This protocol describes the use of a simple 1D numerical model to evaluate a single-stage (magneto)caloric refrigerating device with static thermal switches. The model can be used to find appropriate values of parameters that lead to a significant refrigerating effect and COP of the device. The modeled device can comprise any type of static thermal switch in combination with any kind of magnetocaloric or electrocaloric material. Simulation parameters need to be set with care for acceptable computational time.

Publisher’s note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.
Protocol
Protocol to evaluate a (magneto)caloric device with static thermal switches using a 1D numerical model

Katja Vozel,1,2,3,4,* Katja Klinar,1,2 and Andrej Kitanovski1,2

1University of Ljubljana, Faculty of Mechanical Engineering, Askerceva 6, 1000 Ljubljana, Slovenia
2These authors contributed equally
3Technical contact
4Lead contact
*Correspondence: katja.vozel@fs.uni-lj.si
https://doi.org/10.1016/j.xpro.2022.101576

SUMMARY
This protocol describes the use of a simple 1D numerical model to evaluate a single-stage (magneto)caloric refrigerating device with static thermal switches. The model can be used to find appropriate values of parameters that lead to a significant refrigerating effect and COP of the device. The modeled device can comprise any type of static thermal switch in combination with any kind of magnetocaloric or electrocaloric material. Simulation parameters need to be set with care for acceptable computational time.
For complete details on the use and execution of this protocol, please refer to Klinar et al. (2022).

BEFORE YOU BEGIN
We have developed a simple 1D numerical model for simulating a single-stage (magneto)caloric refrigerating device that has static thermal switches (where static refers to those switches whose effective thermal conductivity changes during the on and off states without any moving component). The model serves as a basis for understanding and searching for the optimal parameters for the operation of such a device.
Such a (magneto)caloric device consists of a (magneto)caloric material, two thermal switches, a heat sink, and a heat source. This protocol provides the instructions on how to use the numerical model with either already-provided data or user-provided data. In the first part, we list the required Python packages and required data (properties of all the components and the operating parameters of the device). This is followed by the description of the model (additional information can be found in (Klinar et al. (2022)). The second part of the protocol provides step-by-step instructions for running the simulation: description of the modules and input parameters. The code can be run in two ways: on a PC or on an HPC. This is followed by a presentation of the results, the limitations and the solutions to some problems that might occur.
Two examples of the code with calculated results are available at https://github.com/alahde-unilj/mc_switch, or https://zenodo.org/badge/latestdoi/468418161. In one example, the temperatures in the heat source and the heat sink do not fluctuate during one cycle in the quasi-steady state, while in the other example they do.

Installation of the Python packages and the mc_switch code
© Timing: <30 min
This section contains instructions on what software is needed for the protocol and how to get it.

1. Make sure you have Python 3.0 or higher installed.
2. If needed, also install the following modules: `math`, `numpy`, `os`, `time`, `multiprocessing`.
3. Download the `mc_switch.zip` from GitHub (mc_switch) or Zenodo (10.5281/zenodo.6628383) and extract it to the `mc_switch` folder. It contains the code and datasets used as examples in the paper (Klinar et al. 2022). Check that the folder contains all the modules listed in the Code preparation section, the data and the results folder.

**Data collection**

© Timing: 10 min (assuming data from the accompanying files are used)

This section contains instructions on what data are needed for the protocol.

**Note:** If the required data of some other (magneto)caloric material in the correct format are at your disposal, it takes 10 min to change the default files.

4. There is a text file in the directory `mc_switch/data` named `s_total_Gd.txt` that contains the data for the (magneto)caloric material (mcm) gadolinium. If you would like to use a different material, it might take some time to obtain the corresponding data.

**Note:** For magnetocaloric or electrocaloric materials, the data must contain a table of the total specific entropy values of the material with respect to the magnetic field density and temperature. The first row of the table includes the temperatures in K (kelvin) and the first column includes the magnetic fields in T (tesla) at which the total specific entropy values in J/kg (joules per kilogram) are given.

**Note:** In our example, we provided the total specific entropy for gadolinium for magnetic field densities between 0 and 2 T (with a step of 0.01 T) and temperatures between 265 and 326 K (with a step of 0.02 K). The specific-entropy data of gadolinium (m)cm in `mc_switch` were calculated using the mean-field theory (Kitanovski et al. 2015). The specific heat capacity of the (m)cm is calculated from the same data and depends on the temperature and the magnetic field density. Therefore, it is calculated at each node as a product of the node temperature and the derivative of the node entropy with respect to the temperature. The calculation is given in the module `makematrix.py`. Linear interpolation between given discrete values is used.

5. The `mc_switch` modules contain exemplary material properties and parameters. The device consists of the following components (Figure 1A): two static thermal switches, a (magneto)caloric material, a heat source, and a heat sink. Note that you need to define the components and provide your own data for each of them (thickness, density, specific heat capacity, thermal conductivity as well as total entropy for the (m)cm), so make sure you have all the data prepared.

**Model description**

© Timing: 10 min

This section describes the model used in the protocol in detail.

6. The `mc_switch` simulates the operation of a (magneto)caloric refrigerating device or a heat pump, shown in Figure 1A. This section describes the preparation of the numerical model for the
simulations: provide the properties of the device’s components, and the operating conditions of the (magneto)caloric refrigerating device or the heat pump (magnetic field values, magnetization time, frequency of switching).

a. The main part of the mc_switch directory consists of 11 Python files, of which two are the main scripts and one of them should be run to start the simulation, depending on whether the code is running on a PC (runcycles.py) or an HPC (runcycles_HPC.py), while the other nine are imported as modules into these two scripts. There are also two folders, one of which contains the (m)cm data and the other one is for storing the simulation results. Additional files are required for running the simulation on an HPC (more information about these files is given in the section Preparation of code and files for simulations).

The mc_switch makes it possible to run more (depending on the available CPU cores on the computer) simulations at the same time for different thicknesses of components ((m)cm, thermal switches, heat sink, and heat source) under the same operating conditions and with the same properties.

b. The numerical model is based on the implicitly discretized, transient heat-conduction equation (Fourier’s law), coupled with the (magneto)caloric effect. For the mc_switch, a single-stage, non-regenerative Brayton thermodynamic cycle is used, as shown in Figure 1B. It consists of four processes: 1) adiabatic temperature change inside the (m)cm due to

---

**Figure 1. Caloric embodiment with ferrofluidic thermal switches**

(A) (Magnetocaloric embodiment evaluated in a numerical model with the boundary conditions (constant heat flow on the left and convection on the right) and the coordinate system.

(B) Presentation of the single-stage, non-regenerative Brayton thermodynamic cycle with thermal switches 1 and 2. Figure reprinted with permission from Klinar et al. (2022).
the (magneto)caloric effect when the external (magnetic) field is turned on, 2) heat transfer during the high isofield while the external (magnetic) field is still on, 3) adiabatic temperature change inside the (m)cm due to (magneto)caloric effect when the external (magnetic) field is turned off, and 4) heat transfer in low isofield while the external (magnetic) field is still off.

c. Two thermal switches are used to control the heat flux between the heat source and the (m)cm (thermal switch 1) or the heat sink and the (m)cm (thermal switch 2). The heat flows through the switch during its “on” state, while no or very little heat flows through the switch during its “off state”. The operating frequency is calculated as the inverse of the time that is needed for one complete cycle (all four processes). Repeating the cycles should result in a temperature change between the heat source (cold part) and the heat sink (hot part) in a way that heat is transported from the cold to the hot side of the device.

The flowchart of the mc_switch code is presented in Figure 2.

d. The governing equation for the heat-transfer processes is thus:

$$\frac{\partial T}{\partial t} - \frac{k(T)}{C_p(F, T)\rho} \frac{\partial^2 T}{\partial x^2} = 0,$$

(Equation 1)

where $T$ stands for temperature, $t$ for time, $c_p$ for specific heat capacity, $F$ for trigger type (magnetic field, electric field, force or pressure), $\rho$ for density, and $x$ for spatial coordinate. The caloric effect is implemented by a temperature change for each node of the caloric material,

$$T_{\text{fi}} = T_{\text{in}} + \Delta T_{\text{ad, app}}(T_{\text{in}}, F_{\text{fi}}, F_{\text{in}}),$$

(Equation 2)

$$T_{\text{fi}} = T_{\text{in}} - \Delta T_{\text{ad, rem}}(T_{\text{fi}} + \Delta T_{\text{ad, app}}), F_{\text{fi}}, F_{\text{in}}),$$

(Equation 3)

where “fi” stands for the final value, “in” for the initial value, “ad, app” for the adiabatic external field application, and “ad, rem” for the adiabatic external field removal. For discretization, the finite-difference method is used with a finite number of control nodes for each device component, i.e.,;
\[
\frac{T_{i+1} - T_i}{\Delta t} - \frac{k(T)}{c_p(F,T)\rho} \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2} = 0, \tag{Equation 4}
\]

where index \( i \) stands for time discretization and index \( k \) stands for spatial discretization. On the boundaries, the chosen boundary conditions are used, for example, a constant incoming heat flux on the left and convection on the right. In each step, the equations are rewritten in a tridiagonal matrix using the coefficients \( a, b, c, z \) in the following order:

\[
\begin{bmatrix}
a_0 & b_0 & 0 & 0 & 0 \\
c_1 & a_1 & b_1 & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & c_{m-1} & b_{m-1} \\
0 & 0 & 0 & 0 & c_m \\
\end{bmatrix}
\begin{bmatrix}
T_0^{i+1} \\
T_1^{i+1} \\
\vdots \\
T_{m-1}^{i+1} \\
T_m^{i+1} \\
\end{bmatrix}
= \begin{bmatrix}
z_0 \\
z_1 \\
\vdots \\
z_{m-1} \\
z_m \\
\end{bmatrix}, \tag{Equation 5}
\]

and solved with the Thomas algorithm.

**Note:** Because the (de)magnetization process is considered adiabatic in our case (there is no heat transfer between any of the components during the process), this numerical model works only for cases where the duration of the (de)magnetization process is much shorter than the duration of the heat-transfer processes. The (de)magnetization process occurs as a step function of time at the point of total (magnetic) field change.

**Note:** In this original code, we use an example of a magnetocaloric device with mcm gadolinium.

**Note:** The numerical model \texttt{mc_switch} is validated with the numerical model \texttt{heatrapy} (Silva et al. (2019)). For more details see (Klinar et al. (2022)). The numerical model is not validated experimentally.

**Note:** For more details on the numerical modeling of (magneto)caloric devices see (Kitanovski et al. (2015); Nielsen et al. (2011); Silva et al. (2021)), or contact the corresponding author.

### KEY RESOURCES TABLE

| REAGENT or RESOURCE | SOURCE | IDENTIFIER |
|---------------------|--------|------------|
| Software and algorithms | Python 3 | Python Software Foundation [https://www.python.org/](https://www.python.org/) |
| NumPy package | NumPy project and community [https://numpy.org/](https://numpy.org/) |
| mc_switch Python Code | Klinar et al. (2022) | Zenodo: [https://zenodo.org/badge/latestdoi/468418161](https://zenodo.org/badge/latestdoi/468418161) |
| Other | |
| Materials data | N/A | [https://zenodo.org/badge/latestdoi/468418161](https://zenodo.org/badge/latestdoi/468418161) |
| Hardware: PC or HPC | N/A | N/A |

### STEP-BY-STEP METHOD DETAILS

**Code preparation for your (m)cm device**

**© Timing:** 1 h

This section defines Python modules and describes the corresponding free parameters for a particular device.
Note: The following is the description of the modules in the mc_switch folder.

runcycles.py and runcycles_hpc.py are scripts for setting the thicknesses of the components and running the simulation. They are presented in detail later in the text.

The preparedata.py module contains a function for importing and initializing the (m)cm data to be used in the simulations.

The simulation.py module contains a class named Simulation with all the attributes needed for the simulation. This module is presented in detail later in the text.

The cycle.py module contains a function that defines the processes inside the cycle: magnetization, demagnetization, and both heat-transfer processes. It also updates and returns the temperatures of the nodes after each process.

The magnetization.py module contains a function to calculate the adiabatic temperature change of the nodes inside the (m)cm based on the current temperature and the predicted magnetic field change. It updates and returns the temperatures of the nodes inside the (m)cm after magnetization (increased temperature) or demagnetization (decreased temperature) process.

The heat_transfer.py module contains functions that define the heat-transfer process and the return temperatures of all the nodes after the heat-transfer process.

The makematrix.py module contains the functions for building the system of equations for the process of heat transfer inside the device. It considers the locations of the components, the properties of the nodes inside the components, the thermal resistance between the components, the internal heat generation and the boundary conditions. The system of equations is arranged in a tridiagonal matrix and a vector of right-hand sides.

The myprint.py module contains the functions called by other modules for exporting different data (temperatures, properties, resulting temperatures, heat fluxes, etc.) to the console or in the form of output files.

The thomas.py module contains a function for solving the tridiagonal system of equations by Thomas algorithm (tridiagonal matrix algorithm). The function takes lists of the coefficients a, b, c (under-diagonal, diagonal and above-diagonal coefficients) and z (right-hand sides), calculated with the functions from the makematrix.py module, and returns a new, modified set of temperatures for all the nodes of the device after a heat-transfer step.

The magneticwork.py module contains a function for calculating the magnetic work from the entropy change for each node inside the (m)cm for the whole thermodynamic cycle. For the model to be thermodynamically accurate, the magnetic work of the (m)cm calculated in this way must be equal to the difference between the heat fluxes into and out of the device plus the eventual internal generated heat. Both values are exported at the end of the simulation in the quasi-steady state of the system to check the validity of the simulation.

1. Modify the code according to the device you want to simulate.
   a. In the simulation.py file, input the values of the attributes, listed in Table 1, according to your case (the thicknesses of components are set in the runcycles.py or in the runcycles_hpc.py script).
   b. Some exemplary values of the attributes are already given in the code and some recommended values are given in the comments. You can also refer to two exemplary simulations, available at GitHub/Zenodo, for which the results are already calculated.
   c. Note that any of the values assigned to the attributes listed in Table 1 are constant and do not change with temperature or (magnetic) field.
Table 1. Main attributes of the Simulation class in simulation.py

| Attribute                     | Type         | Meaning                                                      | Unit                      |
|-------------------------------|--------------|--------------------------------------------------------------|---------------------------|
| self.path                     | String       | The path to the results folder                               | /                         |
| self.st_data_filename         | String       | The path to the file containing (m)cm data                   | /                         |
| self.rho_source               | Float        | Density of heat source                                       | kg m⁻³                    |
| self.rho_sink                 | Float        | Density of heat sink                                         | kg m⁻³                    |
| self.rho_switch               | Float        | Density of thermal switches (both have the same thickness)  | kg m⁻³                    |
| self.rho_mcm                  | Float        | Density of (magneto)caloric material                         | kg m⁻³                    |
| self.cp_source                | Float        | Specific heat of heat source                                 | J kg⁻¹ K⁻¹                |
| self.cp_sink                  | Float        | Specific heat of thermal switch                              | J kg⁻¹ K⁻¹                |
| self.k_source                 | Float        | Thermal conductivity of heat source                          | W m⁻¹ K⁻¹                 |
| self.k_sink                   | Float        | Thermal conductivity of heat sink                            | W m⁻¹ K⁻¹                 |
| self.k_on                     | Float        | Effective thermal conductivity of thermal switch during on state | W m⁻¹ K⁻¹                 |
| self.k_off                    | Float        | Effective thermal conductivity of thermal switch during off state | W m⁻¹ K⁻¹                 |
| self.k_mcm                    | Float        | Thermal conductivity of (magneto)caloric material            | W m⁻¹ K⁻¹                 |
| self.q_left                   | Float        | Left bound. cond.: heat flux = cooling power                 | W m⁻²                     |
| self.h_right                  | Float        | Right bound. cond.: convection coefficient                   | W m⁻² K⁻¹                 |
| self.right_temp               | Float        | Right bound. cond.: temperature of ambient                   | K                         |
| self.switch_work              | Float        | Work input for thermal switch                                | W m⁻²                     |
| self.resist_12                | Float        | Thermal contact resist.: interface heat source - thermal switch 1 | m² K W⁻¹                  |
| self.resist_23                | Float        | Thermal contact resist.: interface thermal switch 1 – (m)cm  | m² K W⁻¹                  |
| self.resist_34                | Float        | Thermal contact resist.: interface (m)cm - thermal switch 2  | m² K W⁻¹                  |
| self.resist_45                | Float        | Thermal contact resist.: interface thermal switch 2 - heat source | m² K W⁻¹                  |
| self.generated_heat_on        | Float        | Internal heat gen. inside thermal switch during on state      | W m⁻³                     |
| self.generated_heat_off       | Float        | Internal heat gen. inside thermal switch during off state    | W m⁻³                     |
| self.low_field                | Float        | (Magnetic) field density when the field is off               | T                         |
| self.high_field               | Float        | (Magnetic) field density when the field is on                 | T                         |
| self.start_temperature        | Float        | Starting temperature in all nodes                             | K                         |
| self.t_mag                    | Float        | Duration of (de)magnetization                                 | s                         |
| self.freq                     | Float        | Operating frequency                                          | Hz                        |
| self.tolerance                | Float        | Internal tolerance                                          | K                         |
| self.end_tolerance            | Float        | End tolerance = quasi steady-state condition                 | K                         |
| self.N                        | Integer      | Minimum number of executed cycles                           | /                         |
| self.n0                       | Integer      | Number of nodes in heat source and heat sink                 | /                         |
| self.n1                       | Integer      | Number of nodes in each thermal switch                       | /                         |
| self.n2                       | Integer      | Number of nodes in a (m)cm                                   | /                         |
| self.t_transfer               | Float        | Duration of each heat-transfer process                       | s                         |
| self.time_step                | Float        | Time step for the heat-transfer process                      | s                         |
| self.print_cycle              | Integer      | Result (temperatures) printing frequency                     | /                         |

*When running the code on an HPC, the path must include the folder mc_switch (e.g., ‘mc_switch/results’), otherwise not (e.g., ‘results’).

Additional work input required for thermal switch operation (e.g., electric energy, mechanical energy). This value is only used for the COP (coefficient of performance, defined later in the text) calculation in runcycles.py.

Internal heat generation inside the thermal switches (e.g., due to Joule heating, chemical reactions). The internal heat generation can be different during the on and off state of the thermal switch. Ideally, these values are 0.

Internal tolerance helps to optimize the accuracy of the newly calculated temperatures during a time step of a heat transfer.

The ending tolerance helps determine whether the quasi-steady state is reached. When two consecutive sets of node temperatures, defined by the attribute check_list of the Simulation class, differ by less than this given tolerance, the simulation stops.

Must not be changed; calculated from other given values.

The time step refers only to the heat-transfer process. Magnetization and demagnetization are considered adiabatic processes and thus considered as step functions of time.
Δ CRITICAL: We strongly recommend that you only make changes to the simulation.py and runcycles.py or runcycles_hpc.py files, and leave the rest of the code unchanged, as any changes might lead to errors in the code.

Preparation of code and files for simulations

⏱ Timing: 1 h

This section contains instructions on how to prepare the code and accompanying files to run the simulations on a PC a or an HPC.

2. Prepare the data and scripts in accordance with what hardware you are using.
   a. When running the code on a PC:
      i. Insert your (m)cm data text (.txt) file into the data folder.
      ii. Open the runcycles.py script and set the thicknesses of the components of your device: sw_th (thickness of each switch), mcm_th (thickness of the (m)cm) and hex_th (thickness of the heat source and the heat sink). These three parameters are passed to the constructor of your Simulation object.
   b. When running the code on an HPC:
      i. Create a folder on the HPC, named, for example, my_simulations_folder, where you will place the downloaded and extracted mc_switch folder.
      ii. As with running on a PC, put the (m)cm data in the data folder.
      iii. Open the runcycles_hpc.py script and navigate to line 102, where the function mp_handler is defined. This function prepares the threads for separate simulations with different combinations of thicknesses of your device components. The module used to do that is called multiprocessing.

Note: The processes to be run are saved in a pool and then run simultaneously as separate threads. The whole process exits when all the processes from the pool are finished. In the mp_handler, define three lists: sw_ths, mcm_ths and hex_ths. Combinations of the values in these lists will define the different device structures. For example, if the sw_ths, mcm_ths and hex_ths contain two values each, the function will prepare eight (2^3) simulations, each one of them considering one of the eight combinations of device components’ thicknesses.

Note: The number of combinations should not exceed two times the number of CPU cores available for your simulations. Create three additional files in the my_simulations_folder: an empty text file named error.txt (this is error log), an empty text file named out.txt (here the list of simulations will form), job.sh file containing execution instructions for the HPC.

Below is an example of the job.sh file.

```bash
#!/bin/bash
#SBATCH --job-name=job # job name
#SBATCH -e error.txt # error file name
#SBATCH -o out.txt # output file name
#SBATCH --partition grid # choose HPC partition, grid is an example
#SBATCH --time=2:00:00 # reserved time
```
Note: The defined number of cores must not be less than half of the number of threads defined in your multiprocessing pool in the runcycles_hpc.py script. At the same time, the number of threads in your multiprocessing pool should not exceed twice the number of CPU cores of a node if you are working with one node.

△ CRITICAL: Follow the rules of the HPC provided by the HPC administrator (especially the reserved time, which partition to use, how many cores and nodes may be used, which Python version is available, etc.).

Running the simulation

♫ Timing: 11 h

This section contains instructions on how to run the simulations.

Note: The code execution time is given for one of the exemplary simulations, available at GitHub. The code was executed with a single thread on a PC with an Intel Core i7 processor. The execution time strongly depends on the chosen parameters, particularly the space and time discretization. It increases with number of nodes of the device and a decrease of the time step.

When all the parameters and files are prepared and loaded to a PC or an HPC, you can run the simulation.

3. Start the simulation by running the runcycles.py script when on a PC or the runcycles_hpc.py script when running on an HPC.
   a. The easiest way to run the script is by using an IDE like PyCharm, opening the mc_switch folder as a PyCharm project and running the runcycles.py script from within. But you can, of course, use any other method for running the script.
   b. If everything works, you should see the results files in your results folder and you can proceed to step 5 of this section (for the analysis of results).

Note: You can use the runcycles_hpc.py script on your PC, but you should be careful not to run more threads than your CPU allows.

4. The workload on a Linux HPC is usually distributed and organized with SLURM. In the command line, navigate inside the folder my_simulations_folder. To run the simulations, start your job script (for example job.sh) with the sbatch command:

```bash
sbatch job.sh
```

If the job is submitted successfully, a notification like this one will appear:

```bash
#SBATCH --nodes=1 # nodes requested
#SBATCH --tasks=1 # tasks requested
#SBATCH --cores=9 # cores requested
module load Python/3.8 # load module from available modules
srun --exclusive python mc_switch/runcycles_hpc.py # path to the script to be run
```
You can then check whether your job is active with the `squeue` command. If you want to cancel your job, you need its ID. For example, the command to cancel the job with ID 289373 is:

```
scancel 289373
```

Some potential problems are already accounted for in the code and will show immediately after submitting the job (see troubleshooting, problems 1, 2, and 3).

⚠ CRITICAL: Some combinations of device components’ thicknesses will lead to slow convergence to the quasi-steady state. Depending on the discretization in time and space, this might lead to very long execution times. To predict the execution times, run some test simulations (on a PC or an HPC). For example, for your set of device-part thicknesses, try running the simulation with less accuracy, i.e., larger tolerance (self.tolerance and self.end_tolerance) and just a few nodes. Then lower the tolerance and increase the number of nodes to see how the computational time changes. In this way, you can make an estimate of how much time the simulations will take.

Note: It is possible to stop simulations and rerun them later. In that case, the simulations that were already finished will not run again and the “ALREADY DONE” message will appear for each of them.

Note: On GitHub, there are two example codes with exemplary parameter values available. Running the script `runcycles.py` from the example folder will produce the results that are already generated in the results folder.

### Results analysis

© Timing: 2 h

This section contains an example of simulation results and their analysis.

**Figure 3** shows typical results files for one example inside the results folder.

5. Many files appear in the results folder:
   a. Two files form for every simulation (based on different thicknesses of components).
      i. The first number in the name denotes the thickness of each thermal switch, the second is the thickness of the (m)cm and the last is the thickness of both the heat source and heat sink.
      ii. The file with suffix `_temps` shows the temperatures for each node after the end of each completed thermodynamic cycle, while the other file shows the temperatures of all nodes after each process in the cycle, a list of all parameters at the beginning and a short summary at the end of the file.
   b. `Result.txt` contains a short summary of the quasi-steady-state operation of all simulations.
   c. `resultFinished.txt` is empty and appears only as a sign that all simulations inside this folder are finished. It appears only when running the `runcycles_hpc.py` script.

6. Very important: Check whether a simulation gives the correct results. One option is to compare the magnetic work with the difference between the input and output heat fluxes plus the eventual internal generated heat. These two can differ by a few percent (see troubleshooting, problem 4). Additionally, check the fluctuation of the temperature inside the heat sink and the heat source. If
the fluctuations are so large that the temperature rises above and falls below the ambient temperature at some point, there is no cooling effect.

a. Figure 4 shows exemplary results.
   i. Figures 4A and 4C show the temperature evolution of magnetocaloric material, heat sink, and heat source from the beginning of operation until the quasi steady-state is reached.
   ii. Figures 4B and 4D show the temperature profile in the magnetocaloric device of one complete cycle during the quasi-steady state operation.

b. In the quasi-steady state, the temperature of the magnetocaloric material fluctuates between 294.8 and 289.5 K (temperature difference of 5.3 K), which is the maximum achievable cooling potential.
   i. Because thermal switches 1 and 2 are not ideal (thermal conductivity during off state is greater than zero and thermal conductivity during on state is finite), temperature difference between the heat source and the heat sink is only 2 K.
   ii. Heat dissipation from the heat sink to the ambient is adequate, since the temperature of the heat source is slightly above the ambient. If the switches had better rectification ratios, the temperature of the heat source would approach 289.5 K.

The temperatures of the heat source and the heat sink are important refrigerating quality indicators.

i. Figures 4A and 4B show the case where the temperatures of the heat sink and the heat source are constant during quasi-steady state (approx. 293 and 291 K, respectively).
ii. Figures 4C and 4D show the different case where the temperatures of the heat sink and the heat source fluctuate even in the quasi-steady state operation: the temperatures of the heat sink fluctuate between 291 and 291.4 K, and 292.7 and 293 K, respectively. Such operation is not desired, because the temperature of the heat sink should be constantly above the ambient temperature to dissipate heat to the ambient.

d. Different magnetocaloric devices can be compared according to the established temperature span between the heat sink and the heat source.
   i. The higher the temperature span, the better the operation of the refrigerating device. The example in Figure 4 shows a case of zero cooling power, and therefore the highest achievable temperature span. If we increase the cooling power (add a thermal load to the heat source), the temperature span decreases. At some point, the temperature span will be zero: this corresponds to the maximum cooling power of the magnetocaloric device.
   ii. Usually, the cooling power is given per gram of magnetocaloric material, which enables further comparison with other devices and different magnetocaloric materials.
   The operation of the device can be evaluated also in terms of the COP, which is defined as: \[ \text{COP} = \frac{\text{cooling power}}{\text{heating power} - \text{generated heat} - \text{input work} - \text{cooling power}}. \]

The model also enables evaluation of the impact of thermal contact resistance and internal heat generation inside thermal switches on the cooling performance of the device. Increasing one or both parameters decreases the temperature span and consequently the cooling power. The model can be used to determine the maximum allowable values of thermal contact resistance and internal heat generation. It is important to be aware of these values prior to designing the experimental setup.

Further analysis of the results is presented in (Klinar et al. (2022)).
EXPECTED OUTCOMES

The modeling method described here allows you to analyze different static thermal switches implemented in a single-stage, non-regenerative (magneto)caloric device, for example, in terms of their COP or achievable temperature differences. The best cases can then be manufactured and experimentally tested. See (Klinar et al. (2022)) for information on the design of a regenerative device.

LIMITATIONS

Because the (de)magnetization process is considered adiabatic, this numerical model works correctly only for cases where the time for (de)magnetization is much shorter than the time for the heat-transfer processes.
The heat conduction is based on Fourier’s law of heat conduction, which only applies to bulk materials and does not consider phenomena occurring on the nanoscale, therefore, be cautious when using very small dimensions or fine discretizations.

**TROUBLESHOOTING**

**Problem 1**
When you run a simulation, you see the “ALREADY DONE” message (step 4 in running the simulation).

**Potential solution**
The simulation is either finished and no action is needed, or there are old results in the results folder with the same name that need to be deleted.

**Problem 2**
When you run the simulation, you see “Time step too large!” message (steps 3 and 4 in running the simulation).

**Potential solution**
This means that the time step is larger than the length of the time interval for each heat-transfer process (my_sim.t_transfer < my_sim.time_step). Reduce the time step in simulation.py.

**Problem 3**
An error message (like this one: “ValueError: Some errors were detected! Line #513 (got 13 columns instead of 102)”) appears when you try to rerun the simulations after you had to interrupt the execution (step 4 in running the simulation).

**Potential solution**
Before rerunning the simulations, delete all the lines that were not finished in the ..._temps.txt files.

**Problem 4**
At the end of the simulation, the magnetic work differs from the difference between the input and output heat fluxes plus the eventual internal generated heat by more than 5% (Step 6 of results analysis).

**Potential solution**
This means that the system has not reached the quasi-steady state yet or that the operation is not thermodynamically correct. If the steady state is not yet reached, try reducing the end tolerance for checking the steadiness in the simulation.py file. If, however, the steady state is reached, the reason for the discrepancy could be an inappropriate (most likely too large) discretization in time and/or space. Try adjusting the discretization in the simulation.py file.

**Problem 5**
In the quasi-steady state, all temperatures (including the temperatures of the heat source) are above ambient. There is no cooling effect (Step 6 of results analysis).

**Potential solution**
The cooling power of the device is not sufficient, so there is no cooling effect. Possible causes, besides inappropriate thicknesses, are: the operating frequency is too high; cooling power (left boundary condition) value is too large; the heat generated inside the thermal switches is too large; thermal contact resistances are too large; the magnetic (or other field types for other caloric technologies) field change is too small; the temperature range of the simulation does not fit the temperature of the highest caloric effect of the chosen caloric material; the caloric effect of the chosen caloric material is too small (which results in poor cooling power).
RESOURCE AVAILABILITY

Lead contact
Further information and requests for resources should be directed to and will be fulfilled by the lead contact, Katja Vozel (katja.vozel@fs.uni-lj.si).

Materials availability
This study did not generate new, unique materials.

Data and code availability
Original data have been deposited to Zenodo: https://zenodo.org/badge/latestdoi/468418161. Exemplary data have been deposited at GitHub as part of the mc_switch repository and is publicly available as of the date of publication. DOI of a version of record at Zenodo is listed in the key resources table.

All original code has been deposited at GitHub and is publicly available as of the date of publication. DOI of a version of record at Zenodo is listed in the key resources table.

Any additional information required to reanalyze the data reported in this paper is available from the lead contact upon request.

ACKNOWLEDGMENTS

The authors also acknowledge the financial support of the Slovenian Research Agency for the projects MagBoost: Magnetocaloric booster micro-heat pump for district heating system L2-2610, MHD-magcool: Novel MHD-thermal switch essential for nonconventional magnetic cooling system BI-DE/21-22-008, and the research core funding no. P2-0223. The authors gratefully acknowledge the HPC RIVR consortium for funding this research by providing computing resources of the HPC system MAISTER at the University of Maribor, Slovenia. K.K. acknowledges the financial support by the Slovenian Research Agency as part of the Young Researcher PhD program. K.K. also acknowledges the L’Oréal-UNESCO For Women in Science Programme for awarding her with the Slovenian 2022 fellowship.

AUTHOR CONTRIBUTIONS

Conceptualization, A.K., K.K.; software, K.V., K.K.; formal analysis, K.K., K.V.; resources, A.K.; writing – original draft, K.V., K.K.; supervision, A.K.; project administration, A.K.; funding acquisition, A.K.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

Kitanovski, A., Tusek, J., Tomc, U., Plaznik, U., Ozbolt, M., and Poredoš, A. (2015). Magnetocaloric Energy Conversion (Springer).

Klinar, K., Vozel, K., Swoboda, T., Sojer, T., Muñoz Rojo, M., and Kitanovski, A. (2022). Ferrofluidic thermal switch in a magnetocaloric device. iScience 25, 103779. https://doi.org/10.1016/j.isci.2022.103779.

Nielsen, K.K., Tusek, J., Engelbrecht, K., Schopfer, S., Kitanovski, A., Bahl, C.R.H., Smith, A., Pryds, N., and Poredoš, A. (2011). Review on numerical modeling of active magnetic regenerators for room temperature applications. Int. J. Refrig. 34, 603–616. https://doi.org/10.1016/j.ijrefrig.2010.12.026.

Silva, D.J., Amaral, J.S., and Amaral, V.S. (2019). Modeling and computing magnetocaloric systems using the Python framework heatrapy. Int. J. Refrig. 106, 278–282. https://doi.org/10.1016/j.ijrefrig.2019.06.014.

Silva, D.J., Ventura, J., and Araújo, J.P. (2021). Caloric devices: a review on numerical modeling and optimization strategies. Energy Resarch 45, 18498–18539. https://doi.org/10.1002/er.7023.