SH_exercise

June 30, 2023

1 Pseudo-spectral method to solve the Swift-Hohenberg equation

In this exercise we want to solve the Swift-Hohenberg (SH) equation numerically

$$\partial_t u = \left[\epsilon - (q_0^2 + \partial_x^2)^2\right]\ u - u^3 \tag{1}$$

For this we want to use the method of operator splitting. The idea behind operator splitting is to divide the right hand side into two operators, one linear operator containing the highest-order spatial derivatives $\mathcal{L}$ and one nonlinear operator containing the remaining terms $\mathcal{N}$. The linear part and the nonlinear part can then be solved independently, using different numerical approaches.

Consider the linear part

$$\mathcal{L}u = \left[\epsilon - (q_0^2 + \partial_x^2)^2\right]\ u, \tag{2}$$

and the nonlinear part

$$\mathcal{N}(u) = -u^3. \tag{3}$$

To hand in your solution, you may edit this jupyter notebook and print a pdf via `File -> Download as -> PDF via LaTeX`. Make sure that all parts of your code are readable (if not introduce linebreaks \). Don't forget to run all cells to include all plots. Also please create new Markdown cells for written answers. Cells are added via `Insert` and they are changed to Markdown via `Cell -> Cell Type -> Markdown`.

1.0.1 (a) Solving the linear part (3 points)

Before we can apply the full algorithm, we want to implement a few functions in preparation. Our simulation is supposed to solve the SH equaation starting from random initial data.

1. Implement the function `create_random_array`, which should return an array of size *number* (or in 2D of size $(number, number)$) of independent random values drawn uniformly from $[-amplitude, amplitude)$.

We will solve the linear part via transformation into fourier space. We consider the function u on an interval of length L at the discretization points $x_n = nL/N$, $n = 0, ..., N-1$.

The Fourier transform of the function u, which is known at discrete points, can be approximated as

$$\hat{u}(q_n) = \frac{1}{\sqrt{2\pi}} \sum_{k=0}^{N} \mathrm{e}^{-iq_n x_k} \; u(x_k) \frac{1}{L}, \tag{4}$$

with wave numbers $q_n = 2\pi nL/N$, where the wave numbers with $n = 0, ..., N$ are sufficient to reconstruct the original u, because we cannot resolve more modes. The inverse transform is then given via

$$u(x_k) = \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{N} \mathrm{e}^{iq_n x_k} \; \hat{u}(q_n) \frac{2\pi L}{N}. \tag{5}$$

We will now use the Discrete Fourier Transform (DFT) to approximate the Fourier transform of our discretized function and the inverse DFT to reconstruct a signal from Fourier space. In `scipy` Fast Fourier Transform methods are implemented to calculate this numerically. A tutorial on the implementation is given in: https://docs.scipy.org/doc/scipy/reference/tutorial/fft.html.

2. Read the tutorial on FFT from the SciPy documentation. How is the DFT defined that is used in SciPy and why doesn't it make any difference?
   Implement the function `create_wavenumber_array`, which should return an array of the wave numbers q_n in the correct ordering used by SciPy. *Hint: Look at* ***`fftfreq`****.*

Now we want to solve the linear part of the SH equation

$$\partial_t u = \mathcal{L}u = \left[\epsilon - (q_0^2 + \partial_x^2)^2\right] \; u. \tag{6}$$

For this we use a Fourier transform of the spatial domain on the linear equation and solve the resulting differential equation. One can see that advancing the Fourier transform by a timestep Δt in Fourier space can be written as an operator, which acts on the Fourier representation of $u(t)$.

3. Show that in Fourier space, for the linear part of the SH equation the time evolution from initial condition $\hat{u}(t, q_n)$ for a timestep Δt and wavenumber q_n is given by

$$\hat{u}(t + \Delta t, q_n) = \exp\left(\left[\epsilon - (q_0^2 - q_n^2)^2\right] \cdot \Delta t\right) \cdot \hat{u}(t, q_n). \tag{7}$$

   Implement this operator in `create_time_evolution_operator`, which should return the operator as an array. (Note: Remember the ordering of the wavenumbers!)

Extra points: Implement the 2D case.

Solution 2. Answer 2. here.

Solution 3. Answer 3. here.

[ ]:
```
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits import mplot3d
from matplotlib import cm
from scipy.fft import fft, ifft, fftn, ifftn
```

[ ]:
```
N = 64*2 # Num of discretisation points

L=10.0*2.0*np.pi
dt=0.005
q02=1.0 # q0^2 of SH model, should be kept 1
NUM_STEPS = 100001
epsilon=0.1 # epsilon of SH model
alpha=0.0 # alpha for the quadratic term in SH model

dx=L/N
```

[ ]:
```
def create_random_array(number=N, amplitude=0.01, two_dim=False):
    '''
    This function should create an array of size N, which is initialized with␣
 ↪random numbers drawn
    from the uniform distribution in [-amplitude, amplitude).

    Args:
        number (int): number of elements in random array
        amplitude (float): amplitude of random array
    Return: random array
    '''
    ################################
    # HERE IS SPACE FOR YOUR CODE #
    ################################


def create_wavenumber_array(num_modes=N):
    '''
    This function should create an array of size N, which is initialized with␣
 ↪the wavenumbers.

    Args:
        number (int): number of wavenumbers
    Return: array
    '''
    ################################
    # HERE IS SPACE FOR YOUR CODE #
    ################################

def create_time_evolution_operator(wavenums, zero_wavenum=q02, eps=epsilon,␣
 ↪two_dim=False):
    '''
    This function should create the time evolution operator in fourier space,␣
 ↪which is of size
    (num_nodes) in 1D and (num_nodes,num_nodes) in 2D.
```

```
    If you want you can program the 2D case and use the flag two_dim to␣
↪differentiate between the cases.

    Args:
        wavenums (array): wavenumbers in one dimension for the evolution␣
↪operator, ordered correctly for SciPy FFT
        zero_wavenum (float): q_0^2 term in SH equation - standard value was␣
↪defined above
        eps (float): epsilon in SH equation - standard value was defined above
        two_dim (bool): Wether to return the 2D operator
    Return: array containing the time evolution operator
    '''
    ################################
    # HERE IS SPACE FOR YOUR CODE #
    ################################
```

1.0.2 (b) Implementing the operator-split Pseudo-spectral method (2 points)

Now that we have implemented an operator to solve the linear part, we can look at the nonlinear part

$$\partial_t u = \mathcal{N}(u) = -u^3, \quad \text{initial data:} \quad u(t). \tag{8}$$

This we can integrate using Euler stepping (i.e. linear extrapolation). The equation to approximate the solution of the nonlinear equation at $t + \Delta t$, i.e. $\tilde{u}_t$ is then

$$\tilde{u}_t = u(t) - u(t)^3 \Delta t. \tag{9}$$

Now implement the operator splitting method, as described on the sheet and above. For this implement `perform_simulation`. You should generate random initial data using `create_random_array`, generate the wavenumbers in the correct ordering with `create_wavenumber_array`, use these to create the time evolution operator in fourier space with `create_time_evolution_operator` and then integrate the SH equation with the algorithm, saving each step into an array/list, which you output for plotting. To solve the linear system, you have to fourier transform the output of the nonlinear equation, apply the time evolution operator and then transform back into real space.

Extra points: Implement the 2D case with the quadratic nonlinear term $-\alpha u^2$.

[ ]:
```
def perform_simulation(num_steps=NUM_STEPS, timestep=dt, two_dim=False,␣
↪num_modes=N, \
                       quadratic=False, alpha_quad=alpha, eps=epsilon):
    '''
    This function solves the Swift Hohenberg equation with operator splitting␣
↪and a pseudospectral method.
    It should create noisy initial data with the function␣
↪"create_random_array", the wavenumbers to be considered
```

```
    with "create_wavenumber_array" and the time evolution operator with␣
↪"create_time_evolution_operator".
    Then it should integrate the SH equation with the pseudospectral method,␣
↪using fft & ifft (1D) or
    fftn & ifftn (2D).

    Args:
        num_steps (int): Number of timesteps to integrate - standard value was␣
↪defined above
        timestep (float): Size of one timestep to integrate - standard value␣
↪was defined above
        two_dim (bool): Wether to return the 2D operator
        num_modes (int): number of modes - standard value was defined above
        quadratic (bool): Wether to include the quadratic term
        alpha_quad (float): alpha coefficient of the quadratic term
        eps (float): epsilon in SH equation
    Return: array, array containing the solutions and the respective times
    '''
    ################################
    # HERE IS SPACE FOR YOUR CODE #
    ################################
```

1.0.3 (c) Plot your results (2 points)

Plot the results of a simulation with $\varepsilon = 0.1$ as a 3D surface plot (u as a function of x and t). You can use `plt.plot_surface` for this. (To get a nicer perspective you should look at `view_init`.) Additionally, plot the simulation results u as a function of x for a few interesting times.

Extra points: Plot the 2D case with and without the quadratic nonlinear term $-\alpha u^2$ for a few times t. You may use `plt.pcolormesh` for this. It makes sense to use N=64 here for faster computations.

[ ]:
```
    ################################
    # HERE IS SPACE FOR YOUR CODE #
    ################################
```

1.0.4 (d) Comparison to the amplitude equation (1 point)

In the lecture we derived an amplitude equation for the SH equation. Plotting the result in 1D, you should notice that the pattern has grown over a period of time and then remained constant. We now want to check our simulation against the analytical result.

1. For $\epsilon = 0.1$ plot the squared amplitude A^2 of the pattern you found as a function of the simulation time to check whether your simulation has converged.
2. Calculate the squared amplitde A^2 for $\epsilon = 0.05,\ 0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5$. What is the theoretical expectation for A^2 as a function of ϵ? Plot the expectation and the simulation results for $A^2(\epsilon)$.

[ ]:
```
################################
# HERE IS SPACE FOR YOUR CODE #
################################
```