Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

NTMpy: An open source package for solving coupled parabolic differential equations in the framework of the three-temperature model[☆,☆☆]

Lukas Alber[a], Valentino Scalera[b], Vivek Unikandanunni[a], Daniel Schick[c], Stefano Bonetti[a,d,*]

[a] Department of Physics, Stockholm University, 106 91 Stockholm, Sweden
[b] Department of Electrical Engineering and ICT, University of Naples Federico II, 80125 Naples, Italy
[c] Max Born Institute for Nonlinear Optics and Short Pulse Spectroscopy, Max-Born-Straße 2A, 12489 Berlin, Germany
[d] Department of Molecular Sciences and Nanosystems, Ca' Foscari University of Venice, 30172 Venezia-Mestre, Italy

ARTICLE INFO

Article history:
Received 6 April 2020
Received in revised form 8 March 2021
Accepted 14 March 2021
Available online 22 April 2021

Keywords:
Ultrafast condensed matter dynamics
Coupled parabolic differential equations
N-temperature model
Transfer matrix method

ABSTRACT

The NTMpy code package allows for simulating the one-dimensional thermal response of multilayer samples after optical excitation, as in a typical pump-probe experiment. Several Python routines are combined and optimized to solve coupled heat diffusion equations in one dimension, on arbitrary piecewise homogeneous material stacks, in the framework of the so-called three-temperature model. The energy source deposited in the material is modelled as a light pulse of arbitrary cross-section and temporal profile. A transfer matrix method enables the calculation of realistic light absorption in presence of scattering interfaces as in multilayer samples. The open source code is fully object-oriented to enable a user-friendly and intuitive interface for adjusting the physically relevant input parameters. Here, we describe the mathematical background of the code, we lay out the workflow, and we validate the functionality of our package by comparing it to commercial software, as well as to experimental transient reflectivity data recorded in a pump-probe experiment with femtosecond light pulses.

Program summary
Program title: NTMpy v.0.1.1
CPC Library link to program files: https://doi.org/10.17632/5czr76gmwr.1
Developer's repository link: https://github.com/udcm-su/NTMpy
Code Ocean capsule: https://codeocean.com/capsule/5661399
Licensing provisions: MIT license
Programming language: Python
External routines: Python 3.5 or higher, numpy, matplotlib, bsplines, tqdm
Nature of problem: 1-dimensional coupled non linear partial differential equations; diffusion and relaxation dynamics formultiple systems and multiple layers.
Solution method: Simulate the diffusion and relaxation dynamics of up to 3 coupled systems via an object oriented user interface. In order to approximate the solution and its derivatives in space B-Spline interpolation is used. The solution is developed in time via the Explicit Euler method.
Additional comments including restrictions and unusual features: A routine to automatically select the ideal time step for stability of the algorithm is implemented. Routines for output of raw data in order to post process and pre- made visualization routines are implemented.

1. Introduction

The understanding of how heat is transported at the nanometer level and at ultrafast time scales in different materials is an open question in modern condensed matter research. The increasing availability of commercial femtosecond laser systems makes it

☆ The review of this paper was arranged by Prof. D.P. Landau.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
* Corresponding author.
E-mail address: stefano.bonetti@fysik.su.se (S. Bonetti).

https://doi.org/10.1016/j.cpc.2021.107990
0010-4655/© 2021 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

possible to study material dynamics at sub-picosecond time scales, allowing for investigating non-equilibrium energy transport. This also asks for numerical computations able to model the experimental evidence and to either validate or extract a set of physical parameters.

A commonly used strategy to describe the highly non-equilibrium processes induced by ultrafast laser excitation, is the N-temperature model ($N = 2$ or 3, typically) posed by Anisimov et al. [1]. This model is a set of coupled parabolic differential equations, which describe the temporal evolution of the energy of the electron, lattice, and spin systems as well as the transfer of energy between these systems. The N-temperature model is used to simulate a broad range of experiments in the ultrafast community, e.g. Ref. [2–5], but an open source implementation of it is still missing.

Here, we provide an open source, user-friendly Python package able to solve the N-temperature model in one-dimension, with arbitrary, piece-wise homogeneous layers with different physical properties [6]. The time-step selection to optimize the solving routines and guarantee their convergence, the calculation of the deposited energy by an arbitrarily shaped laser pulse using the transfer matrix method, and visualization routines, are all automatized for the early-user. However, the object oriented design of the package, also allows for easy customization for the advanced users.

After laying out the mathematical background on which the implementation of the solver is based on in Sec. 2, we introduce the workflow of the program by showing the most important commands and the output, users can expect in Sec. 3. To validate our results we compare the computation of this open source package to commercial software, and further more to real experimental data, in Sec. 4.

2. Mathematical methods and background

When a material is illuminated by light, the energy of the electromagnetic radiation is partly reflected, partly transmitted through the material, and partly absorbed by it. The absorbed electromagnetic energy essentially heats the material, rising its temperature. While this is a simple problem to solve for the case of a continuous light source, it becomes immediately more complex for the case of an ultrashort laser pulses, i.e. with sub-picosecond pulse duration. In this case, the concept of temperature is in itself an ill-defined one, because the different energy reservoirs in a material (the electronic E and lattice L systems, and also the spin S system for a magnetically ordered sample) respond on different time scales and are not in equilibrium among each other. Generally, only the electrons can react fast enough to the ultrashort laser pulse, and the energy is released from the electronic system to the other heat reservoirs only at a later time.

To treat this problem, three reservoirs are considered, in order to allow for the definition of a temperature and to solve the heat equation in time, and then coupled to allow for the energy to flow between them. This is addressed by considering three independent reservoirs, in order to allow for the definition of a temperature and to solve the heat equation in time in each of the systems. Finally, the three systems are coupled such that the energy can flow between them. This idea is generalized to a N-temperature model, describing the heat exchange between the systems and the heat diffusion along the one-dimensional, multiple layered material. Each of the systems - electron, lattice, and spin - has its individual temperature $T^{E,L,S}(x,t)$, where x denotes the depth in the specimen and t is the time.

The dynamics of every system are described by a parabolic partial differential equation, namely

$$\begin{cases} C^E(T^E)\cdot\rho\cdot\partial_t T^E = \partial_x\left(k^E(T^E)\cdot\partial_x T^E\right) + \\ \qquad G^{EL}\cdot(T^L - T^E) + G^{SE}\cdot(T^S - T^E) + S(x,t)\,, \\ C^L(T^L)\cdot\rho\cdot\partial_t T^L = \partial_x\left(k^L(T^L)\cdot\partial_x T^L\right) + \\ \qquad G^{EL}\cdot(T^E - T^L) + G^{LS}\cdot(T^S - T^L)\,, \\ C^S(T^S)\cdot\rho\cdot\partial_t T^S = \partial_x\left(k^S(T^S)\cdot\partial_x T^S\right) + \\ \qquad G^{SE}\cdot(T^E - T^S) + G^{LS}\cdot(T^L - T^S)\,. \end{cases} \tag{1}$$

Here $C^{E,L,S}(T)$ and $k^{E,L,S}(T)$ are the specific heat capacity and thermal conductivity, which can be defined as constants or as functions of the respective temperature $T^{E,L,S}$. Since we consider a stack of multiple layers C^i and k^i also depend on the respective layer, hence they are $C^i_s(T^i)$ and $k^i_s(T^i)$, the dependence on the layer s will be omitted when not necessary in order to have a simpler notation.
The term $S(x,t)$ is responsible for the heat injection to the electronic system and physically corresponds to a pulsed laser source hitting the sample at the surface. Note that in Eq. (1), we assume the coupling G, responsible for the heat exchange and relaxation between the systems, to be linear as in the formulation from Anisimov et al., Ref. [7], [8] and other groups. However, it should be mentioned, that considering a non-constant $G(T)$ is current subject to research [9].

The temperature profile is expressed as a linear combination of basis functions $B_m(x)$ for $m = 1,\ldots,M$ with time depending coefficients for every subsystem, $i \in \{E, L, S\}$, $c^i_m(t)$, i.e.

$$T^i(x,t) = \sum_{m=1}^{M} c^i_m(t) B_m(x)\,. \tag{2}$$

The solution of Eq. (1) is now reduced to a finite dimensional problem and consequently the diffusion equation cannot generally be solved exactly in the whole domain.

The solution can be approximated by imposing Eq. (1) to be satisfied on a given grid of points $\{x_1, x_2, x_3, \ldots, x_{M-1}\}$, i.e. *collocation points*. Two additional points x_0 and x_M are added to the grid on the boundary of the domain to impose the boundary conditions.

The temperatures and their derivatives have an exact analytic expression at the grid points

$$\begin{aligned} T^i(x_j,t) &= \sum_{m=1}^{M} c^i_m(t) B_m(x_j)\,, \\ \frac{\partial T^i}{\partial x}(x_j,t) &= \sum_{m=1}^{M} c^i_m(t) \frac{\partial B_m}{\partial x}(x_j)\,, \\ \frac{\partial^2 T^i}{\partial x^2}(x_j,t) &= \sum_{m=1}^{M} c^i_m(t) \frac{\partial^2 B_m}{\partial x^2}(x_j)\,. \end{aligned} \tag{3}$$

By introducing the $M \times M$ matrices D_0, D_1 and D_2 with generic elements

$$\begin{aligned} \{D_0\}_{jm} &= B_m(x_j)\,, \\ \{D_1\}_{jm} &= \frac{\partial B_m}{\partial x}(x_j)\,, \\ \{D_2\}_{jm} &= \frac{\partial^2 B_m}{\partial x^2}(x_j)\,, \end{aligned} \tag{4}$$

the temperatures and their derivatives can be obtained by matrix products. By using the Leibnitz formula Eq. (1) can be the reformulated as follows

$$\rho C^i D_0 \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{c}^i =$$

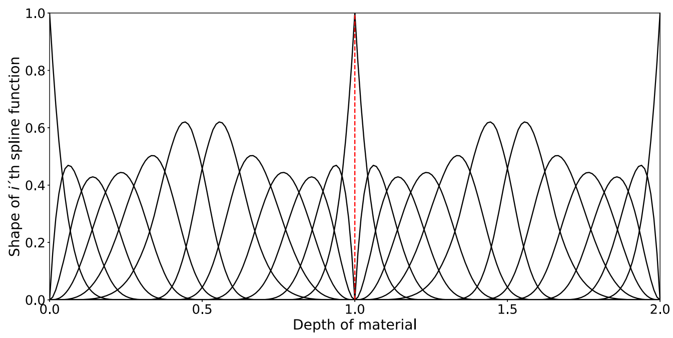

Fig. 1. A set of order 5-continuous B-Splines for a two layer system, used to represent the solution in space, where each layer has a length of 1.

$$k^i(D_0\boldsymbol{c}^i)\cdot D_2\boldsymbol{c}^i+\frac{\mathrm{d}k_s^i}{\mathrm{d}(D_0\boldsymbol{c}^i)}\cdot(D_1\boldsymbol{c}^i)^2+S^i(x,t)+ \sum_{\substack{k\in\{E,L,S\}\\k\neq i}}G^{ik}(D_0(\boldsymbol{c}^k-\boldsymbol{c}^i))\,, \quad (5)$$

for $i=\{E,L,S\}$, such that $\boldsymbol{c}_i=\{c_1^i,c_2^i,\dots,c_M^i\}$ is the vector of coefficients relative to the i-th temperature, the dot $(\cdot)$ denotes the elementwise product of two vectors and $(\boldsymbol{a})^2$ denotes the vector whose elements are the squares of the elements of the vector $\boldsymbol{a}$.

The time evolution of the coefficient vectors $\boldsymbol{c}^i$ is evaluated using the explicit Euler formula, which reads

$$D_0\boldsymbol{c}^i(t+\Delta t)=D_0\boldsymbol{c}^i(t)+\Delta t D_0\frac{\mathrm{d}\boldsymbol{c}^i}{\mathrm{d}t}(t)\,, \quad (6)$$

where the time derivative is calculated as shown in Eq. (5) and S is present only for $i=E$.

When an analytical formula for $\mathrm{d}k^i/\mathrm{d}T^i$ is unavailable, this derivative can be computed numerically without introducing significant errors since the conductivity is typically a regular function. Equation (6) must be completed with the initial- and boundary conditions at the left and right end of the material under consideration.

The boundary conditions in this software can be of Dirichlet type or of a modified Neumann type according to whether the temperature or the heat flux is assigned at the boundaries. Hence we have the two following options for the boundary conditions

$$\text{Dirichlet}:\begin{cases}(D_0)[\,0,:]\;\boldsymbol{c}^i=T_{BC}(x_0,t)\\(D_0)[M,:]\;\boldsymbol{c}^i=T_{BC}(x_M,t)\end{cases}\quad,\ \text{or}$$

$$\text{Neumann}:\begin{cases}k^i(T^i(x_0))((D_0\boldsymbol{c}^i)D_1)[\,0,:]\;\boldsymbol{c}^i=H_{BC}(x_0,t)\\k^i(T^i(x_M))((D_0\boldsymbol{c}^i)D_1)[M,:]\;\boldsymbol{c}^i=H_{BC}(x_M,t)\,,\end{cases}$$

where the notation $[\,0,:]$ and $[M,:]$ indicates the first and the last row of the matrix and T_{BC} or H_{BC} are the assigned boundary conditions.

The basis functions chosen for the approximation of the solution are B-Splines realized with the Cox-de Boor algorithm. These splines are continuous and have continuous derivatives up to a chosen order. A set of B-Spline basis functions, as used in the software, are depicted in Fig. 1, where a stack of two layers is under consideration.

The use of C^k-continuous functions is justified by the nature of the physical problem: the presence of a discontinuity in the temperature would cause an infinite heat flux, while a discontinuity in its first derivative would imply a finite heat flux into an infinitesimal control volume unless the conductivity is discontinuous.

Discontinuities in the conductivity can be present when the specimen is made of two or more different materials stacked together. In this case the heat flowing into the interface between the s-th and the $(s+1)$-th layer, for the i-th temperature is

$$H_{\text{Interface}}=k_{i,s}(T^i)\lim_{x\to x_I^-}\partial_x T^i-k_{i,s+1}(T^i)\lim_{x\to x_I^+}\partial_x T^i=0\,, \quad (7)$$

where the second subscript of k indicates the layer, x_I is the position of the interface and the superscript $+$ and $-$ indicates the limit from right and the left respectively.

In order to correctly represent the temperatures at the interface a different set of B-Spline is considered for each layer (see Fig. 1), and the continuity of the temperature and the condition Eq. (7), i.e. the conservation of the heat flow, are imposed at the interface.

Notice that when the coefficients c_{im} are determined the limit and derivative appearing in Eq. (7) are analytically computed.

2.1. Evaluation of the time step

A crucial point for the precision, speed and stability of the code is the choice of an appropriate time step. When this is not supplied by the user, the time step is automatically determined according to a criterion guaranteeing stability on one side, but keeping the running time of the simulation as small as possible on the other side.

For a linear N-temperature system with a single layer the dynamical matrix is given by

$$\frac{1}{\rho}\mathrm{diag}\left\{\frac{k^E}{C^E},\frac{k^L}{C^L},\frac{k^S}{C^S}\right\}\otimes(D_0^{-1}D_2)+\frac{1}{\rho}M\otimes I_N\,, \quad (8)$$

with

$$M=\begin{bmatrix}-\dfrac{G^{EL}+G^{SE}}{C^E} & \dfrac{G^{EL}}{C^E} & \dfrac{G^{SE}}{C^E}\\ \dfrac{G^{EL}}{C^L} & -\dfrac{G^{EL}+G^{LS}}{C^L} & \dfrac{G^{LS}}{C^L}\\ \dfrac{G^{SE}}{C^S} & \dfrac{G^{LS}}{C^S} & -\dfrac{G^{LS}+G^{SE}}{C^S}\end{bmatrix} \quad (9)$$

where I_N is the N dimensional identity matrix and $\otimes$ denotes the Kronecker product. In this system, to preserve stability, the time step needs to satisfy the condition

$$\Delta t<\frac{2}{|\lambda|_{max}}\,, \quad (10)$$

where $|\lambda|_{max}$ is the eigenvalue of the dynamical matrix with the largest absolute value. We note that, Δt depends on the input parameters C, k, G and on the desired spatial resolution, represented in D_0 and D_2, but not on the heating source term $S(x,t)$.

In the nonlinear context we can still use condition (10) with the eigenvalues of an adapted version of matrix (8). We consider the worst scenario for stability represented by contemporaneous large k^i's (thermal conductivities) and small C^i's (specific heats), which produce large eigenvalues in absolute value. To this purpose, in the worst scenario matrix, the k^i's are replaced by their maximum, evaluated on a set of values of T^i and the C^i's are replaced by their minimum evaluated on the same set. This yields the following matrix

$$\frac{1}{\rho}\mathrm{diag}\left\{\frac{k^E_{max}}{C^E_{min}\rho},\frac{k^L_{max}}{C^L_{min}\rho},\frac{k^S_{max}}{C^S_{min}\rho}\right\}\otimes(D_0^{-1}D_2)+\frac{1}{\rho}M_{max}\otimes I_N\,, \quad (11)$$

where

$$M_{max} = \begin{bmatrix} -\frac{G^{EL}+G^{SE}}{C^E_{min}} & \frac{G^{EL}}{C^E_{min}} & \frac{G^{SE}}{C^E_{min}} \\ \frac{G^{EL}}{C^L_{min}} & -\frac{G^{EL}+G^{LS}}{C^L_{min}} & \frac{G^{LS}}{C^L_{min}} \\ \frac{G^{SE}}{C^S_{min}} & \frac{G^{LS}}{C^S_{min}} & -\frac{G^{LS}+G^{SE}}{C^S_{min}} \end{bmatrix} \tag{12}$$

As this linearization, from $k_i(T)$ to k_i^{max}, depends on the set of values chosen for T, typical values of temperatures reached in the experiments on ultrafast dynamics are considered in the default case. That is, $T \in [270, 3000]$ K, but they can be changed by the user via `sim.stability_lim([lowlimit,highlimit])`. Considering the worst scenario allows to evaluate the time step only once at the beginning of the simulation.

In multilayer systems the above procedure is repeated for each layer obtaining $\Delta t_1, \Delta t_2, \ldots, \Delta t_L$ where L is the number of layers, the time step is $\Delta t = \min\{\Delta t_1, \Delta t_2, \ldots, \Delta t_L\}$.

3. Background and implementation

The code is implemented in Python with dependence on the *numpy* and the *bspline* package for the numerical computation, on the *matplotlib* library for plotting of the results and the *progressbar* package to monitor the elapsed time. Installation of the NTMpy package will automatically check if the dependencies are fulfilled and download the additional software if not.

In order to make the user interface friendly, the code is object oriented and it can be used either with command line or in a script. The package contains three main classes which are `source`, `simulation`, and `visual`: While the `source` class is a collection of methods for the generation of the energy injection matrix, the `simulation` class handles the computation of the solution and builds the core of the program. After a solution has been found, it can be passed on to the `visual` classes allowing a fast and easy depiction of the results respectively.

3.1. Data input

Even though in principle the source function $S(x, t)$, injecting heat in space and time, can be of generic type, the most common types of sources are already defined in the code and one needs to specify only the main properties. That is, the user can choose between Lambert Beer's law or the transfer matrix method (TMM) to calculate the absorption profile in space and independently from that select either a Gaussian, a repeated Gaussian or even a custom time profile to evaluate the shape of the heating source in time.

For example a source with a Gaussian profile in time and exponential decay in space, considering multiple reflections, incident angle and polarization, i.e. TMM, is introduced with the following lines of code.

```
#Define a Source
s                   = source()
# Set source type
s.spaceprofile      = 'TMM'
s.timeprofile       = 'Gaussian'
# Width of the Gaussian (in s)
s.FWHM              = 0.1e-12
# Area under the Gaussian (in J/m^2)
s.fluence           = 6*1e-3/1e-2**2
# Set the time of the Gaussian peak (in s)
s.t0                = 1e-12
# Wavelength in vacuum (in nm)
s.lambda_vac        = 400
# Incident angle (in rad)
# (0 is perpendicular to the surface)
s.theta_in          = pi/4
s.polarization      = 'p'
```

Note, that there is also a predefined way to calculate the spacial absorption according to Lambert Beerś law via `s.spaceprofile = 'LB'`, which follows, except for the input of the incident angle and the polarization of the light, the same commands as above. Alternatively a custom profile in time can be given for the source, if arrays of data, here `my_time` and `my_intensity`, are provided to the program. Now considering the Lambert Beer decay law in space, the commands to initialize a customized source are

```
# Set source type
s.spaceprofile = 'LB'
s.timeprofile  = 'Custom'
# Set the value for the source
s.loadData = [my_time, my_intensity]
```

After the initialization of a source, one can proceed to initialize the simulation object, providing material specific parameters for every layer of the stack under consideration and then run the simulation. The constructor of the simulation class has a mandatory input which is the number `N` of temperatures under consideration and an optional input which is the source `s`.

```
# Define the simulation object
sim = simulation(N, s)
```

`N` can be 1, 2 or 3. When the properties of `s` are not specified as shown above, by default it is assumed that there is no source term in Eq. (1), i.e. the fluence is 0.

The properties of the media are assigned by adding sequentially the layers of the materials with the method `addLayer()`. For instance

```
# Add a layer  for a 2 temperature model
sim.addLayer(length, n, [k1, k2],
                [C1, C2], density, G)
```

where the inputs of the method are in order the length of the layer (`length`), the complex refractive index of the layer (`n`), the list of the thermal conductivities of each temperature system (`[k1, k2]`), the list of the specific heats of each temperature system (`[C1, C2]`), the mass density (`density`) and the exchange coupling (`G`).

Thermal conductivities and specific heats can either be numbers or `lambda` functions when they vary with temperature. In case $N = 3$, `G` can be a list containing the coupling constants G_{12}, G_{23} and G_{31} in this order.

Before running the simulation, a final time must be given by the user through the command

```
# Set final time (in s)
sim.final_time = final_time
```

The commands described so far are sufficient to run the simulation. If no further input is given some default values are selected according to the most common conditions or models employed for the experiments.

By default the time step is assigned automatically through the procedure illustrated in Sec. 2.1. However the user can define a different time step by

```
sim.time_step = time_step
```

The usual initial condition for all the temperatures T^i by default is $T^i = 300$ K. The user can modify the initial condition through the command line

```
# Modify initial condition of i-th temp. (in K)
sim.changeInit( i, T0)
```

where `i` varies between 1 and `N`, indicating, that different initial conditions can be set for each subsystem. Furthermore the initial temperature `T0` can be either a number or a `lambda` function describing the space profile of the temperature $T(x, 0)$.

By default the boundary conditions are of modified Neumann type with no heat flux through the boundaries, corresponding to an insulated system. The boundary condition type and value can be modified through the following methods

```
# Change BC type for the i-th temp
sim.changeBC_Type( i, side, Type)
# Change BC value for the i-th temp
sim.changeBC_Value( i, side, BC)
```

where again `i` is the index of the temperature system subject to the change, `side` can be either `'left'` or `'right'` depending on which side the change is applied to, `Type` is either `'dirichlet'` or `'neumann'` respectively, and `BC` is a constant or a `lambda` function providing the value of the boundary condition as the time varies.

Once the input of the data is complete the simulation is executed by the command

```
# Compute temperature map
[x, t, T] = sim.run()
```

It yields the `numpy.array` containing the $2 + \mathrm{N}$-dimensional arrays `T` of the temperature of the N subsystems on the space grid `x` at the times `t`.

3.2. Plot results

Once the simulation has been executed the results, that is the dynamics of all respective temperatures in space and time, are provided to the user in array form, as described above. However, in order to make the visualization both, easier and quicker for the user, a separate class with relevant plotting methods has been defined. This gives the user the freedom to output the raw data and do post-processing on their own but also to use the visualization class in order to depict some properties immediately.

Among others, there are methods to visualize the heating source $S(x, t)$, a contour plot of the temperature in space and time $T^{E,L,S}(x, t)$, but also some simple ready made post-process routines like the plot of the average temperature of a layer over time, see Fig. 2. Moreover the package provides a method to play an animation of how all the temperature systems evolve in time which can be useful either to visually detect the impact of the fundamental mechanisms in the case considered, or for pedagogical use. Different routines can be called by following the commands

```
# Creating a visual object
v = visual(sim)
# Depicting results and obtaining raw data
source = v.source()
[timegrid,average_temp] = v.average()
v.contour('1')
v.animation(speed,save)
```

where the output of the first two visualization routines are arrays, which allow users, to look and process the raw data themselves. The only visualization methods that require input arguments are `v.contour('N')`, where `N` can be 1,2 or 3, corresponding to electron-, the lattice- or the spin- system, and `animation(speed,save)`, where the speed of the animation can be adjusted and the user can decide whether they want to save the animation, with `save = 1` or not, with `save = 0`.

This makes it easy to quickly check results and analyze properties of the dynamics of the simulation.

4. Package validation

4.1. Comparison with other simulation tools

In order to validate the results obtained from the NTMpy package, the focus is set on the two main parts of the software. First,

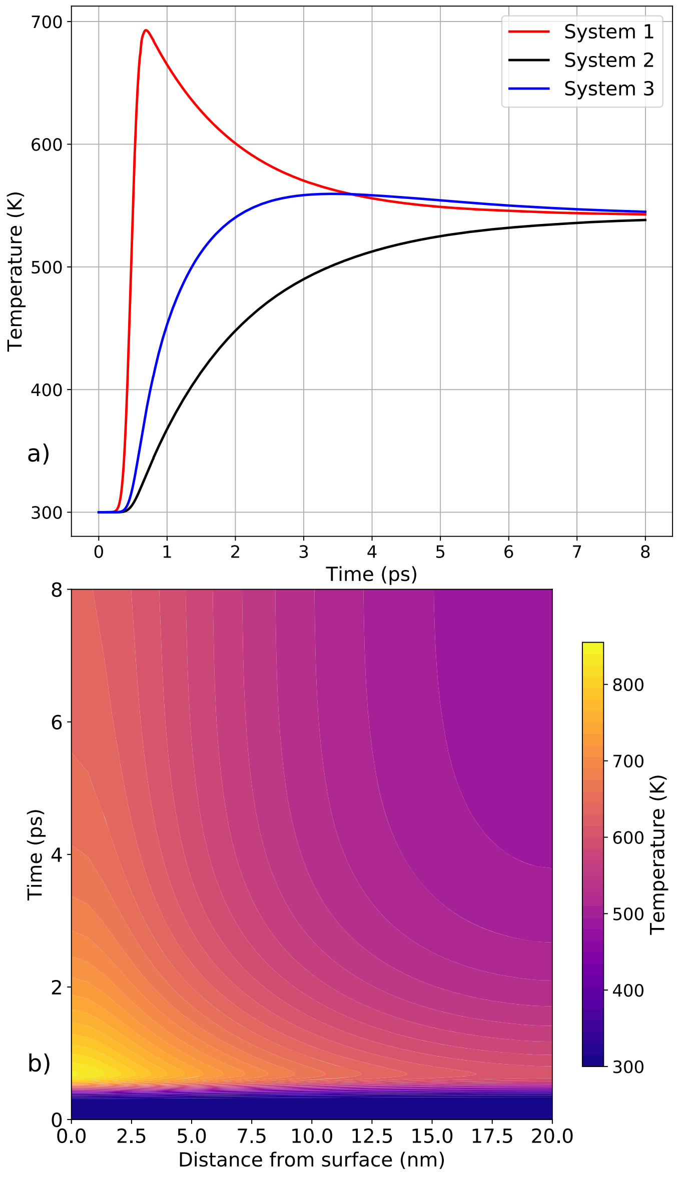

Fig. 2. Example of a 3-temperature simulation and the output of two visualization methods: a) `v.average()` and b) `v.contour('1')` described in the text. Here a 20 nm ferromagnetic nickel thin film, heated by a femtosecond laser source is under consideration. The physical parameters are taken from Ref. [2]. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

the evaluation of the local absorption with respect to the incident laser pulse, and second, the performance of the solver itself. Looking at Eq. (1), we are interested of how accurately we are computing the source term $S(x, t)$ responsible for heating and how well our obtained solution $T(x, t)$ matches with commercial software, when experimentally relevant cases are under investigation.

For the evaluation of the local absorption of $S(x, t)$, we are considering a [Pt 3 nm|Co 15 nm|Cr 5 nm|MgO]-material stack and compute the local absorption per unit incident power, with respect to the distance from the surface, with the implemented transfer matrix method. The obtained profile is compared to the result, obtained from COMSOL Multiphysics™, Ref. [10], simulations, where the same light and material parameters are considered. In Fig. 3 a), we see, that both, the commercial and our open source software show an overlapping result (mean relative discrepancy $\overline{\delta(x)} = 2.2\%$).

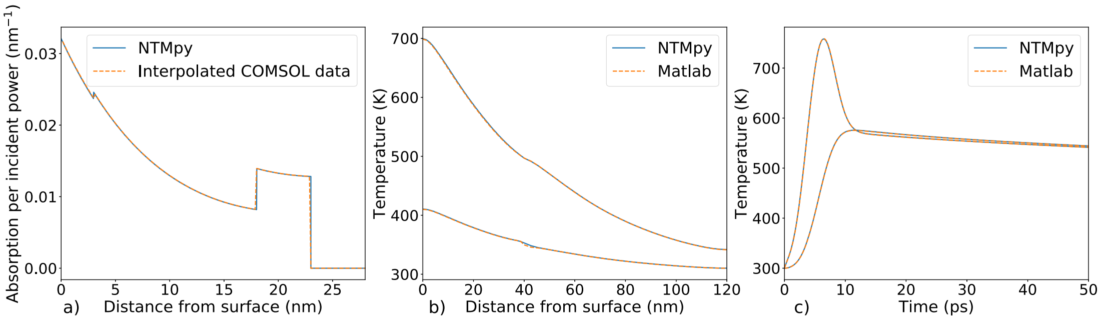

Fig. 3. (a) Local absorption per incident laser power computed with NTMpy's TMM-module and with the commercial software COMSOL Multiphysics™, Ref. [10]. For both simulations a three-layer material with identical parameters is considered. Line cuts for two coupled temperature systems $T^{E,L}(x,t)$ at (b) a fixed time $t = t_0$ and at (c) a fixed point in space $x = 0$. NTMpy's solutions for both systems (electronic and lattice), computed for a two-layer stack, are compared to Matlab's built-in partial differential equation `pdepe()` solver, Ref. [11].

Table 1
Material parameters used for the TTM simulation in Fig. 5.

Parameter	Symbol/units	Platinum	Silicon
Thickness	l (nm)	10	100000
Refractive index (at 400 nm)	n	1.7176 + i2.844 [13]	5.5674 + 0.38612j [14]
Electron heat conductivity	k_e (Wm^{-1} K^{-1})	72 [15]	130 [16]
Lattice heat conductivity	k_l (Wm^{-1} K^{-1})	72 [15]	$k_l(T)$ [17]
Electron heat capacity	$C_e(T)$ (J kg^{-1} K^{-1})	$740/\rho_{Pt} T_e$ [15]	$150/\rho_{Si} \cdot T_e$ [17]
Lattice heat capacity	C_l (J kg^{-1} K^{-1})	$2.78E6/\rho_{Pt}$ [15]	$1.6E6/\rho_{Si}$ [17]
Electron phonon coupling	G (Wm^{-3} K^{-1})	2.5E17 [18]	18E17 [19]

For the evaluation of computation time and accuracy of the obtained solution, we compare $T(x,t)$ to Matlabś, Ref. [11], built in partial differential equation, `pdepe()`, - solver. Again we consider the same light and material parameters for both software and obtain output that is very much in agreement, with respect to space Fig. 3 b) and time c) dimension. That is the relative error of the temperature enhancement $\frac{T_{NTMpy} - T_{Matlab}}{T_{Matlab} - 300} \leq 6\%$. Also the time needed until a solution is obtained from our free package is comparable to Matlab's performance.

4.2. Comparison with experiment

Even though the software is able to solve generic coupled non linear 1-D diffusion equations in the form of Eq. (1), the focus is laid on heat transfer between systems and transport along the material, within the framework of the two/three temperature model.

If the physical parameters of all the material layers under consideration are known, the software can be used to depict the temperature dynamics in space and time of the selected configuration. This provides an opportunity to be able to detect local phenomena in a material, as opposed to averaged effects, measured in most experiments. However reliable data on certain parameters is sparse and it is a current interest of research to find those properties.

In order to demonstrate the use of this software, we measured the change in reflectivity on a 10 nm platinum thin film on a silicon substrate and used the NTMpy software to perform a simulation of the temperature dynamics of the same system. The experimental data was retrieved from a pump-probe transient reflectivity measurement. According to Refs. [7,12], a linear relationship between the measured change in reflectivity and the change in temperature can be assumed, i.e. $\frac{\Delta R}{R} \propto \frac{\Delta T}{T}$. This makes it possible to compare the experiment to the simulation.

In the simulation, we used the same nominal laser source parameters as in the experiment. That is a 400 nm, p-polarized laser pulse with a FWHM of 100 fs and a fluence of 6 mJ/cm^2 of which 53% get absorbed, which is hitting the sample in a 45° angle. The local absorption profile is shown in Fig. 4. For the platinum film and the silicon substrate the parameters from Table 1 are used.

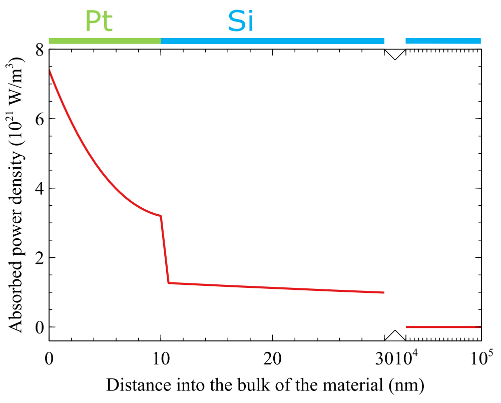

Fig. 4. Local power absorption for the platinum/silicon system under consideration, calculated via the transfer matrix method implemented in NTMpy. Here a p-polarized laser pulse with a wavelength of 400 nm, hitting the sample at a 45° angle with a fluence of 6 mJ/cm^2 and a FWHM of 0.1 ps is considered.

The simulation is set up as follows:

```
sim = simulation(2,s)
sim.addLayer(length_Pt,n_Pt,[k_el,k_lat],
             [C_el,C_lat],rho,[G_Pt])
sim.addSubstrate("Si")
sim.final_time = 7*u.ps
[x,t,T] = sim.run()
```

To compare the simulated data with the experimental case, we computed the exponentially weighted temperature data $T^{E,L}(x,t)$ with respect to the depth of the material. This mimics the effect of limited optical penetration depth of the probe laser. In Fig. 5, we show a comparison between a simulation with and without the silicon substrate.

The results, depicted in Fig. 5 show that the multilayer solution obtained from the NTMpy software is able to reproduce the experimental data to a very good degree. Furthermore, one can clearly see that disregarding the substrate in the simulation, which is the simplest way to solve the two-temperature model, leads to a considerably different results. In this case, a four times larger

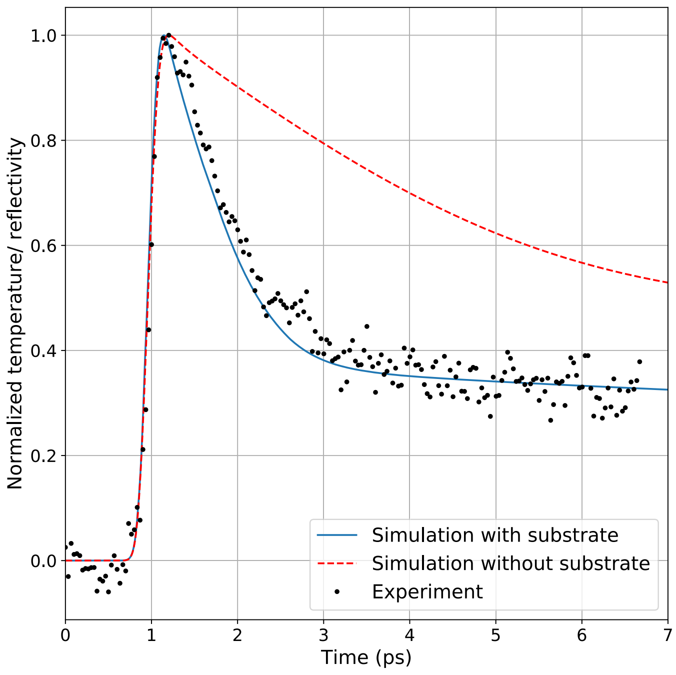

Fig. 5. Experimental data of a reflectivity measurement (symbols) compared to the simulated temperature dynamics of a platinum thin film on a silicon substrate (solid line) and free-standing (dashed line). A linear relation between the change in temperature and change in reflectivity is assumed, and they are normalized to the maximum change. The simulation parameters for the materials are taken from Table 1.

electron-lattice coupling constant from $G = 2.5 \cdot 10^{17}$ $(\mathrm{Wm^{-3}\,K^{-1}})$ to $G \approx 11 \cdot 10^{17}$ $(\mathrm{Wm^{-3}\,K^{-1}})$ would be needed to reestablish agreement with the experiment.

5. Conclusion

We implemented NTMpy, an open source Python based software package for solving coupled parabolic differential equations in one dimension. Along with the mathematical background of the algorithm, we introduced the structure and the work flow of the program. The object oriented way in which the program is designed gives the user the freedom to run tailor-made simulations, without having to worry about coding details, which are automatized or ready-made. This helps to focus more on the output and analysis of the fundamental dynamics. Together with the visualization class, the software can not only be used by researchers to investigate relaxation and diffusion dynamics, but also for educational purpose. Finally, a comparison of the output of NTMpy with the one of both commercial software and also of new experimental data, demonstrated the numerical reliability of the software, and its ability to produce physically realistic results.

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.

Acknowledgements

L.A. acknowledges support from the Swedish Research Council (VR), Grant: 2018-04611. V.U. and S.B. acknowledge support from the European Research Council, Starting Grant 715452 “MAGNETIC-SPEED-LIMIT”.

References

[1] B.L.K.S.I. Anisimov, T.L. Perel’man, Sov. Phys. JETP 39 (February 1974) 375–377.
[2] E. Beaurepaire, J.-C. Merle, A. Daunois, J.-Y. Bigot, Phys. Rev. Lett. 76 (May 1996) 4250–4253, https://doi.org/10.1103/PhysRevLett.76.4250.
[3] A.A. Maznev, J.A. Johnson, K.A. Nelson, J. Appl. Phys. 109 (7) (2011) 073517, https://doi.org/10.1063/1.3569731.
[4] B. Koopmans, G. Malinowski, F. Dalla Longa, D. Steiauf, M. Fähnle, T. Roth, M. Cinchetti, M. Aeschlimann, Nat. Mater. 9 (3) (2010) 259.
[5] D. Schick, A. Bojahr, M. Herzog, R. Shayduk, C. Von Korff Schmising, M. Bargheer, Comput. Phys. Commun. 185 (2) (2014) 651–660, https://doi.org/10.1016/j.cpc.2013.10.009.
[6] L. Alber, V. Scalera, V. Unikandanunni, S. Bonetti, udcm-su/ntmpy: Ntmpy-package1.1.1, https://doi.org/10.5281/ZENODO.3553742, https://zenodo.org/record/3553742, 2019.
[7] J. Hohlfeld, S.-S. Wellershoff, J. Güdde, U. Conrad, V. Jähnke, E. Matthias, Chem. Phys. 251 (2000) 237–258, https://doi.org/10.1016/S0301-0104(99)00330-4.
[8] P.M. Norris, A.P. Caffrey, R.J. Stevens, J.M. Klopf, J.T. McLeskey, A.N. Smith, Rev. Sci. Instrum. 74 (1) (2003) 400–406, https://doi.org/10.1063/1.1517187.
[9] Z. Lin, L.V. Zhigilei, V. Celli, Phys. Rev. B 77 (Feb 2008) 075133, https://doi.org/10.1103/PhysRevB.77.075133.
[10] COMSOL Multiphysics™, www.comsol.com, COMSOL AB, v.5.4.
[11] MATLAB, www.matlab.com, The MathWorks Inc., v. R2018b (9.5.0.944444).
[12] A.P. Caffrey, P.E. Hopkins, J.M. Klopf, P.M. Norris, Microscale Thermophys. Eng. 9 (4) (2005) 365–377, https://doi.org/10.1080/10893950500357970.
[13] A.D. Rakić, A.B. Djurišić, J.M. Elazar, M.L. Majewski, Appl. Opt. 37 (22) (Aug 1998) 5271–5283, https://doi.org/10.1364/AO.37.005271, http://ao.osa.org/abstract.cfm?URI=ao-37-22-5271.
[14] D.E. Aspnes, A.A. Studna, Phys. Rev. B 27 (Jan 1983) 985–1009, https://doi.org/10.1103/PhysRevB.27.985.
[15] D.E. Gray, I.E. Dayton, Am. J. Phys. 32 (5) (1964) 389, https://doi.org/10.1119/1.1970399.
[16] C.J. Glassbrenner, G.A. Slack, Phys. Rev. 134 (1964) A1058–A1069, https://doi.org/10.1103/PhysRev.134.A1058, taken at room temperature (May 1964).
[17] I. Lazanu, S. Lazanu, Astropart. Phys. 75 (2016) 44–54, https://doi.org/10.1016/j.astropartphys.2015.09.007, http://www.sciencedirect.com/science/article/pii/S0927650515001528.
[18] J. Hohlfeld, S.-S. Wellershoff, J. Güdde, U. Conrad, V. Jähnke, E. Matthias, Chem. Phys. 251 (1) (2000) 237–258, https://doi.org/10.1016/S0301-0104(99)00330-4, http://www.sciencedirect.com/science/article/pii/S0301010499003304.
[19] A. Chettah, H. Kucal, Z. Wang, M. Kac, A. Meftah, M. Toulemonde, in: Proceedings of the 23rd International Conference on Atomic Collisions in Solids, Nucl. Instrum. Methods Phys. Res., Sect. B, Beam Interact. Mater. Atoms 267 (16) (2009) 2719–2724, https://doi.org/10.1016/j.nimb.2009.05.063, http://www.sciencedirect.com/science/article/pii/S0168583X09006569.