Problem (a)

Device a numerical scheme that will successfully follow the exact 1-soliton solution of Eq (1). Follow this soliton as it performs 3 transits of the spatial region $[0, L]$. The soliton should travel without any distortion. Plot the time evolution of the numerical solution.

Approach to the Solution:

Nonlinear Schrodinger equation (NLS) is a non-linear partial differential equation.

$$i\frac{\partial \psi}{\partial t} + \frac{\partial^2 \psi}{\partial x^2} + |\psi|^2 \psi = 0 \quad -(1)$$

The periodic boundary condition for all the time:

$$\psi(0,t) = \psi(L,t) \quad -(2)$$

From the literature 1-soliton solution of equation (1)

$$\psi_{NLS}(x,t) = a\sqrt{2}\exp\left[i\left\{\frac{bx\sqrt{2}}{2} - \left(\frac{b^2}{4} - a^2\right)t\right\}\right].$$
$$\operatorname{sech}\left[a(x\sqrt{2} - bt)\right] \quad -(3)$$

with two parameters a and b. The parameters independently control the soliton speed $(b/\sqrt{2})$ and soliton amplitude $(a/\sqrt{2})$.

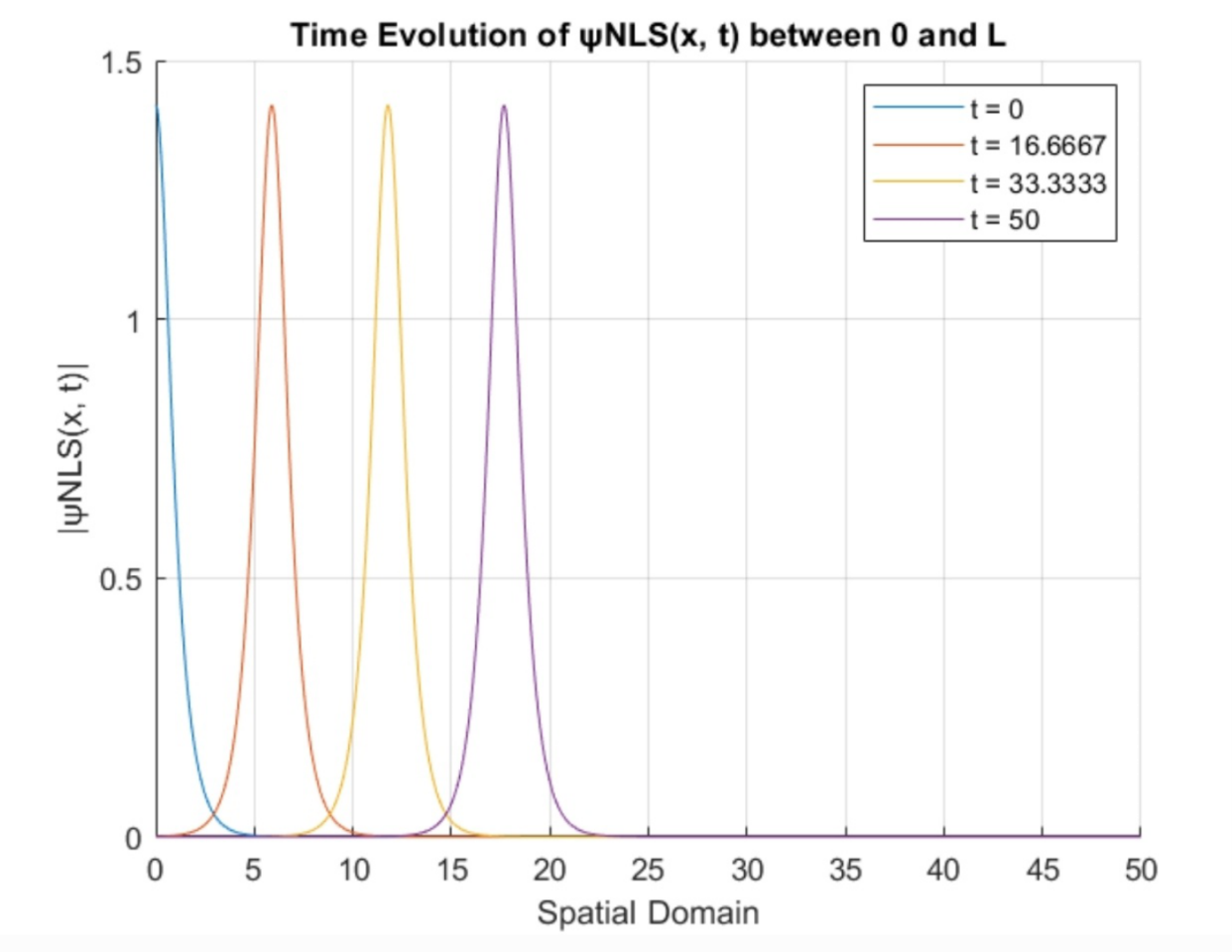
Therefore, following the equation (3) and with user defined a, b and t value we observe the time evolution of soliton.

The following code in written using MATLAB 2020 version.

```
% Define parameters
a = 1.0;
b = 0.5;
L = 50;  % Spatial region [0, L]
time_steps = 4;
num_points = 1000;  % Number of points in the spatial domain

% Define spatial domain
x = linspace(0, L, num_points);

% Initialize plot
figure;
hold on;

% Time evolution
for t = linspace(0, L, time_steps)
    psi = a * sqrt(2) * exp(1i * ((b * x * sqrt(2))/ 2 - (b^2 / 4 - a^2) *
t)) .* sech(a * (x * sqrt(2) - b * t));

    % Plot amplitude (absolute value of psi)
    plot(x, abs(psi), 'DisplayName', ['t = ' num2str(t)]);
end

hold off;
xlabel('Spatial Domain');
ylabel('|ψNLS(x, t)|');
title('Time Evolution of ψNLS(x, t) between 0 and L');
legend('show');
grid on;
```

amplitude
velocity

Equation (3)



From the above figure, we observe the soliton solution over 3 time steps without any distortion. The prefined range for $(0, L) = (0, 50)$ and for this particular problem we defined the amplitude $a = 1.0$ and velocity, $b = 0.5$.

Problem (b)

Now consider a 2-soliton solution having different amplitude and velocities of different sign. That is, the 2 solitons do not overlap and move towards each other for a soliton collision. Numerically follow this dynamics for at least 12 soliton-soliton collisons. Plot some of these results. What one should find is that there is a distinctive spatial induced in the post-collision state. Verify that the spatial shift is the same for each soliton-soliton collison.

Approach to the solution:

The solution for two isolated non-overlapping solitons moving with the same speed towards each other, with one soliton having twice the amplitude of the other, The initial profile

$$\psi(x, t_0 = 0) = a\sqrt{2}\exp\left[i\frac{bx}{2}\right].\operatorname{sech}\left[a(x - x_0)\right] + a_1\sqrt{2}\exp\left[i\frac{b_1 x}{2}\right].\operatorname{sech}\left[a_1(x - x_1)\right]$$

.... Eq (4)

where,

$$a_1 = 2a$$

$$b = -b_1 > 0$$

x_0 and x_1 are the initial location of the soliton peaks.

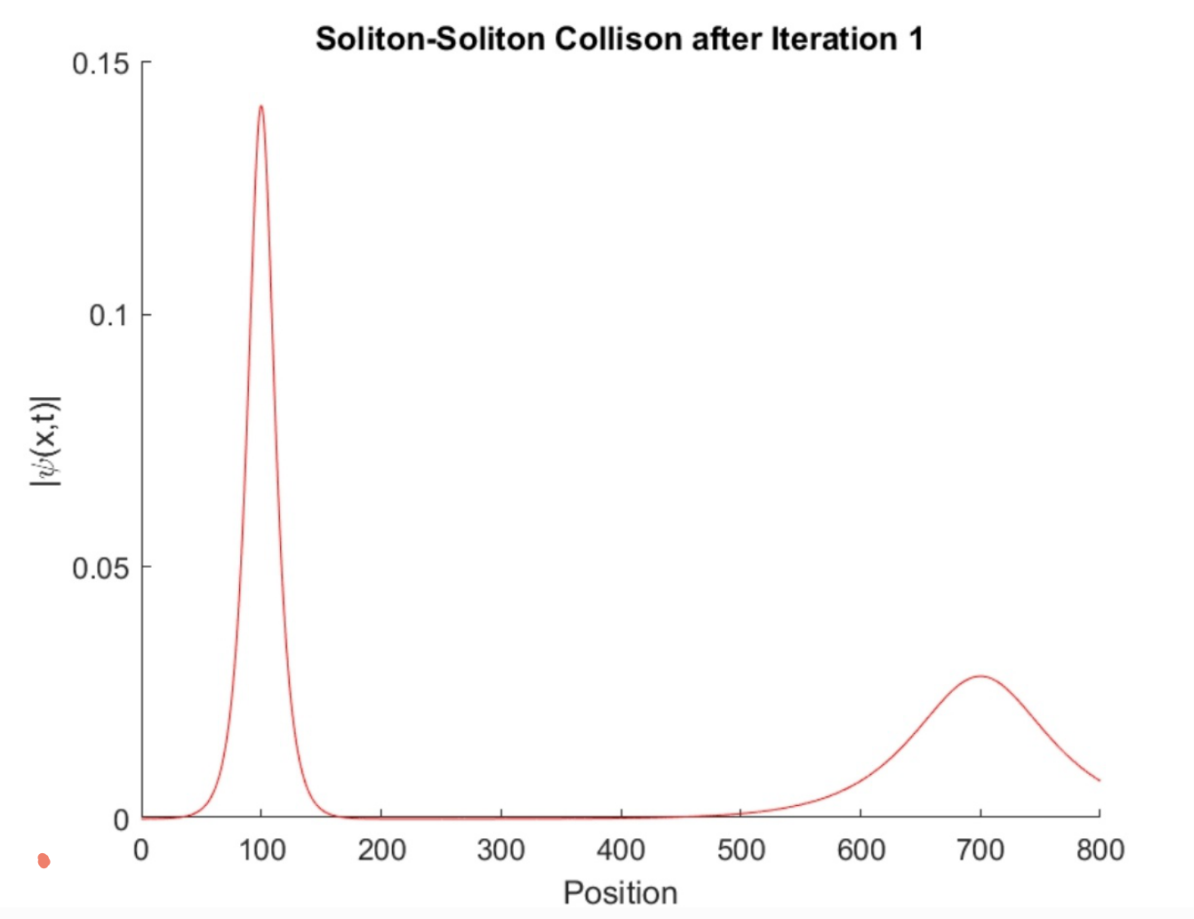
Following the Equation (4) we simulate the collison of soliton in the MATLAB.

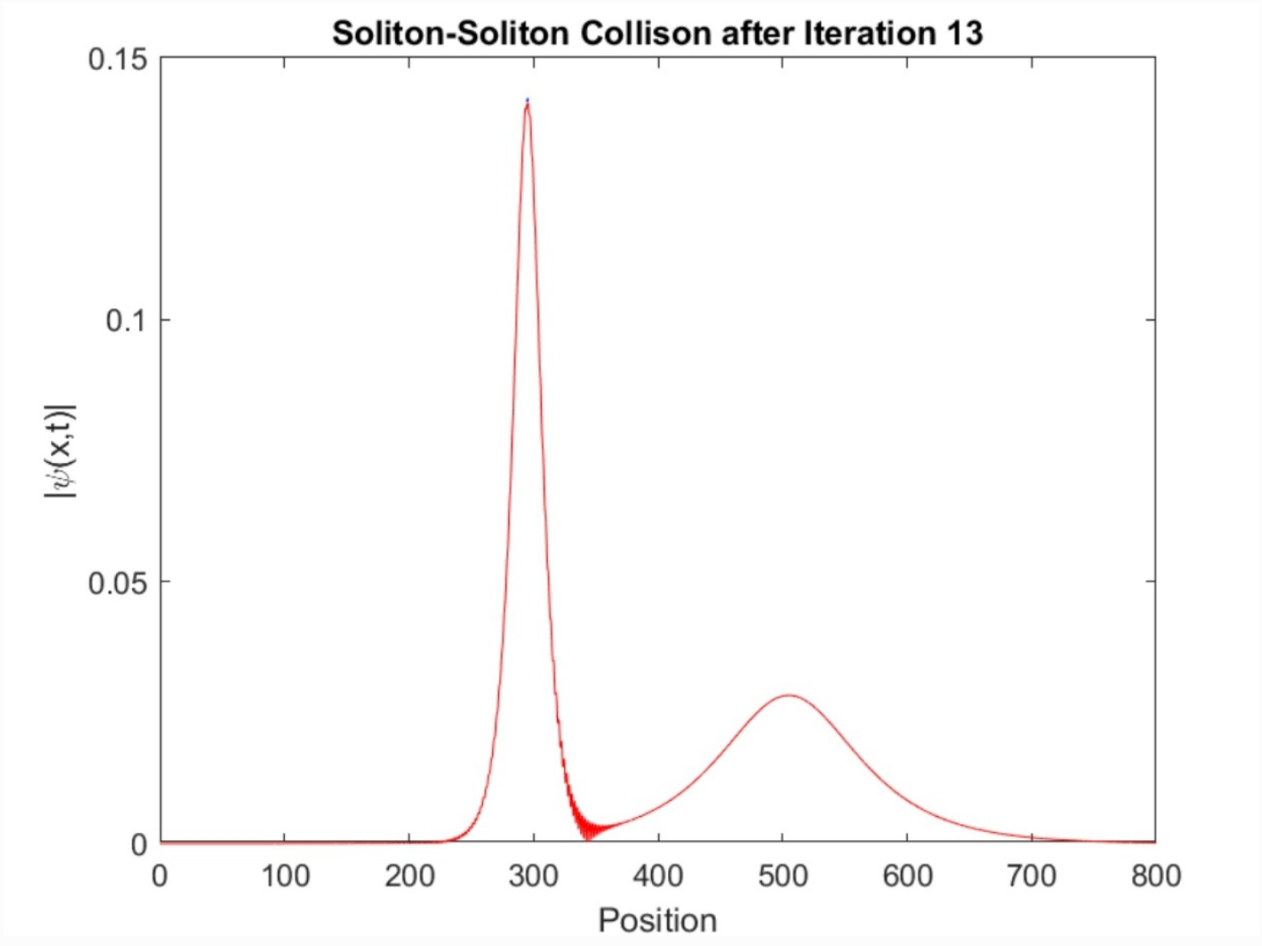
```
num_collisions = 20; % Number of collisions to simulate
N = 6000;          % Number of spatial nodes
L = 800;           % Length of spatial domain
dx = L / N;        % Spatial step size
x = 0:dx:L;        % Spatial grid
% Initial parameters for the first soliton
a = 0.1;           % Amplitude parameter
b = 3;             % Frequency parameter
x0 = 100;          % Initial position of the first soliton
gif_frames = cell(1, num_collisions);
% Initial parameters for the second soliton
a1 = 0.2 * a;      % Amplitude parameter for the second soliton
b1 = -3;           % Frequency parameter for the second soliton
x1 = 700;          % Initial position of the second soliton

for i = 1:num_collisions
    % Calculate the wave function for each iteration
    psi = a * sqrt(2) * exp(1i * b * x / 2) .* sech(a * (x - x0)) + a1 *
sqrt(2) * exp(1i * b1 * x / 2) .* sech(a1 * (x - x1));

    % For demonstration, let's assume a shift towards each other in each
iteration
    shift_amount = 5 * i; % Increase the shift amount towards each other

    % Shift the positions towards each other
    x0 = x0 + shift_amount / 2; % Shift the first soliton leftwards
    x1 = x1 - shift_amount / 2; % Shift the second soliton rightwards

    % Plot the absolute value of psi after each iteration
    psi_abs = abs(psi); % Take the absolute value of psi

    % Create masks based on amplitude for two solitons
    mask1 = psi_abs >= a * sqrt(2); % Mask for soliton 1
    mask2 = psi_abs < a * sqrt(2);  % Mask for soliton 2
```

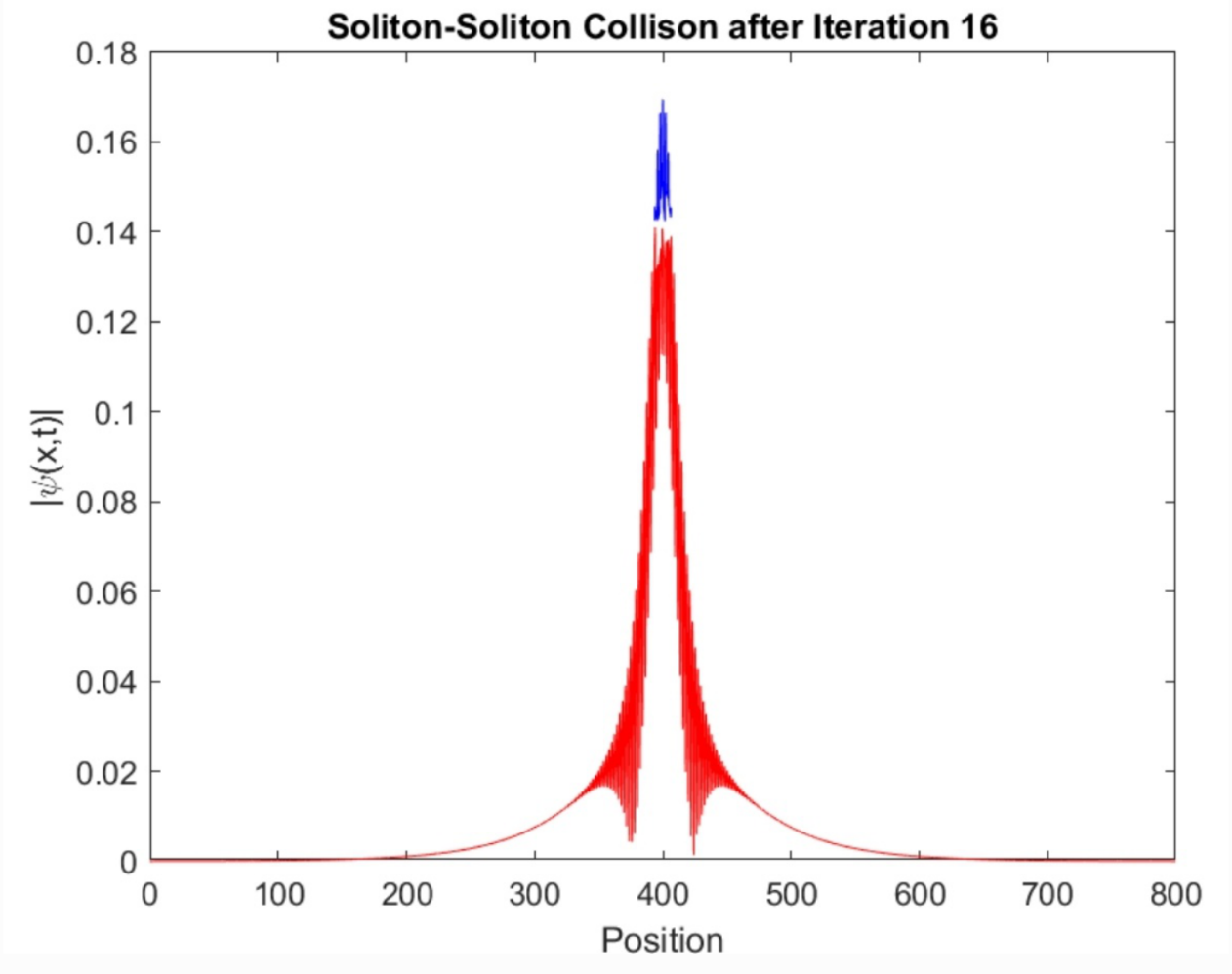
```
    % Plot the absolute value of psi after each iteration

    figure(); % Create an invisible figure
    plot(x(mask1), psi_abs(mask1), 'b'); % Plot soliton 1 with blue color
    hold on;
    plot(x(mask2), psi_abs(mask2), 'r'); % Plot soliton 2 with red color
    hold on;

    xlabel('Position');
    ylabel('|\psi(x,t)|');
    title(['Soliton-Soliton Collison after Iteration ', num2str(i)]);

    % Save the plot as an image within the loop
    filename = ['soliton_collision_' num2str(i) '.png']; % Define the
filename
    saveas(gcf, filename); % Save the current figure

    % Capture the frame for the GIF
    gif_frames{i} = getframe(gcf);
    close(gcf); % Close the figure after capturing the frame
end
```

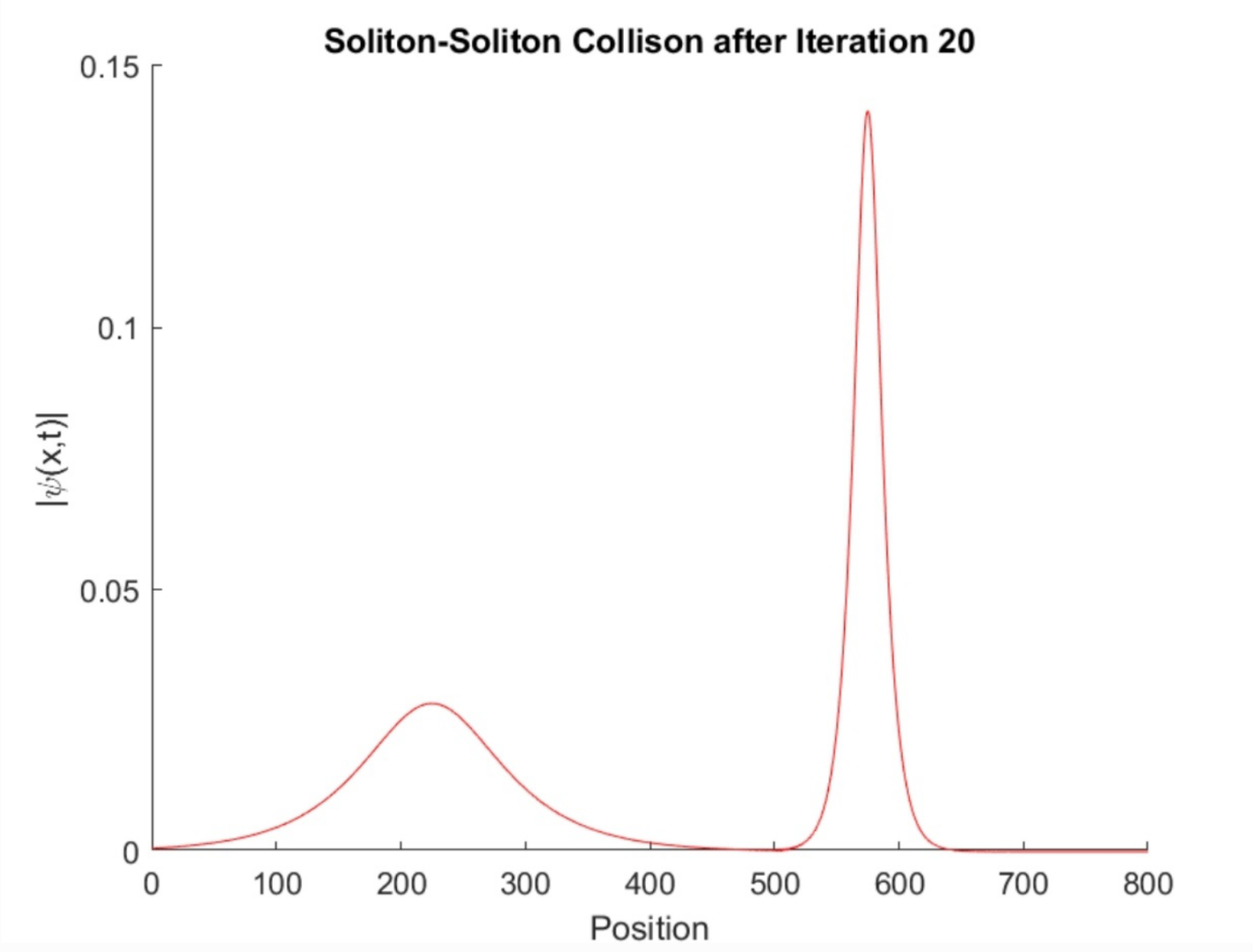
The output of the simulations are shown below. For visualization purpose we consider 20-different collisons between the 2-solitons.

Here, iteration refers to the number of collison.




Soliton-Soliton Collison after Iteration 13
0.15
0.1
0.05
0
|ψ(x,t)|
0
100
200
300
400
500
600
700
800
Position


Soliton-Soliton Collison after Iteration 16
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
|ψ(x,t)|
0
100
200
300
400
500
600
700
800
Position



We observe from the simulation that post-collison the position of the soliton shifted as they were moving towards each other. Initialy the position of peak for the larger and smaller amplitude soliton were at $x = 100$ and $x = 700$ location After the collison we observe the position of the peak for smaller amplitude soliton is at around $x = 250$ and

the peak of the larger amplitude soliton is at around $x = 550$.

For better visualization of the collison, the generated .gif image will be attached as a solution.

Problem (c)

NLS has an infinite number of conservation integrals. Numerically devise a scheme to evaluate the integrals (as a function of time)

normalization: $S_0(t) = \int_0^L |\psi(x,t)|^2 dx$

energy: $S_2(t) = \int_0^L \left[2 \left| \frac{\partial \psi(x,t)}{\partial x} \right|^2 - \frac{1}{2} |\psi(x,t)|^4 \right]$

First verify analytically that $S_0(t)$ and $S_2(t)$ are indeed conserved. Plot the numerical solution to these integrals. Comment on the numerical accuracy.

Approach to the solution:

To show that, that $S_1(t)$ and $S_2(t)$ is conserved with respect to time, first we compute $S_0(t)$ and taking the derivate, the results should be zero which will indicate that both $S_0(t)$ and $S_2(t)$ are indeed conserved.

To solve the equations, we compute the simulation in Python 3.8 version.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 11 16:11:36 2023

@author: Walia Farzana
"""

import numpy as np
from scipy.integrate import quad
import matplotlib.pyplot as plt
from sympy import symbols, exp, sech, diff, integrate, conjugate, sqrt

# Define the symbols
x, t, a, b, x0, a1, b1, L = symbols('x t a b x0 a1 b1 L')

#function for sech, as python does not have built-in sech function
def sech(x):
    return 2 / (np.exp(x) + np.exp(-x))

#function to compute the psi(x,t=0) as per the reference paper (Dr.
Vahala's Paper,equation(24))
def psi(x, t, a, a1, x0, x1, b, b1):
    term1 = a*np.sqrt(2) * np.exp(1j * (b * x)/2) * sech(a * (x - x0))
    term2 = a1*np.sqrt(2) * np.exp(1j * (b1 * x)/2) * sech(a1 * (x - x1))
    return term1 + term2

# the function to get the absolute of psi (x,t=0)
def integrand(x, t, a, a1, x0, x1, b, b1):
    psi_value = psi(x, t, a, a1, x0, x1, b, b1)
    return np.abs(psi_value)**2

## function to compute the definite integral from 0 to L
def calculate_S0(t, a, a1, x0, x1, b, b1, L):
    result, _ = quad(integrand, 0, L, args=(t, a, a1, x0, x1, b, b1))
    return result
```

```
# Set your values for a, a1, x0, x1, b, b1, L
a = 1
a1 = 2 # a1=2*a
x0 = 400
x1 = 700
b = 3
b1 = -3 #b=-b1# Plotting the numerical solution for different L values

# L = 6000


# # Calculate S0 for a specific time t
# t = 0 # As per the initial condition equation
# S0_t = calculate_S0(t, a, a1, x0, x1, b, b1, L)
# print(f"S0({t}) = {S0_t}")


# Calculate S0 for a specific time t and various L values
#t = 0
L_values = [100, 500, 600, 900, 1000,5000,6000,7000,8000]  # Different
values of L

S0_values = [] # empty list to append different value of S_0 integration
for different values of L
for L in L_values:
    S0_t = calculate_S0(t, a, a1, x0, x1, b, b1, L)
    print(f"S0({t}) for L={L}: {S0_t}")
    # Compute the time derivative of S0(t)
    dS0_dt = diff(S0_t, t)
    print('The result of the Derivation of S0 with respect to
time:',dS0_dt)
```

differentiation with time t

Results for S_0 :

```
S0(t) for L=100: 1.0601650967723544e-260
The result of the Derivation of S0 with respect to time: 0
S0(t) for L=500: 3.999999999999999
The result of the Derivation of S0 with respect to time: 0
S0(t) for L=600: 4.000000000000006
The result of the Derivation of S0 with respect to time: 0
S0(t) for L=900: 4.0000000000018305
The result of the Derivation of S0 with respect to time: 0
S0(t) for L=1000: 3.999999999999999
The result of the Derivation of S0 with respect to time: 0
S0(t) for L=5000: 1.320127640176662e-28
The result of the Derivation of S0 with respect to time: 0
S0(t) for L=6000: 1.5192168018089117e-59
The result of the Derivation of S0 with respect to time: 0
S0(t) for L=7000: 6.972563524564891e-12
The result of the Derivation of S0 with respect to time: 0
S0(t) for L=8000: 7.082171068886038e-31
The result of the Derivation of S0 with respect to time: 0
```

Code and Results for $S_2(t)$:

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 11 16:16:22 2023

@author: Walia Farzana
"""

import numpy as np
from scipy.integrate import quad
import matplotlib.pyplot as plt
from sympy import symbols, exp, sech, diff, integrate, conjugate, sqrt

# Define the symbols
x, t, a, b, x0, a1, b1, L = symbols('x t a b x0 a1 b1 L')
# Define the function for sech
def sech(x):
    return 2 / (np.exp(x) + np.exp(-x))

# Define the psi function
def psi(x, t, a, a1, x0, x1, b, b1):
    term1 = a * np.sqrt(2) * np.exp(1j * (b * x) / 2) * sech(a * (x - x0))
    term2 = a1 * np.sqrt(2) * np.exp(1j * (b1 * x) / 2) * sech(a1 * (x -
x1))
    return term1 + term2

# Calculate x values
x_values = np.linspace(-100, 100, 10000)  # Define the range for x values

# Define the parameters
a = 1
a1 = 2  # a1=2*a
x0 = 400
x1 = 700
b = 3
b1 = -3  # b=-b1
# Calculate psi values for the x values
psi_values = psi(x_values, 0, 1, 2, 400, 700, 3, -3)
```

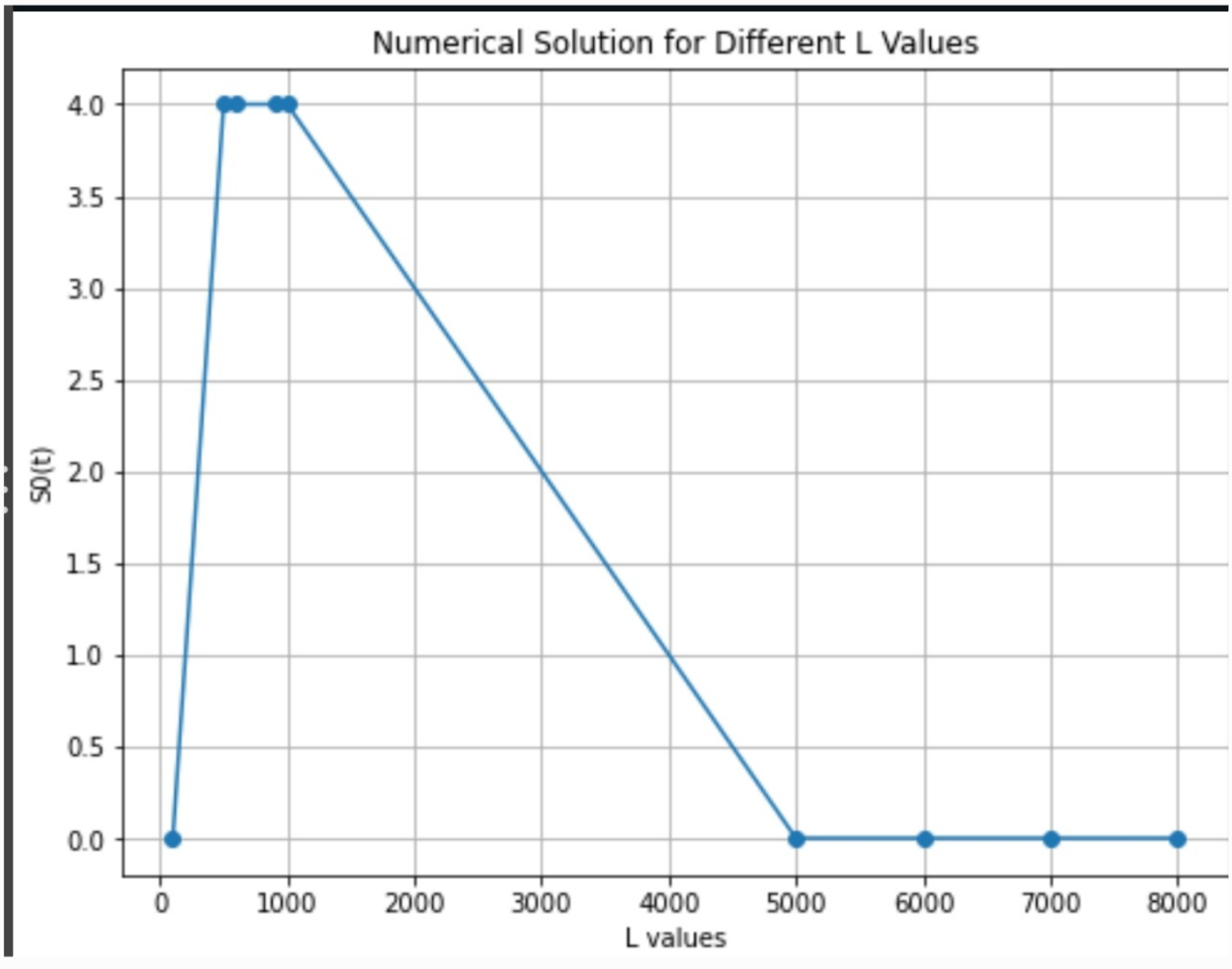
```
# Compute the gradient of psi with respect to x
gradient_psi = np.gradient(psi_values, x_values)

gradient_real_value=np.real(gradient_psi)
gradient_mean_value=np.abs(np.mean(gradient_real_value))**2
# Use the gradients as needed


# the function to get the absolute of psi (x,t=0)
def integrand(x, t, a, a1, x0, x1, b, b1):
    psi_value = psi(x, t, a, a1, x0, x1, b, b1)
    return (2*gradient_mean_value-0.5* np.abs(psi_value)**4)

## function to compute the definite integral from 0 to L
def calculate_S2(t, a, a1, x0, x1, b, b1, L):
    result, _ = quad(integrand, 0, L, args=(t, a, a1, x0, x1, b, b1))
    return result


# Calculate S0 for a specific time t and various L values
#t = 0
L_values = [100, 500, 600, 900, 1000,5000,6000,7000,8000]  # Different
values of L

S2_values = [] # empty list to append different value of S_0 integration
for different values of L
for L in L_values:
    S2_t = calculate_S2(t, a, a1, x0, x1, b, b1, L)
    S2_values.append(S2_t)
    print(f"S2({t}) for L={L}: {S2_t}")
    dS0_dt = diff(S2_t, t)
    print('The result of the Derivative of S2 with respect to
time:',dS0_dt)
    #print(dS0_dt)
```

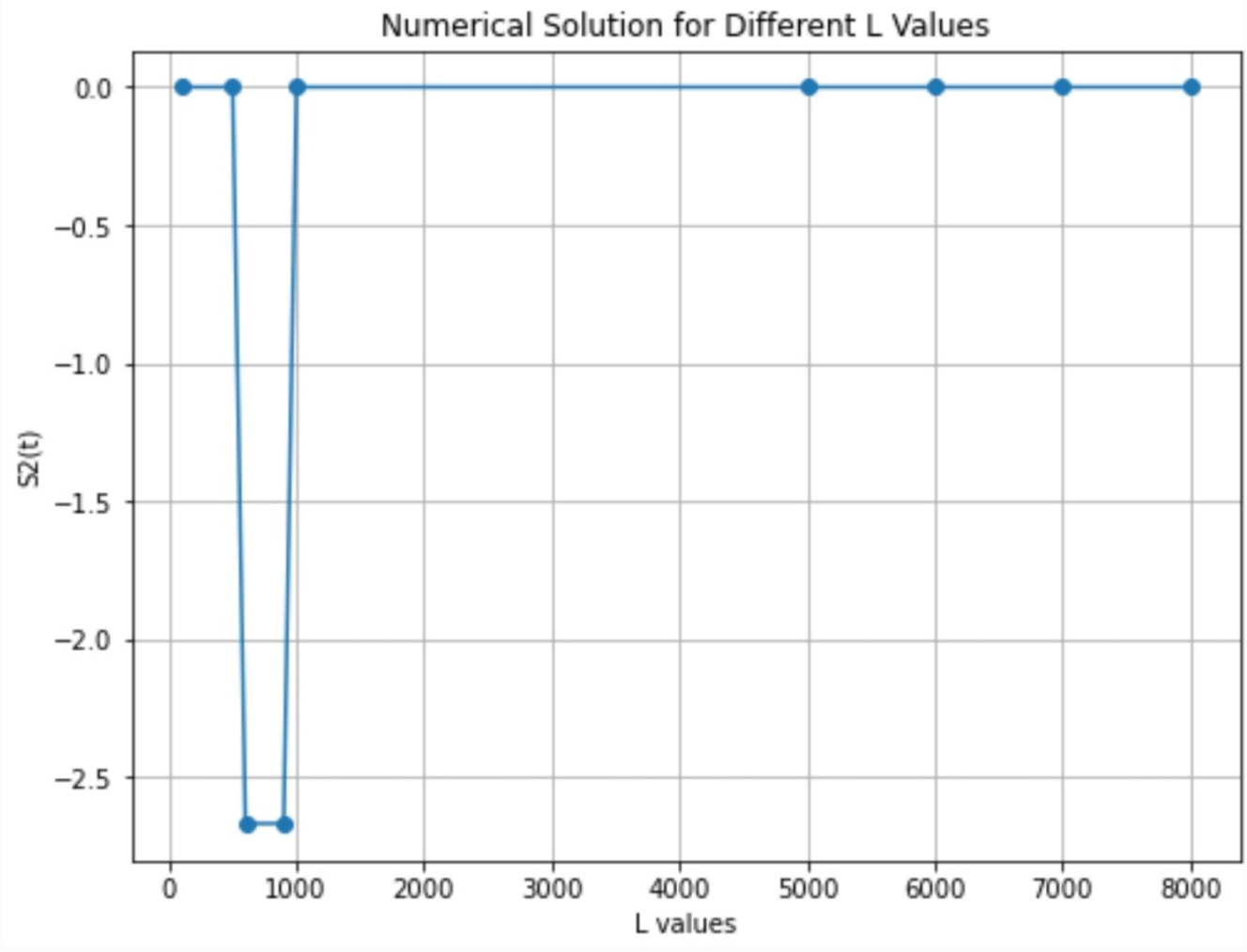
differentiation with time t

Results for S_2:

```
S2(t) for L=100: 5.446622631557685e-263
The result of the Derivative of S2 with respect to time: 0
S2(t) for L=500: -4.674706371436275e-09
The result of the Derivative of S2 with respect to time: 0
S2(t) for L=600: -2.6666666666666816
The result of the Derivative of S2 with respect to time: 0
S2(t) for L=900: -2.666666666667096
The result of the Derivative of S2 with respect to time: 0
S2(t) for L=1000: -6.612380094716647e-14
The result of the Derivative of S2 with respect to time: 0
S2(t) for L=5000: -9.436588130171068e-59
The result of the Derivative of S2 with respect to time: 0
S2(t) for L=6000: -8.261327614864114e-121
The result of the Derivative of S2 with respect to time: 0
S2(t) for L=7000: -1.4915867240265352e-25
The result of the Derivative of S2 with respect to time: 0
S2(t) for L=8000: -1.3464940190946155e-63
The result of the Derivative of S2 with respect to time: 0
```

Using the above codes we observe the values of $S_0(t)$ and $S_2(t)$ for different range of L values and plotted the following graph which indicates the the values converge to zero as the L value increases.






```
S0(0) for L=100: 1.0601650967723544e-260
S0(0) for L=500: 3.999999999999999
S0(0) for L=600: 4.000000000000006
S0(0) for L=900: 4.0000000000018305
S0(0) for L=1000: 3.999999999999999
S0(0) for L=5000: 1.3201276401766 62e-28
S0(0) for L=6000: 1.5192168018089117e-59
S0(0) for L=7000: 6.972563524564891e-12
S0(0) for L=8000: 7.082171068886038e-31
```

However, observing the values of the simulation we see that the values are tensed to zero but no exactly

2000, with the increment of L values.

In summary, while exploring the literature, we learn there is a method called split step fourier method to solve the non-linear equations but this is out of the scope for this particular final project. Basically, we followed the analytical solution found in the literature to compute numerical solution of soliton-1 solution, 2 solitons and integration of $S_0(t)$ and $S_2(t)$.

References:

1. Paper: Quantum lattice gas representation of some classical solitons by George Vahala, Jeffrey Yepez, Linda Vahala

2. An introduction to the Split Step Fourier Method using MATLB by Pablo. V. Suárez.

3. Videos for Better Understanding of NLS:

https://www.youtube.com/watch?v=WcNiA06WNvI&list=PLTjLwQcqQzNKzSAxJxKpmOtAriFS5wWy4&index=1&t=0s