SoftwareX 15 (2021) 100741

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

pyGLLE: A Python toolkit for solving the generalized Lugiato–Lefever equation

Oliver Melchert *, Ayhan Demircan

Institute of Quantum Optics (IQO), Leibniz Universität Hannover, 30167 Hannover, Germany
Cluster of Excellence PhoenixD (Photonics, Optics, and Engineering - Innovation Across Disciplines), Hannover, Germany

ARTICLE INFO

Article history:
Received 9 March 2020
Accepted 9 June 2021

Keywords:
Nonlinear partial differential equations
Lugiato–Lefever equation
Dissipative solitons
Python

ABSTRACT

We present a Python toolkit for simulating the propagation dynamics of dissipative solitons in a variant of the Lugiato–Lefever equation (LLE) including dispersion terms of third and fourth order. In addition, the provided software allows to prepare initial conditions given by stationary localized solutions of the standard LLE in the anomalous group-velocity dispersion regime. Propagation scenarios for custom control parameters and initial conditions can be specified by the user via a simple class data structure. We demonstrate the implemented functionality by showing how to obtain stationary solutions of the standard LLE containing a dissipative soliton, and, demonstrating different characteristic propagation scenarios. The `pyGLLE` software package is open-source and released under the X11 License in a publicly available software repository.

© 2021 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Code metadata

Current code version	1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2020_88
Legal Code License	MIT License
Code versioning system used	none
Software code languages, tools, and services used	Python, GitHub
Compilation requirements, operating environments & dependencies	The `pyGLLE` package requires Python, numpy and scipy. The provided examples need Pythons `matplotlib` for figure generation.
If available Link to developer documentation/manual	Documentation provided within code
Support email for questions	melchert@iqo.uni-hannover.de

1. Introduction

The Lugiato–Lefever equation (LLE) represents a paradigmatic non-conservative model system relevant to nonlinear optics. Originally introduced to study pattern formation in nonlinearly responding, pumped lossy optical resonators [1], it constitutes a variant of the nonlinear Schrödinger equation (NSE) with added damping and driving. In the conservative NSE, solitary-wave solutions, i.e. solitons, are of special significance. They describe localized field pulses that exist due to a balance of dispersive and nonlinear effects and propagate without changing their shape [2–4]. In case of the LLE a similarly important role is played by dissipative solitons [1,5,6]. These are localized dissipative structures (LDSs) embedded in a homogeneous background field, existing due to a balance between dispersion, nonlinear effects, loss and external driving. Albeit both models are closely related, the LLE does not satisfy any of the NSEs conservation laws and no explicit dissipative soliton solutions are known.

Systems described by the LLE and its generalized variants allow for a detailed investigation of the dynamics of single LDSs as well as stable agglomerates of LDSs [7–9], resembling molecule-like states studied in the conservative system [10]. In particular, for modeling octave-spanning frequency combs for resonators with Kerr nonlinearity it is important to also take into account higher orders of dispersion [11–14].

2. Scientific problem solved by the software

The scientific problem solved by the software is the propagation of a field $A \equiv A(x,t)$ in time t, subject to the generalized

* Corresponding author at: Institute of Quantum Optics (IQO), Leibniz Universität Hannover, 30167 Hannover, Germany.
E-mail addresses: melchert@iqo.uni-hannover.de (Oliver Melchert), demircan@iqo.uni-hannover.de (Ayhan Demircan).

https://doi.org/10.1016/j.softx.2021.100741
2352-7110/© 2021 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Lugiato–Lefever equation (gLLE)

$$\partial_t A = P - (1 - i\theta)\, A - \left(i d_2 \partial_x^2 - d_3 \partial_x^3 - i d_4 \partial_x^4\right) A + i|A|^2 A \qquad (1)$$

for $t \geq 0$ on a coordinate interval $-X \leq x \leq X$, satisfying the boundary condition $A(X,t) = A(-X,t)$ and starting from an initial condition $A_0(x) = A(x, t=0)$. The evolution of the field, described by Eq. (1), is determined by the frequency detuning θ, homogeneous scalar driving field amplitude P, and second, third and fourth order dispersion parameters d_2, d_3 and d_4, respectively. The standard LLE, i.e. a variant of the NSE with added damping and driving, is specified by $(d_2, d_3, d_4) = (-1, 0, 0)$.

Subsequently, we present the main features of our numerical implementation of the generalized Lugiato–Lefever equation in the form of Eq. (1), including the functionality to solve for stationary solutions of the standard LLE, containing one or several localized dissipative structures. The provided Python package pyGLLE derives from our research software, developed with the aim of being simple to use, extendible and reusable. For example, it can be easily amended by linear and nonlinear terms of higher order and it is straightforward to realize models based on a non-homogeneous driving field or coupled equations of the form of Eq. (1).

3. Software description

pyGLLE is written using the Python programming language [15] and depends on the functionality of numpy and scipy [16].

3.1. Source code structure

The provided software exhibits the following modular structure [17]:

```
pyGLLE/
  src/
    data_handler.py
    stationary_solution.py
    solver.py
  scripts/
    pyGLLE.py
  numExp01_stationarySolution/
    main_findStationarySolution.py
    run.sh
    pp_figure/
  numExp02_propagationScenarios/
    main_findStationarySolution.py
    main_propagateInitialCondition.py
    run.sh
    pp_figure_propagationScenarios/
```

The folder src/ contains modules that implement the basic functionality of the software:

data_handler.py provides a class handling data accumulation and output. Output data is stored using the numpy native npz-format, see Section 3.3.1.

stationary_solution.py provides functions allowing to obtain a homogeneous stationary solution and also a stationary solution, possibly including one or several localized dissipative structures, for given control parameters θ, P, and fixed $(d_2, d_3, d_4) = (-1, 0, 0)$.

solver.py implements a solver for the numerical integration of Eq. (1) using the explicit high-order Runge–Kutta method "DOP853" [18]. This can be changed by re-setting the integrator type in function solver.

Folder scripts/ contains the main Python module implementing the interface between the user supplied code, see Section 3.2, and the algorithms and data structures contained in the modules in folder src/:

pyGLLE.py defines the main functions findStationarySolution, see Section 3.3.2, and propagateInitialCondition, see Section 3.3.1. These functions implement the main functionality of the provided software.

The folders numExp01_stationarySolution/ and numExp02_propagationScenarios/ contain scripts that implement the examples detailed in Sections 4.1 and 4.2 , respectively. For a more detailed description of functions, defined in the above modules, their parameters and return values we refer to the example cases and documentation provided within the code [19].

3.2. Specification of a propagation scenario

A simple Python class forms the interface between the user defined code and pyGLLE. It allows to define system parameters for setting up the computational domain, specify control parameters and implement initial conditions. The class, here referred to as SIM_SETUP, detailed in listing 1, needs to declare and specify the subsequent attributes and methods.

Listing 1: Python class used to specify a propagation scenario.

```
class SIM_SETUP:
    # -- COMPUTATIONAL DOMAIN -------
    Nx      = "!!--REQUIRES INT---!!"
    xMax    = "!!--REQUIRES FLOAT-!!"
    Nt      = "!!--REQUIRES INT---!!"
    tMax    = "!!--REQUIRES FLOAT-!!"
    nSkip   = "!!--REQUIRES INT---!!"
    # -- MODEL PARAMETERS -------=---
    d2      = "!!--REQUIRES FLOAT-!!"
    d3      = "!!--REQUIRES FLOAT-!!"
    d4      = "!!--REQUIRES FLOAT-!!"
    theta   = "!!--REQUIRES FLOAT-!!"
    P       = "!!--REQUIRES FLOAT-!!"
    # -- OUTPUT DATA ----------------
    fName   = "!!--REQUIRES STR---!!"

    def initial_field(self, x):
        raise NotImplementedError
```

User-defined attributes. The below attributes are listed using the format "name (data-type): description".

- Nx (int): Number of mesh points used for discretizing the x-coordinate. For best performance it should be set to an integer multiple of 2.
- xMax (float): Bound for the x-coordinate. Internally, the discrete x-mesh extends from the minimal value -xMax to the (exclusive) maximal value xMax in equidistant steps of width dx=2*xMax/(Nx+1).
- Nt (int): Number of mesh points used for discretizing the t-coordinate.
- tMax (float): Value of the largest mesh point along the propagation coordinate t. The discrete t-mesh extends from 0 to tMax in equidistant steps of width dt=tMax/Nt.
- nSkip (int): During a simulation run, every nSkip-th field configuration is stored, only.
- theta (float): frequency detuning.
- P (float): homogeneous driving field amplitude.
- d2 (float): second order dispersion parameter.
- d3 (float): third order dispersion parameter.
- d4 (float): fourth order dispersion parameter.
- fName (str): Output file name for the data generated during a simulation run.

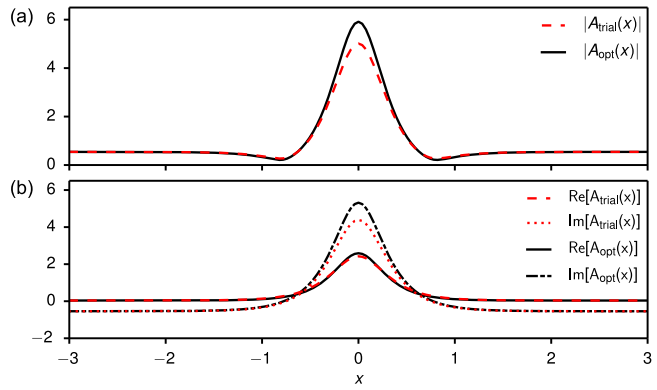

Fig. 1. Comparison of a user-defined trial solution A_{trial} and an optimized stationary solution A_{opt} for the standard LLE with parameters $(\theta, P) = (15, 8)$. (a) comparison of the field amplitudes, and, (b) comparison of the respective individual real and imaginary parts.

User-defined methods. The below method is listed using the format "`method-name(arg1,arg2)`: description".

- `initial_field(self, x)`: As argument, the function provides a `numpy` array `x` of length `Nx`, representing the discrete x-mesh. The function should return array data specifying an initial condition $A_0(x)$.

3.3. Software functionalities

Importing the main module `pyGLLE.py` allows to access the functions `findStationarySolution` and `propagateInitialCondition` that provide an interface between the user supplied code, see Section 3.2, and the algorithms and data structures contained in folder `src/`. A brief description and implementation details of these functions are given subsequently.

3.3.1. The function `propagateInitialCondition`

Using the control parameters provided by an instance of the user-defined interface class `SIM_SETUP`, the function initializes the computational domain and an instance of the GLLE, i.e. a right-hand-side of Eq. (1). An initial field configuration is obtained by calling the method `initial_field` of the `SIM_SETUP` class. The GLLE solver, implemented using the `complex_ode` class of `scipy`'s `integrate` module, employs a pseudospectral scheme for t-propagation wherein x-derivatives are evaluated using the fast-Fourier transform provided by `scipy`'s `fftpack` module.

Upon termination of the propagation routine, the folder `data` is created if it does not exist already, and the data generated during the simulation run is stored under `data/{fName}.npz` (with `fName` as specified by the user, see Section 3.2) using the `numpy` native `npz`-format. The stored data consists of

- `x` (`numpy` array): Discrete x-mesh defining the computational domain.
- `t` (`numpy` array): Discrete t-mesh defining the time coordinates at which data is stored.
- `Axt` (`numpy` array): Discrete field $A(x, t)$ obtained during the simulation run.
- `info` (str): Metadata for data-management.

3.3.2. The function `findStationarySolution`

The provided software includes the functionality to solve for stationary solutions of the standard LLE, satisfying $\partial_t A(x, t) = 0$, starting from a user-supplied trial function. This is accomplished by finding a complex-valued solution $A_0(x)$ such that $F(A_0(x)) = 0$, wherein F is the `Nx`-dimensional object

$$F(A) = P - (1 - i\theta)A + i\partial_x^2 A + i|A|^2 A. \tag{2}$$

An ansatz for an approximate trial solution for the root-finding procedure adopted here is

$$A_{\text{trial}}(x) = A_{\text{HSS}} + A_{\text{LDS}}(x), \tag{3}$$

wherein A_{HSS} denotes a homogeneous steady-state solution, and $A_{\text{LDS}}(x)$ is a bare localized dissipative structure.

The homogeneous steady-state solution of the standard LLE for given control parameters θ and P is [7]

$$\text{Re}[A_{\text{HSS}}] = P/[1 + (I_{\text{HSS}} - \theta)^2], \tag{4}$$

$$\text{Im}[A_{\text{HSS}}] = (I_{\text{HSS}} - \theta)P/[1 + (I_{\text{HSS}} - \theta)^2], \tag{5}$$

with $I_{\text{HSS}} = |A_{\text{HSS}}|^2$ being a solution of

$$I_{\text{HSS}}[1 + (\theta - I_{\text{HSS}})^2] = P^2, \tag{6}$$

obtained by setting $\partial_t A = 0$ and $\partial_x^n A = 0$ in Eq. (1). Internally, a solution of Eq. (6) is obtained by starting from the initial estimate $I_{\text{trial}} = |P/(\theta + i)|$.

As an approximate trial function for a possible localized dissipative structure, the similarity of the LLE to a perturbed nonlinear Schrödinger equation suggests [5,20,21]

$$A_{\text{LDS}}(x) = \sqrt{2\theta}\,\text{sech}\left(\sqrt{\theta}x\right)\, e^{i\zeta}, \tag{7}$$

resembling a standard nonlinear-Schrödinger soliton. In the considered case, i.e. for a homogeneous driving field amplitude P, a constant phase $\zeta = \arccos[\sqrt{8\theta}/\pi P]$ is found as a consequence of the condition $\frac{d}{dt}\int |A_{\text{LDS}}(x)|^2\, dx = 0$ under propagation of the LLE. Note that `pyGLLE` does not directly implement the full trial solution $A_{\text{LDS}}(x)$. To allow for more flexibility in designing a stationary solution that may include one or several LDS seed pulses, the user-defined method `initial_field` of the interface class `SIM_SETUP` is expected to implement a trial solution similar to Eq. (7), see Section 4.1.

For the control parameter ranges considered in our numerical simulation we found that a trial solution given by Eq. (3), with A_{HSS} and $A_{\text{LDS}}(x)$ as detailed above, is already close to a proper stationary solution. Consequently, a rootfinding procedure for Eq. (2) implemented via the `root` solver (using a Newton–Krylov method) provided by `scipy`'s `optimize` module converges quite fast.

Upon termination of the rootfinding procedure, the folder `data_stationary_solution` is created and an output file with user-specified filename is generated (see the example in Section. 4.1). The stored data consists of

- `x` (`numpy` array): Discrete x-mesh defining the computational domain.
- `Ax0` (`numpy` array): Discrete field $A_0(x)$ resulting from the rootfinding procedure.
- `Nx` (int): Number of mesh points used for discretizing the x-coordinate.
- `xMax` (float): Bound for the x-coordinate.
- `theta` (float): frequency detuning.
- `P` (float): homogeneous driving field amplitude.

4. Sample results

For given control parameters θ and P, the standard LLE exhibits homogeneous steady-state solutions (HSS) A_{HSS} satisfying the steady-state equation Eq. (6), cubic in $I_{\text{HSS}} = |A_{\text{HSS}}|^2$. In brief, the LLE exhibits different regimes of patterns formation that can be distinguished by the value of θ. The steady state-curve $I_{\text{HSS}}(P)$

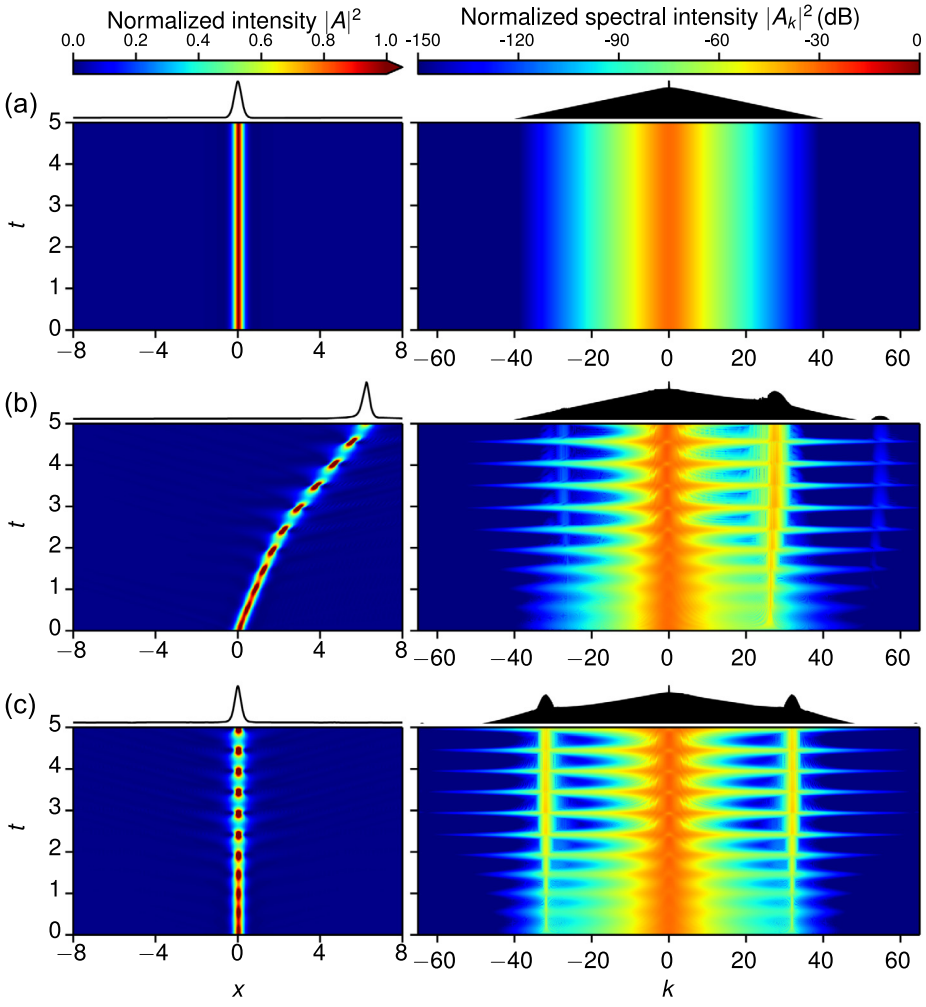

Fig. 2. Illustration of different characteristic propagation scenarios for the GLLE, starting from a stationary solution of the standard LLE. Temporal evolution of the intensity (left) and spectral intensity (right) for control parameters (a) $(\theta, P, d_2, d_3, d_4) = (15, 8, -1, 0, 0)$, (b) $(\theta, P, d_2, d_3, d_4) = (15, 8, -1, 0.04, 0)$, and, (c) $(\theta, P, d_2, d_3, d_4) = (15, 8, -1, 0, 0.001)$.

is single valued for $\theta < \sqrt{3}$ and multi-valued for $\theta > \sqrt{3}$. The examples presented below consider the control parameters $(\theta, P) = (15, 8)$, for which three solutions exist. Here, we focus on the background field $A_{\text{HSS}} = 0.037 - i0.542$ around which LDSs can develop. Detailed analysis of the dynamics of localized and patterned structures exhibited by the LLE and its generalized variants are reported in the physics and mathematics literature [12,22–24].

4.1. Finding a stationary solution for the standard LLE

As a first example, provided in folder `numExp01_stationarySolution`, we show how to solve for a stationary solution of the standard LLE with control parameters $(\theta, P) = (15, 8)$. An exemplary Python script that facilitates this task by implementing the necessary user-defined interface class `SIM_SETUP` is detailed in listing 2. Therein, in lines 15 – 17 a trial function in the form of Eq. (7) is implemented. Next, `pyGLLE` internally adds this user-supplied local disturbance to the homogeneous solution, i.e. Eqs. (4) and (5), and applies a root-finding procedure to identify a steady-state of Eq. (2). Fig. 1 compares the user-supplied trial function to the result of the root-finding procedure. As evident from the figure, the trial solution is already quite close to the optimized solution.

Listing 2: Script `main_findStationarySolution.py` setting up a trial function to search for a stationary solution of the standard LLE.

```
1  import sys; sys.path.append('../scripts/')
2  import pyGLLE
3  from numpy import pi, sqrt, cosh
4
5  class SIM_SETUP:
6      xMax = 20.
7      Nx = 2**10
8      P = 8.
9      theta = 15.
10     fName = 'stationary_solution.dat'
11
12     def initial_field(self, x):
13         st = sqrt(self.theta)
14         cosZeta = sqrt(8)*st/self.P/pi
15         sinZeta = sqrt(1-cosZeta*cosZeta)
16         Ax = sqrt(2)*st/cosh(st*x)
17         return Ax*(cosZeta+1j*sinZeta)
18
19 pyGLLE.findStationarySolution(SIM_SETUP())
```

4.2. Illustration of different propagation scenarios

As a second example, provided in folder `numExp02_propagationScenarios`, we show how to use a previously determined stationary solution of the standard LLE for t-propagation

in terms of Eq. (1). A script that implements the interface class SIM_SETUP, fetching a readily determined stationary solution from a file, is detailed in listing 3. Three exemplary propagation scenes are shown in Fig. 2. Among those, Fig. 2(a) demonstrates the stationary solution for control parameters $(\theta, P, d_2, d_3, d_4) = (15, 8, -1, 0, 0)$, obtained earlier in Section 4.1. Propagation under the dynamics governed by the GLLE, i.e. including higher orders of dispersion, a stationary solution of the LLE exhibiting a LDS can evolve into a solution with an oscillating LDS. Such a scenario with broken $x \to -x$ symmetry in case of nonzero $d_3 = 0.04$ ($d_4 = 0$) is shown in Fig. 2(b), and a scenario with maintained $x \to -x$ symmetry in case of nonzero $d_4 = 0.001$ ($d_3 = 0$) is shown in Fig. 2(c). The broken space-inversion symmetry exhibited in Fig. 2(b), illustrated by the drift of the localized structure towards higher x-values upon propagation, can be interpreted as the consequence of a spectral recoil experienced by the LDS [20,25]. The effect of high-order dispersion on frequency comb generation in optical microresonators for a variant of the LLE with different normalization than Eq. (1), is thoroughly studied in Ref. [26].

Listing 3: Script `main_propagateInitialCondition.py` invoking the GLLE solver using a stationary LLE solution as initial condition.

```
import sys; sys.path.append('../scripts/')
import pyGLLE
import numpy as np

class SIM_SETUP:
    _path = './data_stationary_solution/'
    _fName = 'stationary_solution.dat.npz'
    _data = np.load(_path+_fName)
    _x    = _data['x']
    _Ax0  = _data['Ax0']
    xMax  = float(_data['xMax'])
    Nx    = int(_data['Nx'])
    P     = float(_data['P'])
    theta = float(_data['theta'])

    tMax = 6.0
    Nt = 10000
    nSkip = 20
    d2 = -1.00
    d3 = float(sys.argv[1])
    d4 = float(sys.argv[2])

    fName = 'outFile.dat'

    def initial_field(self, x):
        return self._Ax0

pyGLLE.propagateInitialCondition(SIM_SETUP())
```

5. Impact and conclusions

In this paper we presented pyGLLE, a user-friendly software package for studying the temporal evolution of complex fields governed by a generalized Lugiato–Lefever equation. It is based on our research code, developed with a strong focus on extendibility and reusability. Its utility can extend well beyond the specialized context of the GLLE since its modules `data_handler.py` and `solver.py` can directly be used to assemble software solutions for the simulation of other systems, modeled by first-order partial differential equations. The examples presented in Section 4 briefly describe the usage of the software by illustrating small workflows ranging from the specification of a propagation setup to data visualization. They can be run on a laptop and are intended to show how easy it is to access the core-functionality of pyGLLE and how the generated raw-data can be read in and post-processed. We further pointed out implementation details that can guide the interested user in adapting the software to his or her needs. Particular applications of interest include the study of the dynamics of localized structures supported by the GLLE, and frequency comb generation in nonlinear microring resonators.

The presented software has been extensively used in our research work, including the study of resonant emission of multi-frequency radiation by oscillating dissipative solitons in presence of third order dispersion [27], and the dynamics of LDSs in the GLLE with negative quartic group-velocity dispersion [28]. It can be used to reproduce and build on the results reported therein. The pyGLLE software tool, including scripts that implement the sample results in Section 4, is available for download under [19].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We acknowledge support from the Deutsche Forschungsgemeinschaft (DFG), Germany under Germany's Excellence Strategy within the Cluster of Excellence PhoenixD (Photonics, Optics, and Engineering – Innovation Across Disciplines) (EXC 2122, projectID 390833453). The publication of this article was funded by the Open Access Fund of the Leibniz Universität Hannover.

References

[1] Lugiato LA, Lefever R. Spatial dissipative structures in passive optical systems. Phys Rev Lett 1987;58:2209–11.
[2] Zabusky NJ, Kruskal MD. Interaction of "solitons" in a collisionless plasma and the recurrence of initial states. Phys Rev Lett 1965;15:240–3.
[3] Drazin PG, Johnson RS. Solitons: An introduction. Cambridge: Cambridge University Press; 1989.
[4] Kivshar YS, Agrawal GP. Optical solitons: From fibers to photonic crystals. San Diego: Academic Press; 2003.
[5] Cardoso WB, Salasnich L, Malomed BA. Localized solutions of Lugiato-Lefever equations with focused pump. Sci Rep 2017;7:16876.
[6] Akhmediev N, Ankiewicz A. Dissipative solitons. Berlin, Heidelberg: Springer-Verlag; 2005.
[7] Parra-Rivas P, Gomila D, Colet P, Gelens L. Interaction of solitons and the formation of bound states in the generalized Lugiato-Lefever equation. Eur Phys J D 2017;71:198.
[8] Krupa K, Nithyanandan K, Andral U, Tchofo-Dinda P, Grelu P. Real-time observation of internal motion within ultrafast dissipative optical soliton molecules. Phys Rev Lett 2017;118:243901.
[9] Wang ZQ, Nithyanandan K, Coillet A, Tchofo-Dinda P, Grelu P. Optical soliton molecular complexes in a passively mode-locked fibre laser. Nature Commun 2019;10:830.
[10] Stratmann M, Pagel T, Mitschke F. Experimental observation of temporal soliton molecules. Phys Rev Lett 2005;95:143902.
[11] Coen S, Randle HG, Sylvestre T, Erkintalo M. Modeling of octave-spanning Kerr frequency combs using a generalized mean-field Lugiato-Lefever model. Opt Lett 2013;38:37–9.
[12] Parra-Rivas P, Gomila D, Matías MA, Coen S, Gelens L. Dynamics of localized and patterned structures in the Lugiato-Lefever equation determine the stability and shape of optical frequency combs. Phys Rev A 2014;89:043813.
[13] Gelens L, Gomila D, Van der Sande G, Danckaert J, Colet P, Matías MA. Dynamical instabilities of dissipative solitons in nonlinear optical cavities with nonlocal materials. Phys Rev A 2008;77:033841.
[14] Tlidi M, Gelens L. High-order dispersion stabilizes dark dissipative solitons in all-fiber cavities. Opt Lett 2010;35:306–8.
[15] Rossum G. Python reference manual. Amsterdam, The Netherlands, The Netherlands: CWI (Centre for Mathematics and Computer Science); 1995.
[16] Jones E, Oliphant T, Peterson P, et al. SciPy: Open source scientific tools for Python. 2020, [Online; accessed 2020-03-09] (2001–2018) http://www.scipy.org/.
[17] Noble WS. A quick guide to organizing computational biology projects. PLoS Comput Biol 2009;5:e1000424.

[18] Hairer E, Norsett SP, Wanner G. Solving ordinary differential equations I (2nd Revised Ed.): Nonstiff problems. Berlin: Springer-Verlag; 1993.
[19] Melchert O. pyGLLE – Python toolkit for solving the generalized Lugiato-Lefever equation. 2020, [Online; accessed 2020-03-09] https://github.com/omelchert/pyGLLE.git.
[20] Milián C, Skryabin DV. Soliton families and resonant radiation in a micro-ring resonator near zero group-velocity dispersion. Opt Express 2014;22:3732–9.
[21] Jang JK, Erkintalo M, Luo K, Oppo G-L, Coen S, Murdoch SG. Controlled merging and annihilation of localised dissipative structures in an AC-driven damped nonlinear Schrödinger system. New J Phys 2016;18:033034.
[22] Miyaji T, Ohnishi I, Tsutsumi Y. Bifurcation analysis to the Lugiato-Lefever equation in one space dimension. Physica D 2010;239:2066–83.
[23] Mandel R, Reichel W. A priori bounds and global bifurcation results for frequency combs modeled by the Lugiato-Lefever equation. SIAM J Appl Math 2017;77:315–45.
[24] Hakkaev S, Stanislavova M, Stefanov AG. On the generation of stable Kerr frequency combs in the Lugiato-Lefever model of periodic optical waveguides. SIAM J Appl Math 2019;79:477–505.
[25] Akhmediev N, Karlsson M. Cherenkov radiation emitted by solitons in optical fibers. Phys Rev A 1995;51:2602–7.
[26] Bao C, Taheri H, Zhang L, Matsko A, Yan Y, Liao P, et al. High-order dispersion in Kerr comb oscillators. J Opt Soc Amer B 2017;34:715–25.
[27] Melchert O, Demircan A, Yulin A. Multi-frequency radiation of dissipative solitons in optical fiber cavities. Sci Rep 2020;10:8849.
[28] Melchert O, Yulin A, Demircan A. Dynamics of localized dissipative structures in a generalized Lugiato-Lefever model with negative quartic group-velocity dispersion. Opt Lett 2020;54:2764–7.