# A manual for instant computational tool for optothermal fluidics

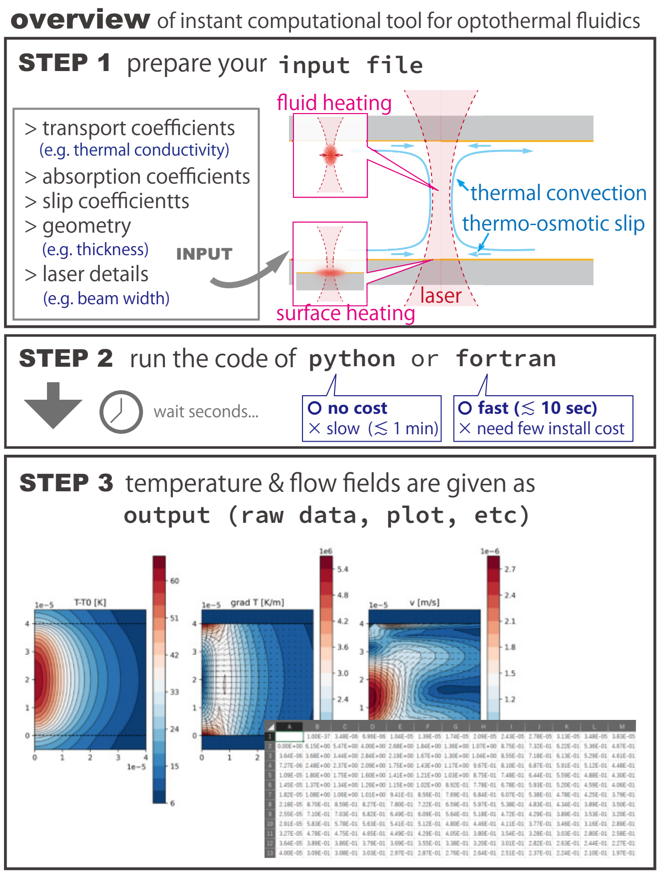
— *Supplemental Material on "Semianalytical model of optothermal fluidics in a confinement [1]"* —

Tetsuro Tsuji,[*] Shun Saito, and Satoshi Taguchi

*Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan*

(Dated: December 13, 2024)

This is a manual for a open-source software written either Python (`otf.py`) or Fortran (`otf.f90`). The software provides an instant computational tool for optothermal fluidics for situations described in Ref. [1]. This document and related source codes are subject to update. Latest versions will be available from the Kyoto University database (KURENAI) (see Ref. [2]).



## FEATURES OF CODES

Two choices are available:

**1. Python code**: otf.py

- A standard Python environment such as jupyter notebook or Spyder can be used. A single Python file `otf.py`, which is written in a naive manner, is all needed; no complicated install process is necessary.
- Plots of temperature and flow fields are generated. See examples case A and case B in Ref. [1] shown in Fig. S1.
- Python code is slower compared with a Fortran code (several tens of times slower).
- Use the Python code if you
  - want to compute only a specific case.
  - only need some coarse information such as the order of magnitude, the flow direction, etc.
  - do not want to pay costs and time.

**2. Fortran 90 code**: otf.f90

- Installation of a Fortran environment is necessary.
- Unix environment is expected. See Sec. B below for advanced options.
- Fortran code is faster compared with a Python program (several tens of times faster). Data in Ref. [1] are produced by the Fortran version.
- All the data are exported in ASCII format. You can use your familiar plotting tools.
- Preliminary plotting is supported using a software Tecplot 360 and its macro script.
- Use the Fortran code if you
  - want to compute systematically over a wide range of physical parameters.
  - want some high-resolution information over whole computational domain.

Both codes read an input file `00input-XXXXXX.txt` (sample file for case A in Ref. [1]). The input file includes physical and computational parameters; any six characters `XXXXXX` is a specific ID for input files. The prefix "`00input-`" is mandatory and the input file should be located in the same directory as the Python (or Fortran) code.

---

## STEP 1: PREPARE AN INPUT FILE

- Create your input file with a name, e.g., `00input-000000.txt`. Figure S2 is an easy guide to prepare the values in the sample file according to your setup. The default values of this sample input file represents the case A in Ref. [1].
- An excel sheet `input-generator.xlsm` [sample for a reference case in Fig. 8(a) of the main text] is available to produce the input files systematically for various parameter sets.

[*] tsuji.tetsuro.7x@kyoto-u.ac.jp; corresponding author

- The name of the parameters are same as those in the main paper [1]. However, some of them need additional description:
  - Lattice in $r$-$z$ space for contour plots is defined as Fig. S3(a). `Nr` (# of grid points in $r$ direction) and `Nz` (# of grid points in $z$ direction in the fluid) determine the spatial resolution of results. Same is true for `Nz_1` and `Nz_2`. A parameter `Aspect` determines the range of $r$ of the region of interest. That is, the size of the fluid domain to be displayed will be `Aspect*H` $\times$ `H`. Computational parameters `Nr`, `Aspect`, `Nz`, `Nz_1`, and `Nz_2` are **NOT** related to the accuracy of the results.
  - `s_max0` and `Nm` are the computational parameters related to accuracy. `s_max0` determines the upper limit of the integrals with respect to $s$ for the inverse Hankel transform. `Nm` is # of grid points in $s$ space. Therefore, ideally, large `s_max0` and `Nm` are preferable. (Automatic search option is available; See Sec. A below.)
  - Try the small values of `Nr`, `Nz`, `Nz_1`, `Nz_2`, and `Nm` as an initial trial. For instance, the computation in the Python code with `Nr=Nz=Nz_1=Nz_2=8` and `Nm=16` will end in 3 secs.

## STEP 2: RUN THE CODE

*Run the Python code*

- The Python code needs mpmath and gmpy2 packages for multiple-precision arithmetic. To install them, just try pip install as

```
pip install mpmath
pip install gmpy2
```

  You may also be required to update `pandas`.
- Put your input file in the same directory of your Python code.
- Change ID specified by `id=XXXXXX` in the code. `XXXXXX` is the same as ID in the input file name.
- Run the code and then you will soon get the results in a new directory `XXXXXX`.
- For advanced options, see Sec. A below. In particular, to change the range of the view, `truncate` and `magnify` options together with Fig. S3 is useful.

*Run Fortran code*

- Compile using a command ('`-o`' option specifies the name of execution file), e.g.,

```
ifort otf.f90 -o otf
```

  (Note: command for compile may change depending on the detail of the Fortran environment.)
- Put your input file in the same directory of your execution file `otf`. Download auxiliary files from here and put them in the same directory.
- Run the code using the command below and then you will get the results in a new directory `date_time-XXXXXX-arg`, where `arg` can be freely used as tags.

```
./otf XXXXXX arg
```

- For advanced option, see Sec. B below.

## STEP 3: CHECK THE RESULTS

*Outputs of the Python code*

- The files with the prefix "`OUTPUT-XXXXXX`" will be created in the directory `XXXXXX`.
- `dimensional_quantities.png` (sample png file for case B) is the image file of macroscopic quantities, i.e., temperature increase $\Delta T = T - T_0$ (left), temperature gradient $\nabla T$ (center), and flow velocity $\mathbf{v}$. (see, e.g., Fig. S1.)
- `macro-**.csv` is the 2D array of the values in the $r$-$z$ plane, where "row 1" and "column A" are $r$ and $z$ values, respectively. A tag `**` indicates below.
  - `dT`: temperature rise of fluid $T - T_0$ [K] (sample csv file for case B)
  - `dT_1`: temperature rise of solid 1 $T_1 - T_0$ [K]
  - `dT_2`: temperature rise of solid 2 $T_2 - T_0$ [K]
  - `Tr`: partial derivative $\partial T/\partial r$ [K/m]
  - `Tz`: partial derivative $\partial T/\partial z$ [K/m]
  - `vr`: flow velocity $v_r$ [m/s]
  - `vz`: flow velocity $v_z$ [m/s]
- `parameters.dat` (sample for case B) is the numerical values of dimensional and nondimensional parameters, which are computed based on the parameters given in the input file. Note that when the Reynolds number $Re$ is not small, then the linearization assumption may not be safe.

*Outputs of the Fortran code*

- All the output files are included in the directory `yyyymmdd_hhmmss-ID-tag` such as

```
\20240227_171932-000031-ref------------
```

  inside of which you find `dbg` directory. Two-dimensional macroscopic profiles are contained here. Each file has a header information (written for Tecplot use). The overview of the contents are
  - `mac-sp-all.dat`: nondimensional quantities
  - `mac-sp-all-dim.dat`: dimensional quantities
  - `mac-sp-log-maxval.dat`: maximum over $(r, z)$ plane of the solutions of the separated problems (i.e., $\tau^{(\mathrm{x})}$ and $\mathbf{u}^{(\mathrm{x,y})}$).

  - `mac-sp-tau-a0.dat`: $\tau^{(\mathrm{a})}$

    ...

  - `mac-sp-u-b2d2.dat`: $\mathbf{u}^{(\mathrm{b2,d2})}$

  In addition to temperature and flow fields, force $\mathbf{F}$ are also computed in the Fortran code.

- The portions of the integrand, i.e., $s\bar{\tau}^{(\mathrm{x})}$, $-s^2\bar{\tau}^{(\mathrm{x})}$, $s(\partial\bar{\tau}^{(\mathrm{x})}/\partial\tilde{z})$, $s\bar{u}_r^{(\mathrm{x,y})}$, and $s\bar{u}_z^{(\mathrm{x,y})}$ are stored in `integrad` directory (see also the inverse Hankel transform for semianalytical solution Eq. (21) in Ref. [1]). These files can be used to check the behavior of the integrand.

- `mac` directory contains the maximum values over $(r, z)$ plane of macroscopic quantities. Each file contains the physical and numerical parameters. The same files are copied in a directory `../summary`; these are useful when comparing the results over various parameters. For instance, Figs. 8(a), 9(a), and 10(a) in Ref. [1] are created using these dataset.

## ADVANCED SETTINGS

### A. *Python code*

Advanced options are available in the header zone in the code (see the documentation in the code for detailed description):

- `display_macro_val_on_console`: Use if you need outputs only in a console. No file will be created.
- `manual_enter_parameter`: Use if you enter parameters directly without preparing the input file.
- `truncate`: Use when omitting the display of solid parts away from the fluid part (see Fig. S3(b)).
- `magnify`: Use when zooming up a specified region in a fluid domain (see Fig. S3(c)).
- `show_nd`: Use when you need to see the nondimensional quantities of separated problems (e.g., $\tau^{(\mathrm{a})}$ and $\mathbf{u}^{(\mathrm{a,c})}$ in Ref. [1]).
- `search_s_max` & `search_Nm`: Use when the automatic search of nice values of `s_max0` or `Nm` is necessary. This option takes a bit of additional computational time. The recommended values of `s_max0` or `Nm` will be displayed on the console and `parameters.dat`.
- `hdl_layout`: Use when the `layout` option in Python is used. The function `plot_result()` controls the layout of the contour plots. Adjust as you like the local variables `fig` and `axsd` there if you need the well-ordered figure for publication.
- `hdl_aspect`: Use when you lock the aspect ratio of the contour plots.
- `ct_lv` & `clb_shk`: Some contour settings.
- `power`: A parameter that controls the density of the grid points of $s$ space in the inverse Hankel transform. Try to change when you want to decrease `Nm` keeping the accuracy.

### B. *Fortran code*

- OS is expected to be Unix. However, since this limitation comes only from the file manipulation in the code, the users will be able to modify easily to accommodate with your environment. For non-Unix users, rewrite the sentences in the code starting with "`call system(...)`" according to your operating system.

- Some preliminary plotting macros for Tecplot are available. Opening these files with Tecplot leads to comprehensive display of

  - `mac-sp_all-summed-vis.mcr`: dimensional and nondimensional macroscopic quantities.
  - `mac-sp_each-solution_***.lay`: separated macroscopic quantities $\tau^{(\mathrm{x})}$ and $\mathbf{u}^{(\mathrm{x,y})}$.
  - `integrand.mcr` the part of integrand as a function of $s$ (i.e., the data in `integrand` directory) for various values of $\tilde{z}$. (This file is used only for debugging and the display setting is specialized to the case with `Nz=20`.)

- Some convenient scripts: `00run.sh` can be used to run all the input files in the working directory. Drag and drop `00export_allframes.mcr` to the working Tecplot window creates the eps and png files.

### C. *Technical comments*

Some trouble shooting is introduced as follows.

- If you get `FileNotFoundError` in the Python code, check whether your current directory displayed in the console is an intended one. (Use a command `os.chdir('C:\Users\...')` to change the working directory.)

- Choosing large values of $H$, $H_1$, $H_2$ and/or small $w_0$ results in the appearance of large numbers in the exponential, cosh, and sinh functions. To handle these large numbers, the Python code employs the floating-point arithmetic with arbitrary precision and the Fortran code uses the quadruple precision. Nonetheless, putting small $w_0$ and large $H$ (e.g., $w_0 = 1$ µm and $H = 1$ mm) may result in error. An ad hoc approach for such cases is to use smaller $H$. when $w_0$ is small, the value of $H$ is not really effective on the temperature and flow fields near the laser focus. Therefore, the use of smaller $H$ is not harmful as long as one is interested in the behavior near the laser focus.

- The integrands in the inverse Hankel transform are assumed to vanish as $s \to 0$. This is confirmed *a posteriori* by observing the integrands for the settings presented in Ref. [1]. To implement this feature in the code, the integrand is forced to be zero at $s = $ `small`, where a constant `small` imitates the value $+0$ and is a small quantity such as $10^{-15}$. Without this forcing, the computation of the value of the integrand suffers from the serious cancellation of significant digits, leading to $0/0$ or `NaN`. When `Nm` is too large, very small $s$ may appear (e.g., $s = 10^{-10}$) and in this case the integrand cannot be obtained due to the error above. For such case, an ad hoc approach is to increase the number of multiple precision by increasing `mp.dps` (default is `mp.dps=200`) in the Python code. Unfortunately, the Fortran code is not compatible to higher precision.

- When large $r$ is concerned, `Nm` and `s_max` should be more carefully selected. For instance, see the temperature gradient of Fig. S1(b). Near $z = 0$ and $r = 8 \times 10^{-5}$ m, the careful observation tells you that the vectors there look nonphysical, and actually it is nonphysical. This is typical behavior when `Nm` is not large enough. An ad hoc approach is to just reduce the range of $r$, provided that you are not interested in the behavior at large $r$. This is not harmful at all because the range of $r$ is not related to the accuracy of the quantities at smaller $r$.

## VERSIONS & ENVIRONMENT

The computations are done under the following system environment:

- Fortran: Intel(R) Fortran Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 14.0.0.080 Build 20130728. The code was on Cent OS 6.4 -bash with Intel(R) Xeon(R) CPU E5-2650 v2 2.60GHz.
- Python: Python 3.10.13 on Spyder 5.4.3 with numpy 1.26.4, matplotlib 3.8.3, mpmath 1.3.0, pandas 2.2.1. The code was run on Windows 10 Pro with Intel(R) Core(TM) i7-9700K CPU 3.60GHz.
- Tecplot: Tecplot 360 EX 2018 R2, version 2018.2.1.93726

[1] T. Tsuji, S. Saito, and S. Taguchi, Semianalytical model of optothermal fluidics in a confinement, Phys. Rev. Fluids **9**, 124202 (2024).

[2] T. Tsuji, S. Saito, and S. Taguchi, Instant computational tool of optothermal fluidics, Kyoto University Research Information Repository (KURENAI) (2024), https://doi.org/10.57723/287529.

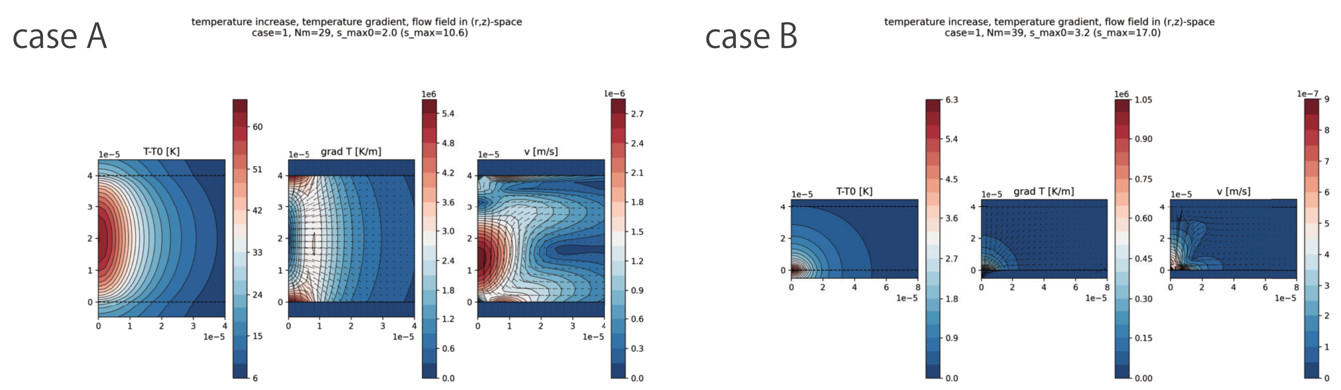


FIG. S1. Examples of the output from the Python code. Computation ends in several seconds.

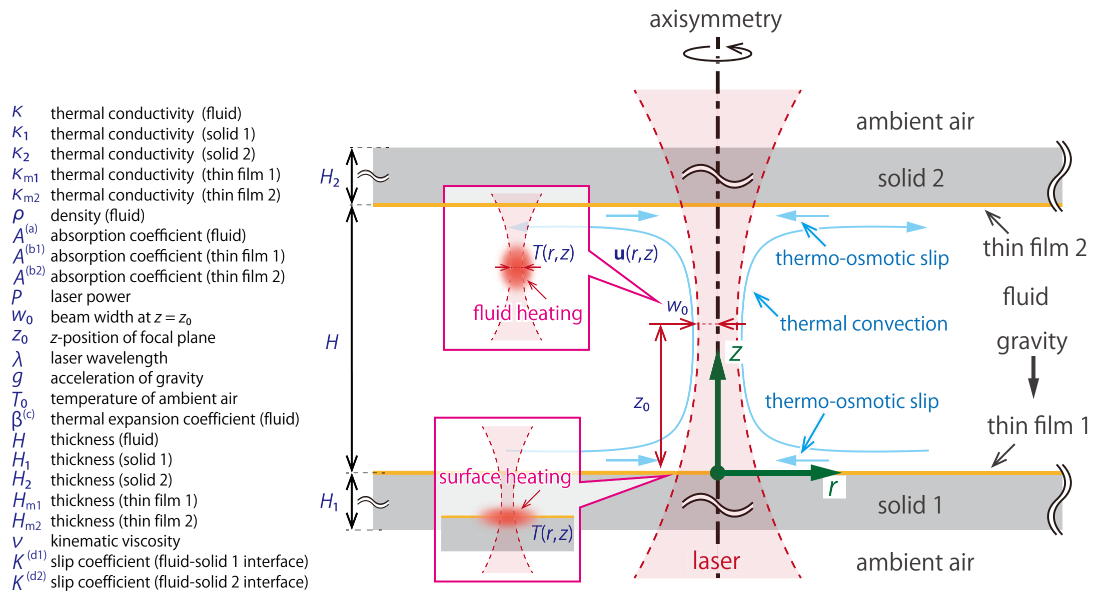

axisymmetry
$\kappa$ thermal conductivity (fluid)
$\kappa_1$ thermal conductivity (solid 1)
$\kappa_2$ thermal conductivity (solid 2)
$\kappa_{m1}$ thermal conductivity (thin film 1)
$\kappa_{m2}$ thermal conductivity (thin film 2)
$\rho$ density (fluid)
$A^{(a)}$ absorption coefficient (fluid)
$A^{(b1)}$ absorption coefficient (thin film 1)
$A^{(b2)}$ absorption coefficient (thin film 2)
$P$ laser power
$w_0$ beam width at $z = z_0$
$z_0$ $z$-position of focal plane
$\lambda$ laser wavelength
$g$ acceleration of gravity
$T_0$ temperature of ambient air
$\beta^{(c)}$ thermal expansion coefficient (fluid)
$H$ thickness (fluid)
$H_1$ thickness (solid 1)
$H_2$ thickness (solid 2)
$H_{m1}$ thickness (thin film 1)
$H_{m2}$ thickness (thin film 2)
$\nu$ kinematic viscosity
$K^{(d1)}$ slip coefficient (fluid-solid 1 interface)
$K^{(d2)}$ slip coefficient (fluid-solid 2 interface)
ambient air
solid 2
$H_2$
$T(r,z)$
$\mathbf{u}(r,z)$
fluid heating
thermo-osmotic slip
thin film 2
fluid
$w_0$
thermal convection
$H$
$z$
gravity
$z_0$
thermo-osmotic slip
thin film 1
surface heating
$r$
$H_1$
$T(r,z)$
solid 1
laser
ambient air

FIG. S2. Description of the contents of the input file.

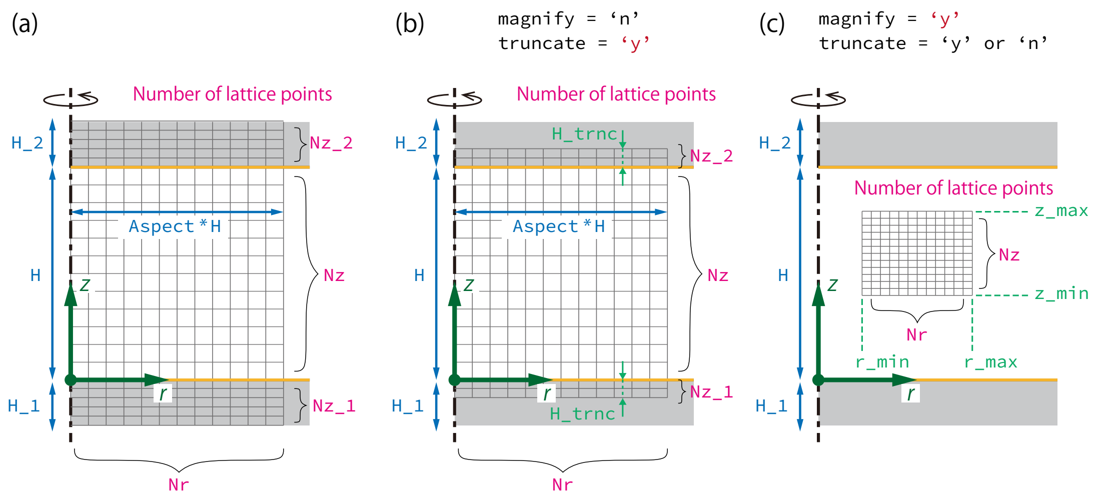


FIG. S3. Description of the computational parameters in the input file.