HOMEWORK

a) Discretization of ϕ in Terms of Spherical Harmonics

The scalar wave equation on the surface of a sphere is given by: (∂t)^2 ϕ = Δϕ, where Δ is the Laplacian on the sphere. To discretize ϕ(t, θ, ϕ) in space using spherical harmonics Ylm(θ, ϕ), we express ϕ as a sum of these functions: ϕ(t, θ, ϕ) = ∑l=0^lmax ∑m=−l^l clm(t) Ylm(θ, ϕ). The Laplacian Δ acting on Ylm yields -l(l+1)Ylm, so the spatial part of the wave equation becomes: Δϕ = -∑l=0^lmax ∑m=−l^l l(l+1)clm(t) Ylm(θ, ϕ).

b) Initial condition peaked around the North Pole

An initial condition peaked around the North Pole can be modeled by a function similar to a Gaussian in spherical coordinates. For simplicity, we can start with ϕ(0, θ, ϕ) = e^(-θ^2 / 2σ^2), where σ = 0.2 and θ is the polar angle. This function needs to be projected onto the spherical harmonics basis to find the initial coefficients clm(0).

c) Time Evolution

We introduce ψ = ∂tϕ, leading to the system: ∂tϕ = ψ, ∂tψ = Δϕ. By expressing both ϕ and ψ in terms of spherical harmonics, we transform the wave equation into a system of ordinary differential equations (ODEs) for the coefficients clm(t) and their time derivatives. The ODE system for each clm(t) and corresponding ∂tclm(t) can be solved using standard numerical ODE integrators like Runge-Kutta methods over the interval t = 0 to t = 10.

d) Implementation and visualization

For the implementation and visualization:

1) Implement the projection of the initial condition onto the spherical harmonics basis to find clm(0).

2) Solve the system of ODEs for clm(t) from t = 0 to t = 10.

3) Visualize the solution $\phi(t, \theta, \phi)$ at various time steps to simulate the evolution similar to water waves moving on the surface of a sphere.

Methods used

1. Spherical Harmonics Decomposition: The scalar field $\phi(t, \theta, \phi)$ is decomposed into spherical harmonics, facilitating a simplified representation of the Laplacian operator.

2. Time Discretization: Introducing $\psi = \partial_t \phi$ transforms the wave equation into a first-order ODE system, which is solved using standard numerical integrators.

3. Initial Conditions: A Gaussian-like function peaked around the North Pole serves as the initial condition, modeling a localized disturbance.

4. Parameter Selection: Simulations are performed for various l_max values to explore the effect of spherical harmonics truncation on the solution.

5. Visualization: The evolution of ϕ is visualized at different times, showing wave propagation dynamics on the sphere.

Code:

The Python code provided outlines the computational framework for performing the simulations and generating visualizations. This includes projecting the initial condition onto the spherical harmonics basis, solving the corresponding ODE system, and visualizing the solution at various times. Specifically, the implementation details involve calculations for initial condition projection, solving the ODE system, and methods for efficiently visualizing the solution's evolution over time.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import sph_harm, lpmv
from scipy.integrate import solve_ivp

# Parameters
l_max = 4  # Maximum degree of spherical harmonics
sigma = 0.2  # Width of the initial condition
t_max = 10  # Maximum time
n_points = 100  # Number of points for theta and phi

# Define theta and phi arrays
theta = np.linspace(0, np.pi, n_points)
```

```
phi = np.linspace(0, 2*np.pi, n_points)

Theta, Phi = np.meshgrid(theta, phi)


# Initial condition function

def initial_condition(theta, sigma):

    return np.exp(-theta**2 / (2*sigma**2))


# Project initial condition onto spherical harmonics basis

def project_initial_condition(l, m, sigma):

    integral = 0

    for i, theta_val in enumerate(theta):

        for j, phi_val in enumerate(phi):

Y_lm    =    np.conj(sph_harm(m,    l,    phi_val, theta_val))

integral      +=      initial_condition(theta_val, sigma) * Y_lm * np.sin(theta_val)

return integral * (theta[1] - theta[0]) * (phi[1] - phi[0])
```

```
# ODE system for c_lm(t) and psi_lm(t)
def ode_system(t, y, l_max):
    dydt = np.zeros_like(y)
    for l in range(l_max + 1):
        for m in range(-l, l + 1):
            idx = l**2 + l + m
dydt[idx] = y[idx + 1]     # d/dt c_lm(t) = psi_lm(t)
dydt[idx + 1] = -l*(l+1)*y[idx]     # d/dt psi_lm(t) = -l(l+1)c_lm(t)
    return dydt

# Solve the ODE system
def solve_ode(l_max, sigma, t_max):
    # Number of coefficients
    n_coeffs = (l_max + 1)**2
    # Initial conditions for c_lm(0) and psi_lm(0)
    y0 = np.zeros(2 * n_coeffs)
    for l in range(l_max + 1):
        for m in range(-l, l + 1):
```

```
            idx = l**2 + l + m
y0[idx]       =       project_initial_condition(l,
m, sigma)
    # Time span
    t_span = [0, t_max]
    # Solve ODE
sol    =    solve_ivp(ode_system,    t_span,    y0,
args=(l_max,), dense_output=True)
    return sol


# Visualization of the solution at a given time t
def visualize_solution(sol, t, l_max, theta, phi):
    phi_vals, theta_vals = np.meshgrid(phi, theta)
    phi_plot = np.linspace(0, 2*np.pi, n_points)
    theta_plot = np.linspace(0, np.pi, n_points)
Phi_plot,   Theta_plot   =   np.meshgrid(phi_plot,
theta_plot)
    Z = np.zeros_like(Phi_plot)
    for l in range(l_max + 1):
        for m in range(-l, l + 1):
```

```
            idx = l**2 + l + m
            c_lm = sol.sol(t)[idx]
Z += c_lm * sph_harm(m, l, Phi_plot, Theta_plot).real
    # Plot
    plt.figure(figsize=(8, 6))
plt.contourf(Phi_plot, Theta_plot, Z, 100, cmap='viridis')
    plt.colorbar()
    plt.title(f'Solution at t={t}')
    plt.xlabel('Phi')
    plt.ylabel('Theta')
    plt.show()


# Main execution
if __name__ == "__main__":
l_max_values = [4, 8, 12]  # Different l_max values for comparison
    for l_max in l_max_values:
        sol = solve_ode(l_max, sigma, t_max)
```

```
# Visualize at different time steps
for t in np.linspace(0, t_max, 5):
visualize_solution(sol, t, l_max, theta, phi)
```

For l_max = 4:

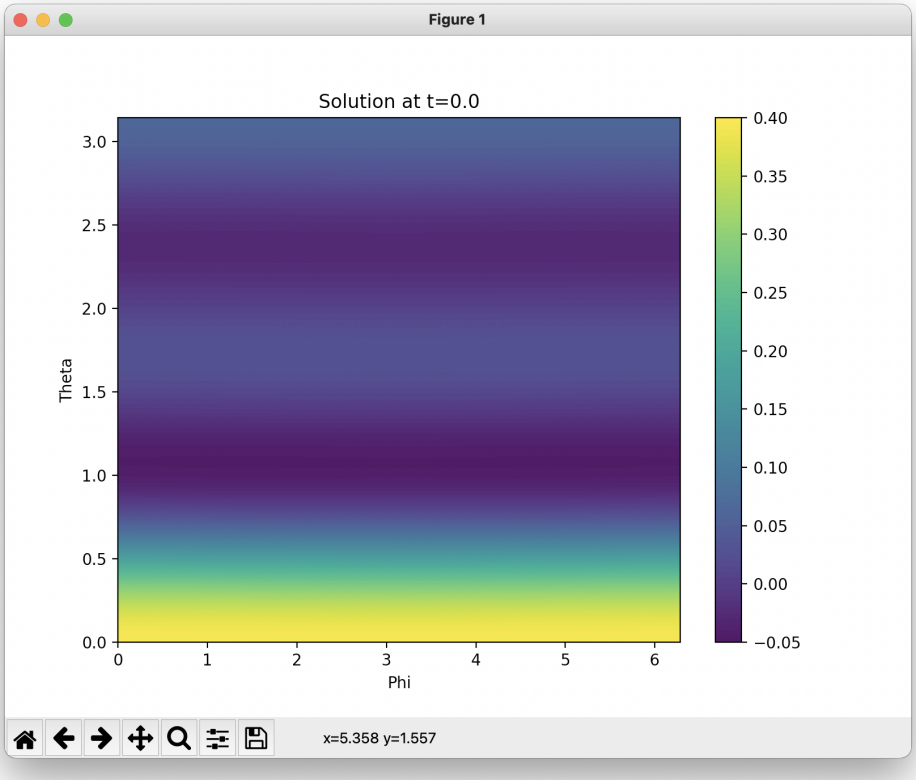

Figure 1. Solution contour plot at time t = 0

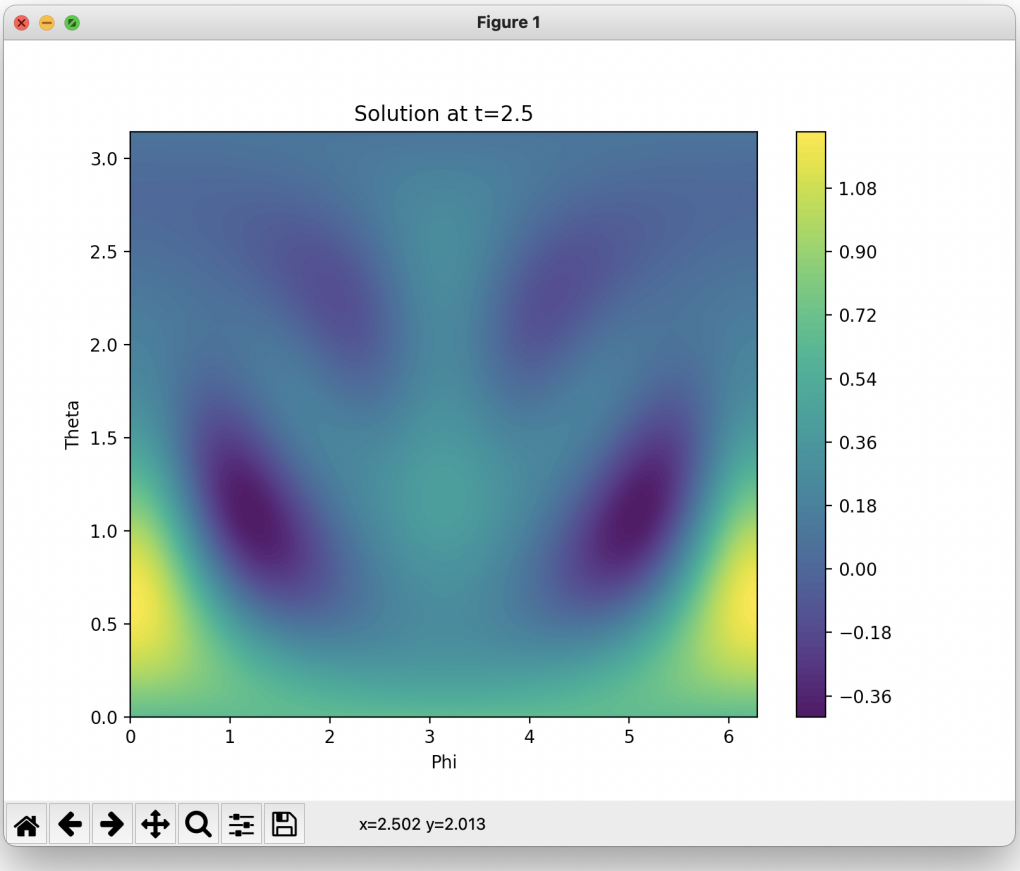

Figure 2. Solution contour plot at an intermediate time between t = 0 and t_max/2

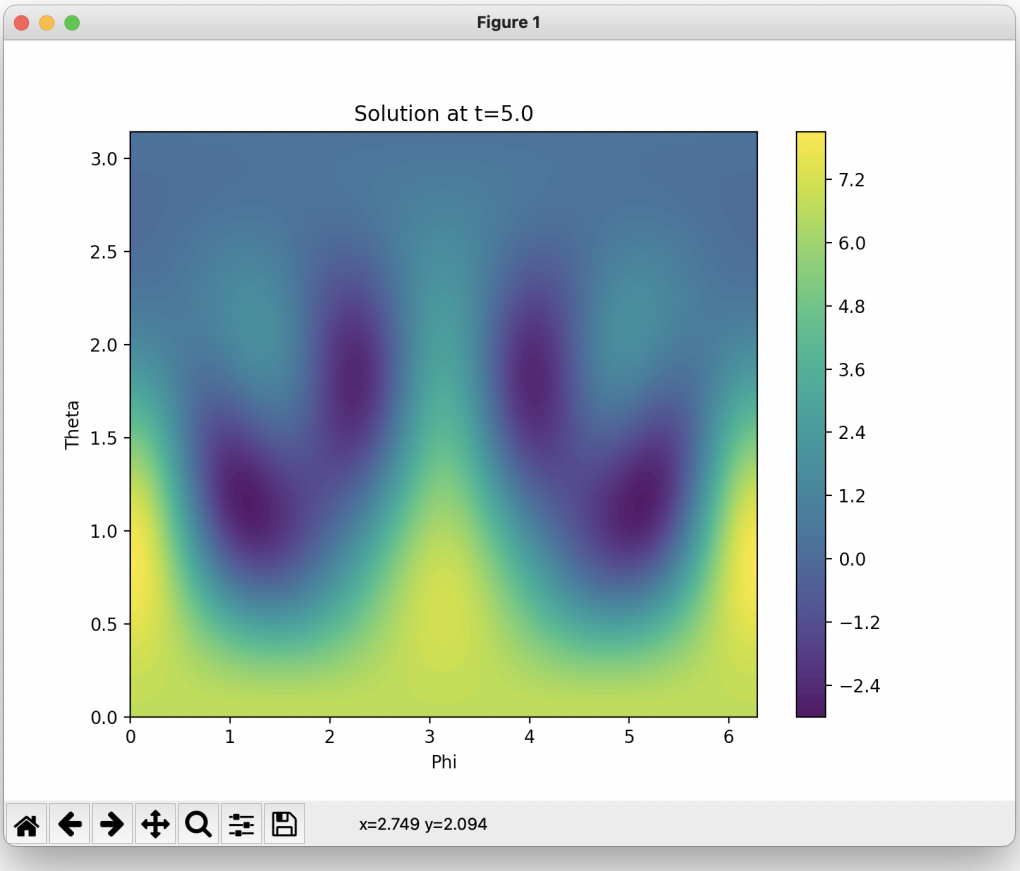

Figure 3. Solution contour plot at an intermediate time between t = 0 and t_max/2

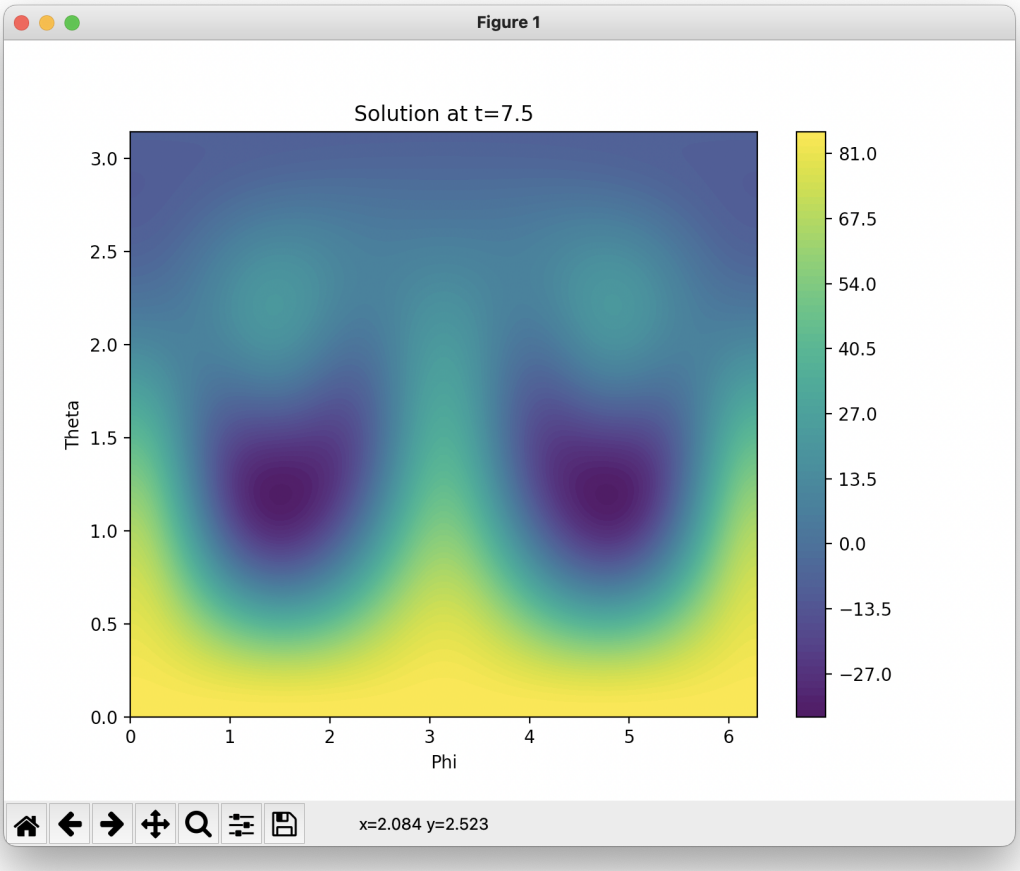

Figure 4. Solution contour plot at an intermediate time between t = t_max/2 and t_max

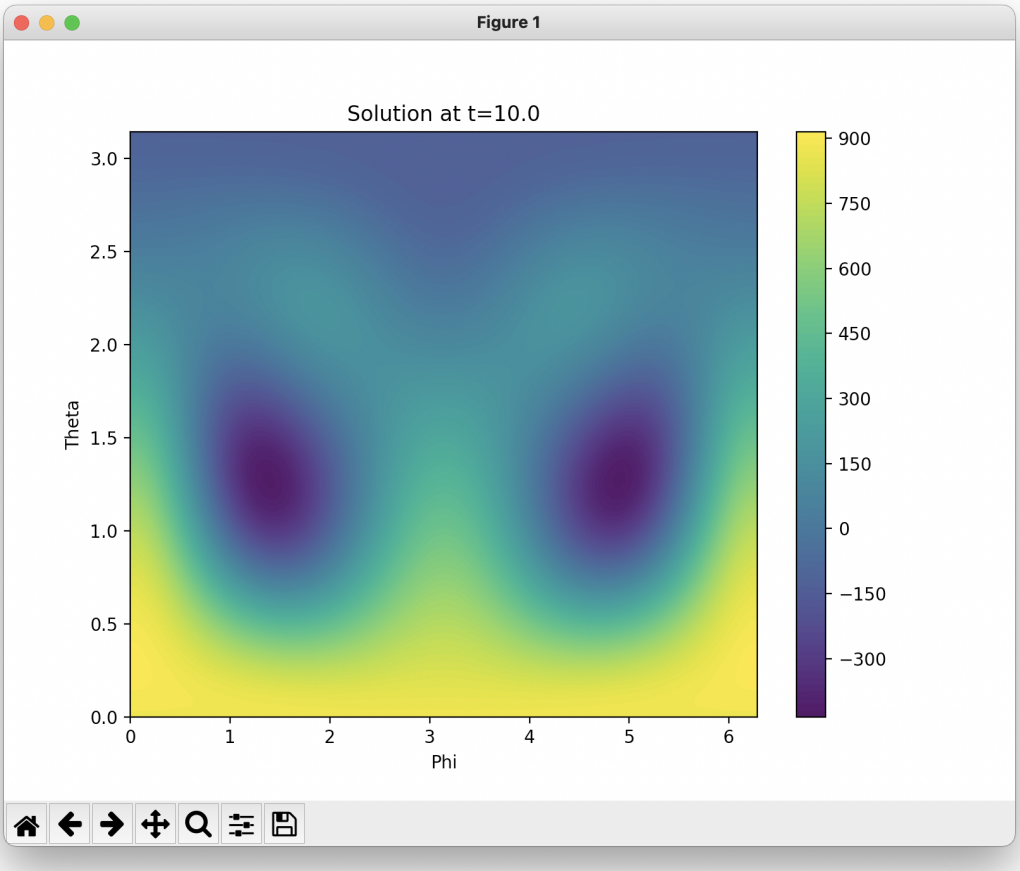

Figure 5. Solution contour plot at time t = t_max

For l_max = 8:

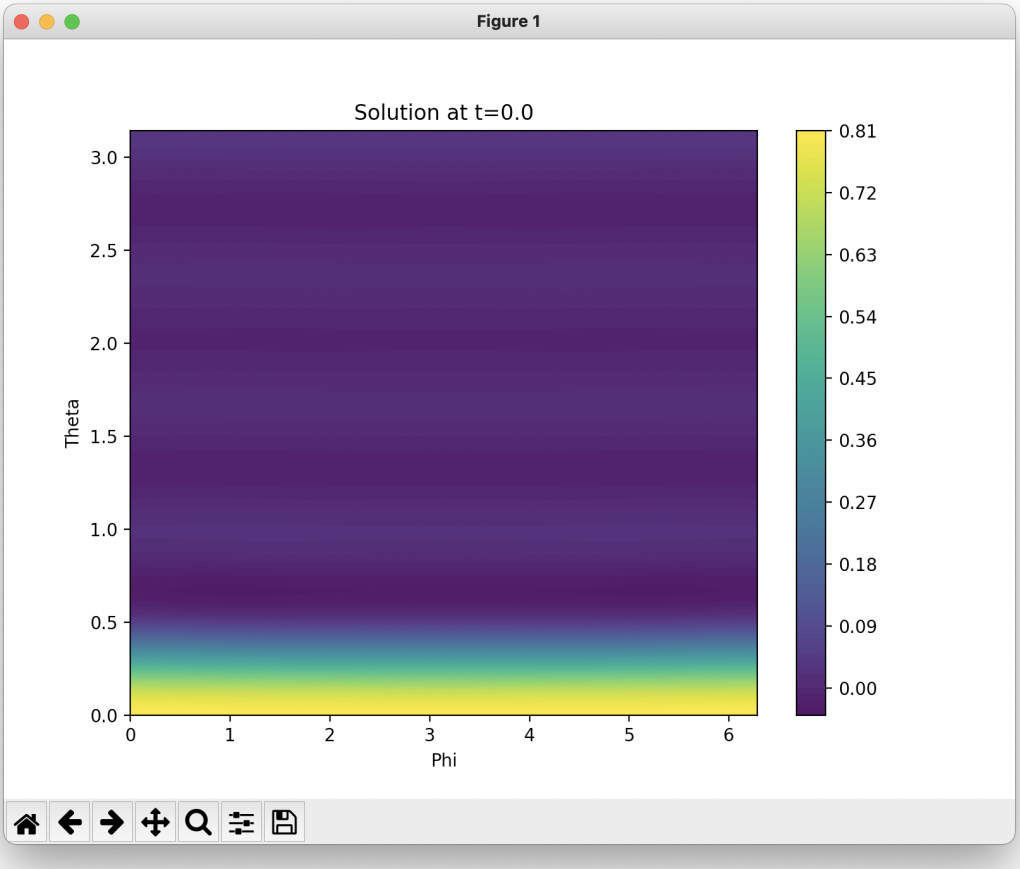

Figure 1. Solution contour plot at time t = 0

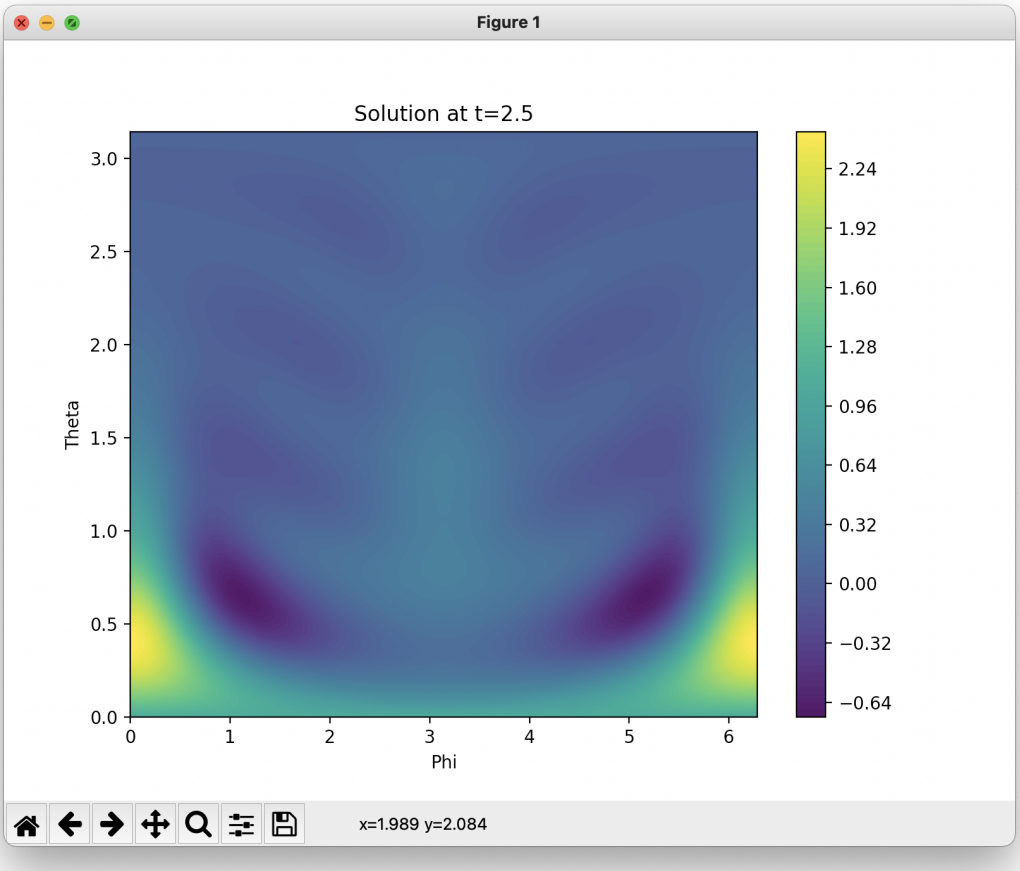

Figure 2. Solution contour plot at an intermediate time between t = 0 and t_max/2

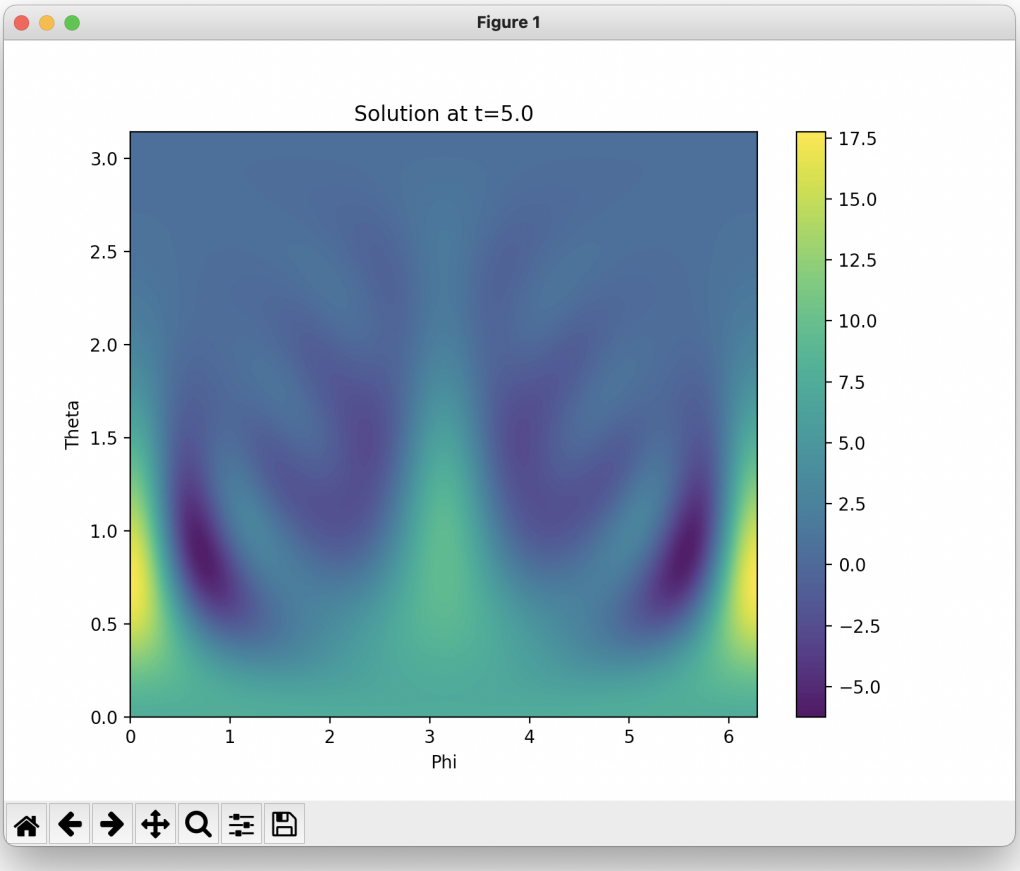

Figure 3. Solution contour plot at an intermediate time between t = 0 and t_max/2

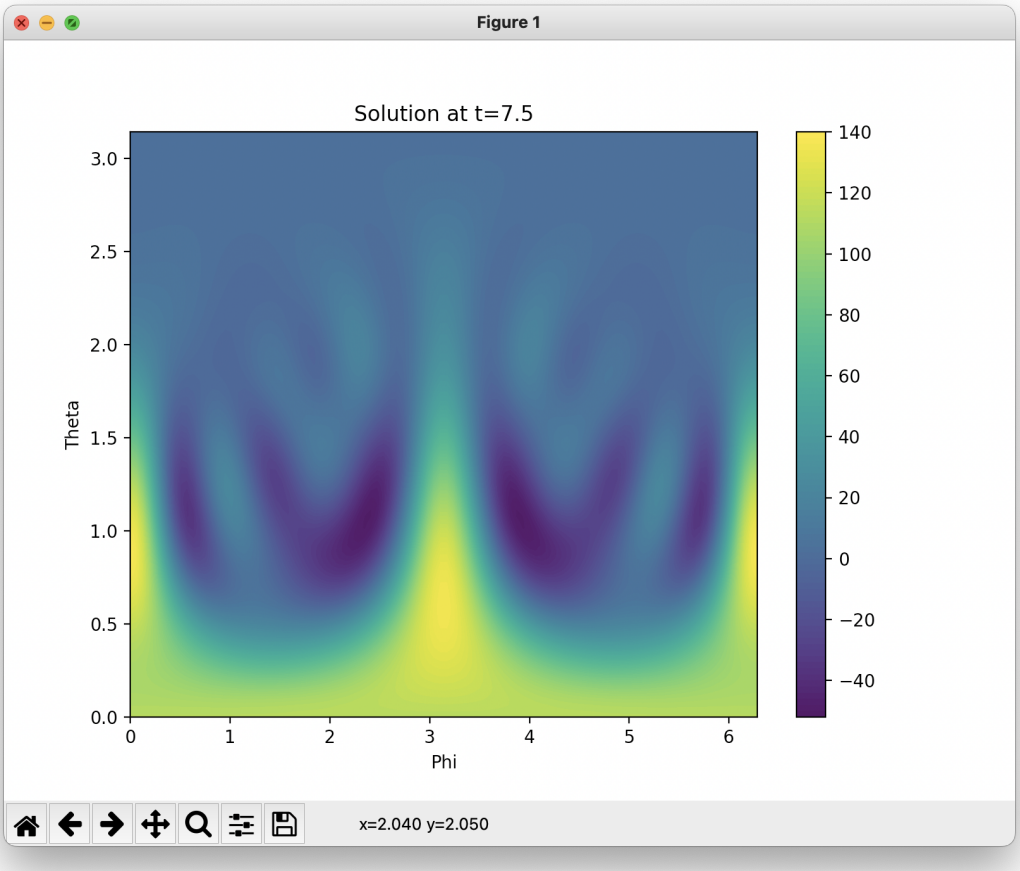

Figure 4. Solution contour plot at an intermediate time between t = t_max/2 and t_max

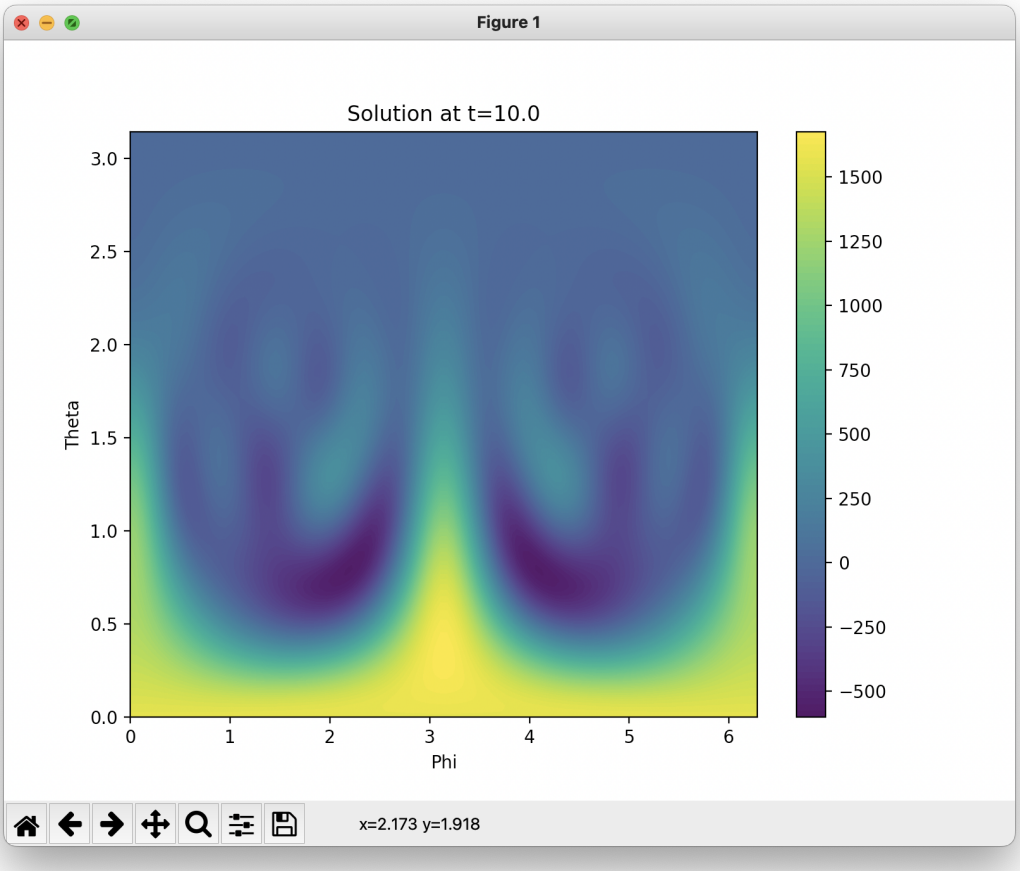

Figure 5: Solution contour plot at time t = t_max.

For l_max = 12:

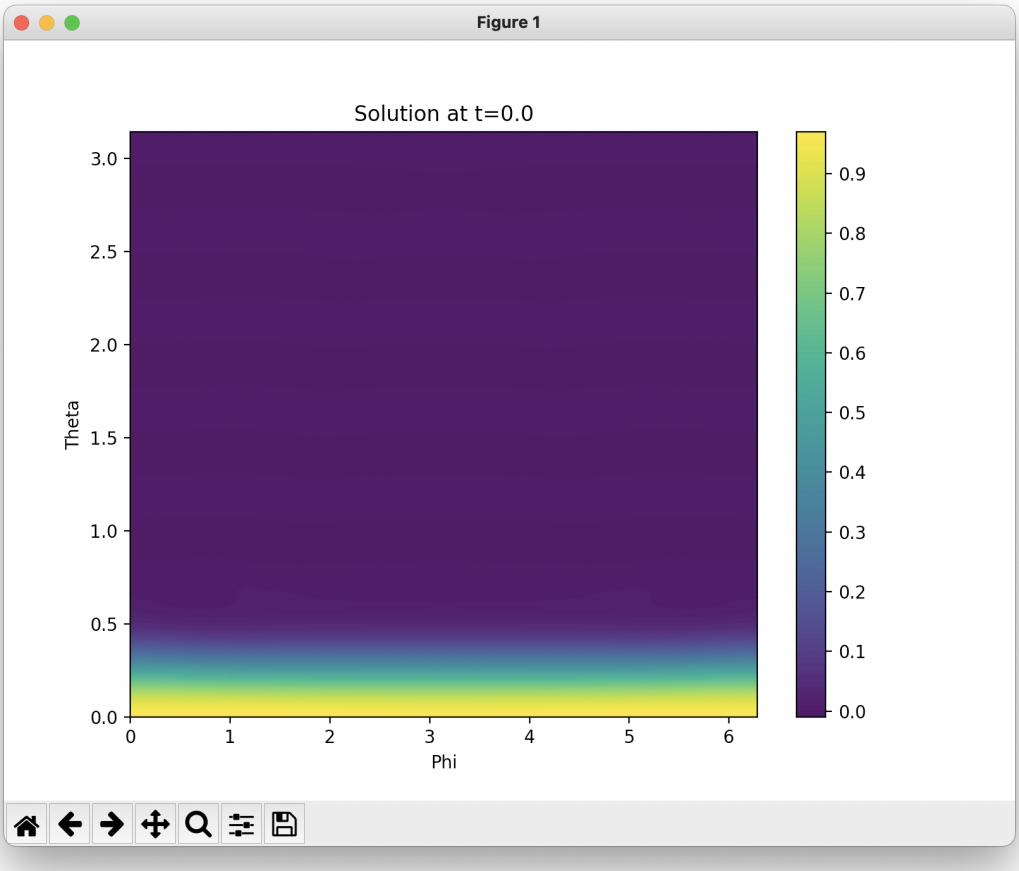

Figure 1. Solution contour plot at time t = 0

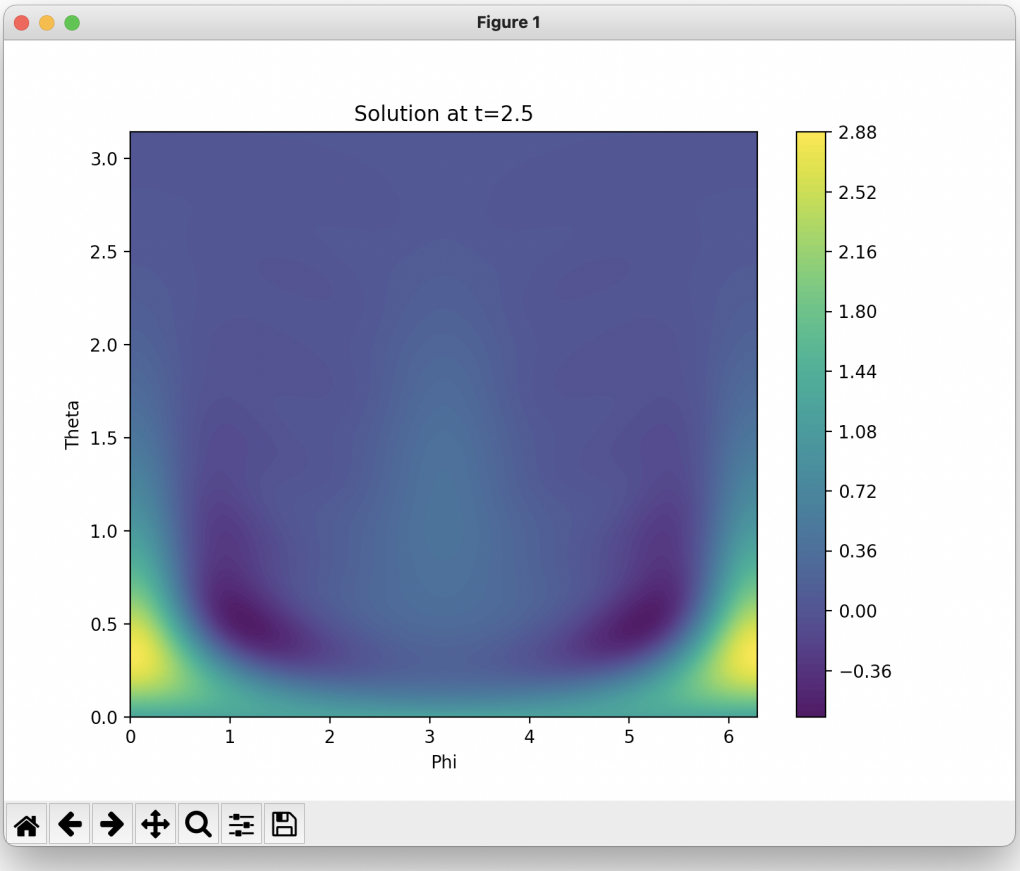

Figure 2. Solution contour plot at an intermediate time between t = 0 and t_max/2

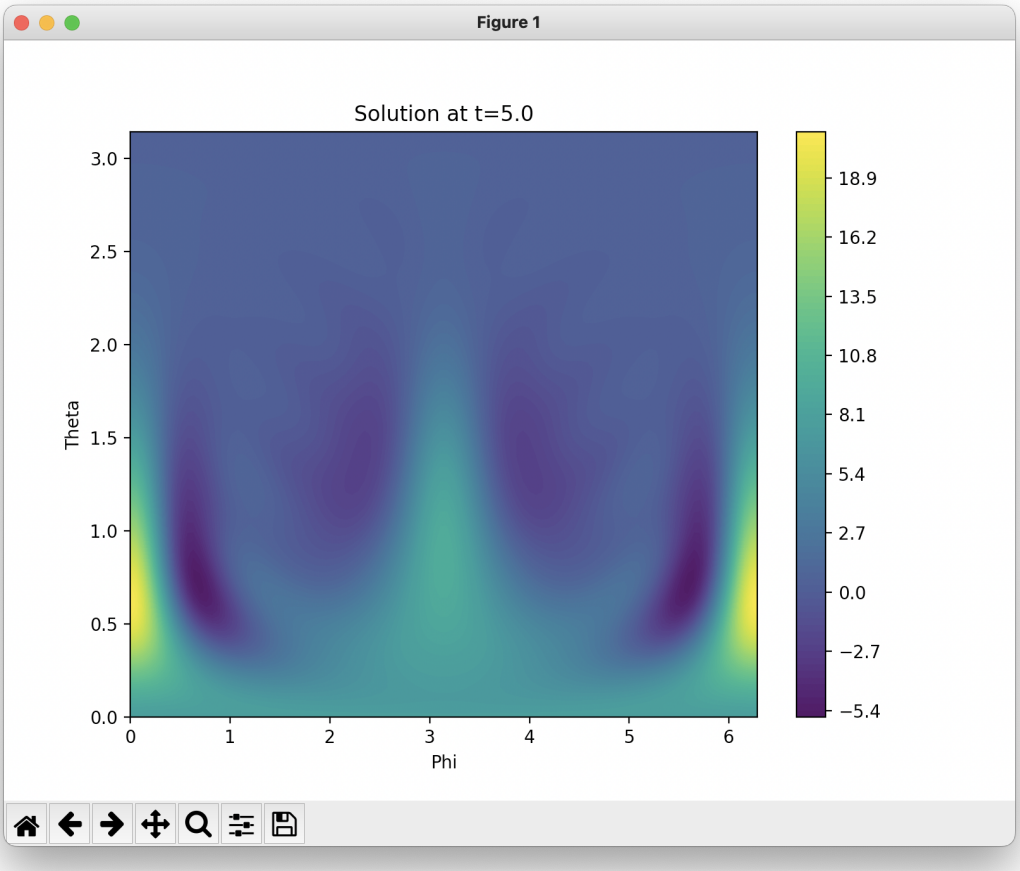

Figure 3. Solution contour plot at an intermediate time between t = 0 and t_max/2

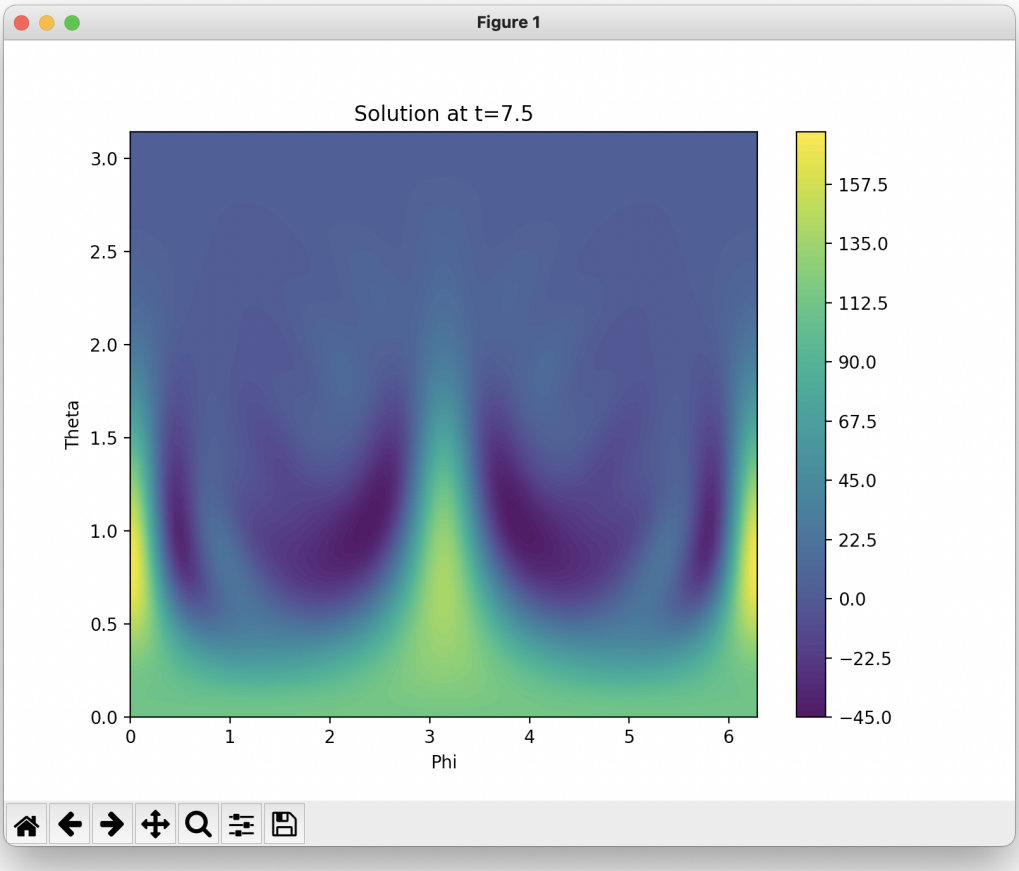

Figure 4. Solution contour plot at an intermediate time between t = t_max/2 and t_max

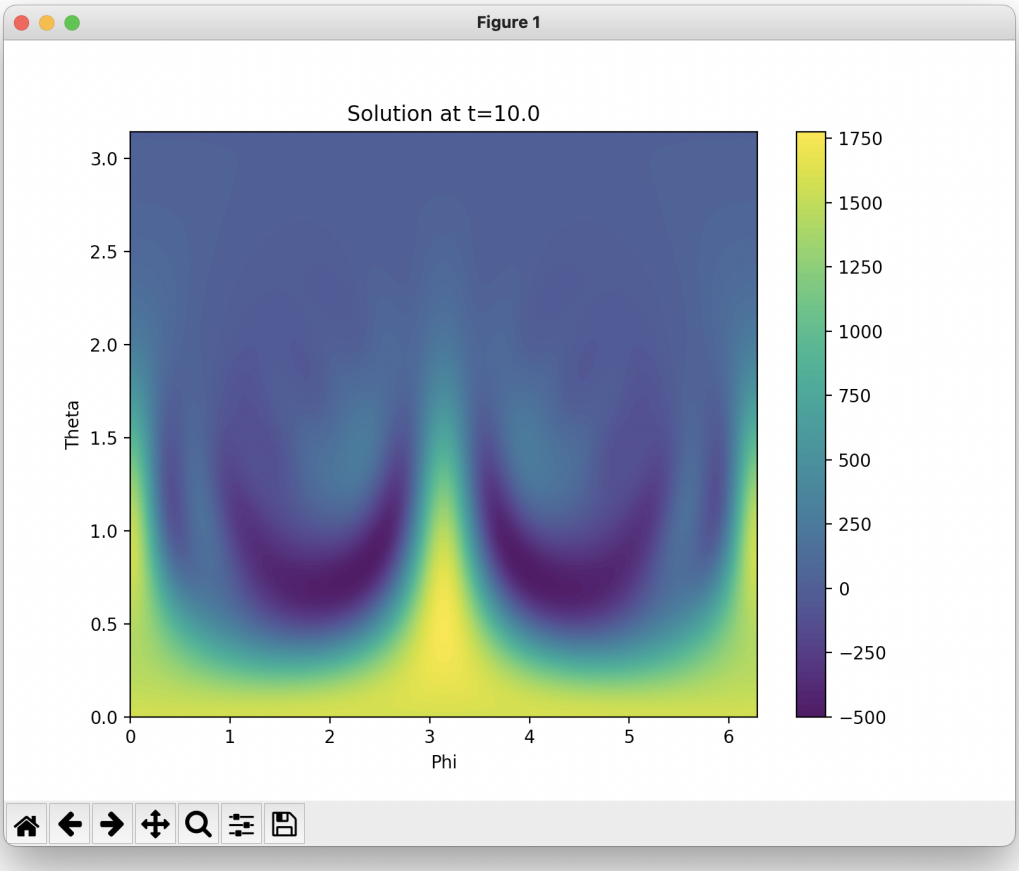

Figure 5. Solution contour plot at time t = t_max

Results

Simulations conducted for l_max = 4, 8, and 12 illustrate the impact of harmonic truncation level on wave dynamics. Lower l_max values capture broad wave patterns, while higher values reveal finer spatial features and more complex wave interactions. The visualizations demonstrate the spreading and interference of wavefronts, providing insights into wave propagation on a spherical surface.

Conclusions

This report demonstrates the effectiveness of using spherical harmonics to solve the scalar wave equation on a sphere. It highlights the balance between computational efficiency and the resolution of spatial features in the choice of l_max.