Numerical Simulation of Solitons in the Nonlinear Klein-Gordon System

1 Introduction

This document outlines the numerical implementation of soliton evolution in the nonlinear Klein-Gordon equation with a ϕ^4 potential. The goal is to explore soliton stability, interactions, and scaling behaviors in the context of the Reciprocal System Theory.

2 Mathematical Framework

The nonlinear Klein-Gordon equation is given by:

$$\frac{\partial^2 \phi}{\partial t^2} - \frac{\partial^2 \phi}{\partial x^2} + m^2 \phi + g\phi^3 = 0 \tag{1}$$

where:

- $\phi(x, t)$ is the scalar field.
- m is a mass-like parameter.
- g is the nonlinear interaction coefficient.

This equation supports **solitonic solutions** that remain stable due to the balance of dispersion and nonlinearity.

3 Numerical Implementation

We discretize the equation using finite differences:

$$\frac{\partial^2 \phi}{\partial t^2} \approx \frac{\phi_i^{n+1} - 2\phi_i^n + \phi_i^{n-1}}{\Delta t^2}, \tag{2}$$

$$\frac{\partial^2 \phi}{\partial x^2} \approx \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{\Delta x^2}. \tag{3}$$

Initial conditions: A standard kink soliton:

$$\phi(x, 0) = \tanh\left(\frac{x}{\sqrt{2}}\right) \tag{4}$$

and an initial velocity perturbation:

$$\left.\frac{\partial\phi}{\partial t}\right|_{t=0} = v\frac{d\phi}{dx} \tag{5}$$

Boundary conditions: Absorbing boundaries are used to prevent artificial reflections.

4 Python Implementation

The numerical scheme is implemented in Python using finite-difference methods:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Parameters
L = 20.0  # Spatial domain size
Nx = 200  # Number of spatial grid points
dx = L / Nx  # Spatial step size
dt = 0.01  # Time step size
Nt = 500  # Number of time steps
m = 1.0  # Mass parameter
g = 1.0  # Nonlinearity coefficient
v = 0.3  # Initial velocity of soliton

# Initialize spatial and temporal grids
x = np.linspace(-L/2, L/2, Nx)
phi = np.tanh(x / np.sqrt(2))  # Initial soliton profile
phi_old = np.tanh((x - v * dt) / np.sqrt(2))  # Apply initial velocity shift
phi_new = np.zeros_like(phi)

# Storage for visualization
phi_evolution = np.zeros((Nt, Nx))

# Time evolution using finite difference scheme
for n in range(Nt):
    d2_phi_dx2 = (np.roll(phi, -1) - 2 * phi + np.roll(phi, 1)) / dx**2
    phi_new = 2 * phi - phi_old + dt**2 * (d2_phi_dx2 - m**2 * phi - g * phi**3
    phi_evolution[n, :] = phi
    phi_old = np.copy(phi)
    phi = np.copy(phi_new)

# Visualization
fig, ax = plt.subplots()
line, = ax.plot(x, phi_evolution[0, :], 'k')
```

```python
def update(frame):
    line.set_ydata(phi_evolution[frame, :])
    ax.set_title(f"Time Step: {frame}")
    return line,

ani = animation.FuncAnimation(fig, update, frames=Nt, interval=30)
plt.xlabel("x")
plt.ylabel("  (x,t)")
plt.title("Soliton Evolution in the Nonlinear Klein-Gordon System")
plt.show()
```

5 Results and Observations

- The soliton remains **stable** throughout the simulation.
- The **initial velocity** $v = 0.3$ induces a slow rightward drift.
- The balance between dispersion and nonlinearity allows the soliton to retain its shape.

6 Next Steps

1. **Soliton Collisions**: Introduce a second soliton with opposite velocity and observe interactions.
2. **Phase Shift Analysis**: Measure displacement before and after collisions.
3. **Energy Conservation**: Verify total energy remains constant.
4. **Scaling Laws**: Explore how soliton properties depend on parameters m and g.