2 / h))
u_exact = np.array([calc_u_exact(f, x, 2) for x in x_vals])

L2 = L2norm(FTBS_ans[-1, :] - u_exact, h)
plt.figure()
plt.xlabel('x')
plt.ylabel('u(t=2,x)')
plt.title(f'FTBS: $\lambda = {L},f_{i + 1}$\n$L2={L2}$')
sns.lineplot(x=x_vals, y=FTBS_ans[-1, :], label=r'$v_i^n$', markers=True)
sns.lineplot(x=x_vals, y=u_exact, label=r'$u_{exact}$')
plt.legend()
plt.savefig(fr'figures/FTBS/FTBS_lambda={L},f{i + 1}.png')
plt.close()

L2 = L2norm(LaxF_ans[-1, :] - u_exact, h)
plt.figure()
plt.xlabel('x')
plt.ylabel('u(t=2,x)')
sns.lineplot(x=x_vals, y=LaxF_ans[-1, :], label=r'$v_i^n$', markers=True)
sns.lineplot(x=x_vals, y=u_exact, label=r'$u_{exact}$')
plt.title(f'LaxFriedrichs: $\lambda = {L},f_{i + 1}$\n$L2={L2}$')
plt.legend()
plt.savefig(fr'figures/LaxFriedrichs/LaxFriedrichs_lambda={L},f{i + 1}.png')
plt.close()

L2 = L2norm(LeapFrog_ans[-1, :] - u_exact, h)
plt.figure()
```

```
plt.xlabel('x')
plt.ylabel('u(t=2,x)')
sns.lineplot(x=x_vals, y=LeapFrog_ans[-1, :], label=r'$v_i^n$', markers=True)
sns.lineplot(x=x_vals, y=u_exact, label=r'$u_{exact}$')
plt.title(f'LeapFrog: $\lambda = {L}, f_{i + 1}$\n$L2={L2}$')
plt.legend()
plt.savefig(fr'figures/LeapFrog/LeapFrog_lambda={L},f{i + 1}.png')
plt.close()
```