Finite difference method applied to the heat diffusion

In this practical we are interested in the heat diffusion. This phenomenon is modelled using finite difference numerical methods in a 1D case. For such a problem, although the solid medium is 3D, the temperature depends on only two variables: x, one of the three spatial coordinates and t, the time, and

$$\frac{\partial T(x,t)}{\partial t} = \kappa \frac{\partial^2 T(x,t)}{\partial x^2}. \tag{1}$$

The thermal diffusivity coefficient is denoted κ ($\mathrm{m^2\,s^{-1}}$). $\kappa = \dfrac{k}{\rho c}$, where k is the thermal conductivity ($\mathrm{W\,m^{-1}\,K^{-1}}$), ρ is the density ($\mathrm{kg\,m^{-3}}$) and c is the specific heat capacity ($\mathrm{J\,kg^{-1}\,K^{-1}}$).

This practical starts with a simple case where the temperature is maintained constant at the domain boundaries. This case allows to define the characteristic time for reaching the stationnary regime or to show that for such a regime the temperature gradient does not depend on κ. Some numerical issues are investigated such as CPU time and numerical divergence which enable to show that implicit scheme never diverges. Few additionnal problems are proposed in the event of all questions (1 to 6) have been previously addressed.
All codes/scripts have to be written using Python language and do not forget to insert a lot of comments in your scripts to explain the different steps.

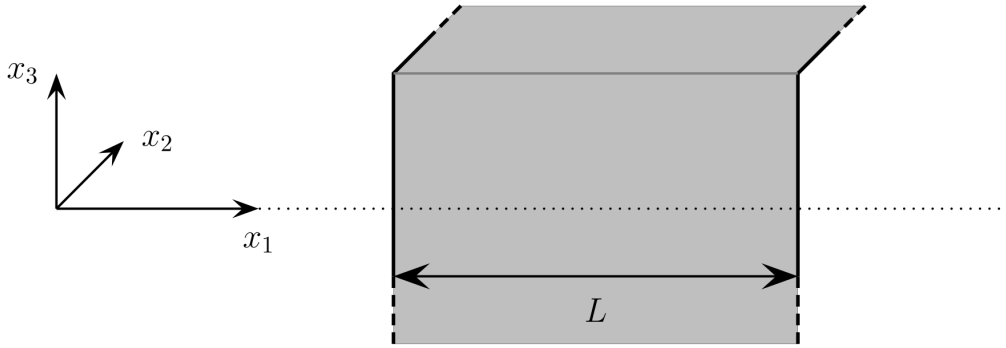

Figure 1: The medium is infinite in both x_2 and x_3 directions, and L represents the wall thickness following the x_1-axis (hereafter simply denoted as x).

1 Explicit scheme

The stationnary case addressed here is the heat diffusion within a solid medium where temperature is set and maintained constant at both edges, for instance an infinite wall following x_2 and x_3 directions and of thickness L along the x_1-axis (Fig. 1) For such a case, $x_1 = x$, and the two other dimensions can be ignored.

1.1. Start a new Python script named `heat_diff_fd-es-dd-1.py`. Such a name is proposed for a heat diffusion code since `fd` means "finite differences" while `es` means "explicit scheme" and `dd` signifies "Dirichlet-Dirichlet" for the boundary conditions.

1.2. As a first approach the following parameters are proposed:

```
0.005 # kappa (thermal diffusivity m^2/s)
1 # Length L (m)
10.0 # Temperature at x=0, T0 (°C)
20.0 # Temperature at x=L, TL (°C)
0.0 # Initial temperature (except 0 and L, °C)
0.25 # x-axis discretization length dx (m)
2.5 # Time discretization, dt (s)
0.00001 # precision for convergence
```

They can be directly written in the script (hard coded solution) or they can be written in an external input text file (named something like `heat_diff_fd-es-dd-0.in`, for instance) that have to be read during the execution of the Python script.

1.3. In order to tend toward the largest abstraction degree, define the amount of cells for the x-axis as a function of L and dx.

1.4. Write the lines composing the FD matrix by introducing the parameter A, as defined during the lecture.

1.5. Write a function called `prodMT` which performs the product of the FD matrix and the temperature vector (1D array) at a given time step. Inputs should be the matrix and the temperature vector and the ouptput is the new temperature profile.

1.6. Test this function by a simple call in the script.

1.7. Write the iteration process (loop) which enables to compute the temperature profile evolution, as a function of time, from the initial temperature condition (see item 1.2.). The exit condition must be coded within the loop using a convergence criterion. For example, it can be the sum of all differences, at the least-squares sense, for each cell, of two temperature vector consecutive configurations.

1.8. Once the convergence is reached the final configuration of the temperature profile should be a straightline connecting temperature values at $x = 0$ and $x = L$. Write the lines to produce such a plot and add labels on axes. Insert a legend box if several profiles are plotted on the same graph.

2 Stationnary case properties

While keeping the configuration as described in 1.2., perform a systematic exploration of κ, between 0.001 and 0.012 $\mathrm{m^2\,s^{-1}}$, every 0.001 $\mathrm{m^2\,s^{-1}}$, and check that

2.1. the final temperature profile remains the same, which means that the stationnary case does not depend on the thermal diffusivity;

2.2. the time to reach convergence is consistent with κ. Produce a plot of the convergence time as a function of κ.

3 Running time optimisation

3.1. Change the input values (item 1.2.) in order to perform a run with $L = 2$ m, $dx = 0.05$ m and $dt = 0.1$ s, and find the way to measure the computing time for such a numerical experiment. If the script is ran from a linux command window, the function `time` can do the job.

3.2. Modify the function `prodMT` in order to only sum the products of matrix elements that are not null and check the result in terms of CPU time.

3.3. By changing L values from 2 to 10 m, plot the CPU time gain between optimized and not-optimized `prodMT` function, as a function of L.

4 Transient regime

We now consider a 50 cm thick granite wall kept at temperature of $-5\,°C$. At a given time, the temperature on the two faces (normal to x-axis) is instantaneously increased to $20\,°C$ and we propose to study the transient regime between the initial and the final states.

4.1. Create a new Python script (`heat_diff_fd-es-dd-2.py`, for instance) which can be simply a copy of the previous one.

4.2. Create a new input file (`heat_diff_fd-es-dd-1.in`, for instance):

```
0.0000012  # kappa (thermal diffusivity m^2/s)
0.5 # Length L (m)
20.0 # Temperature at x=0, T0 (°C)
20.0 # Temperature at x=L, TL (°C)
-5.0 # Initial temperature (except 0 and L, °C)
0.01 # x-axis discretization length dx (m)
41 # Time discretization, dt (s)
0.0001 # precision for convergence
```

4.3. Check that the code runs properly and that the temperature profile converge toward a horizontal line at $T = 20\,°C$.

4.4. There are different ways to measure the diffusion characteristic time. We use here a simple approach telling that convergence is obtained as soon as the temperature reaches a value such that 90% of the initial temperature gap is filled. This temperature will be named `tc`, set its value using the input values (4.2.).

4.5. In our case the previous condition means that we have to monitor the temperature at the farest point of boundaries (*i. e.* middle f the wall) and to determine at which time step this temperature becomes greater than or equal to `tc`. The theoretical value for such a diffusion time is

$$\tau = \frac{L^2}{4\kappa}. \tag{2}$$

Using the input conditions (4.2.) compute the value of the variable `theotau` as defined in eq. (2) and compare it with the one discussed during lectures.

4.6. Write the lines that allow to compute this value, named `exptau`, under the hyptohesis (4.4.) and compare it with `theotau`. In practical, this can be done by stopping iterations as soon as the 90% condition is fulfilled and to convert the time iteration rank into duration using dt.

4.7. By modifying the wall thickness between 0.5 and 10 m, verify that experimental and theoretical values are close. Create a graph showing the difference between theoretical and experimental characteristic time diffusion values as a function of the thickness.

4.8. The characteristic distance of the heat front diffusing from the edge to the center, at time τ, is $l = \sqrt{\kappa\tau}$, check that this value is always close to $L/2$, using experimental values of τ.

5 Numerical divergence

5.1. Considering the input conditions as defined in (4.2.), go back to a wall thickness of 0.5 m and change the time step to $dt = 42$ s. Run and check what is happening.

5.2. Modify the script lines in order to detect any divergent feature which imply an iteration process break.

5.3. Print out the value of $\dfrac{dx^2}{\delta t}$ and compare it to κ.

Add a condition before starting time iteration that checks $\dfrac{\kappa\delta t}{dx^2} < \dfrac{1}{2}$, in order to stop the code and not become numerically unstable.

5.4. By defining a dx range, explore the dt values that define the divergence domain, as the orange domain in FIG. 2.

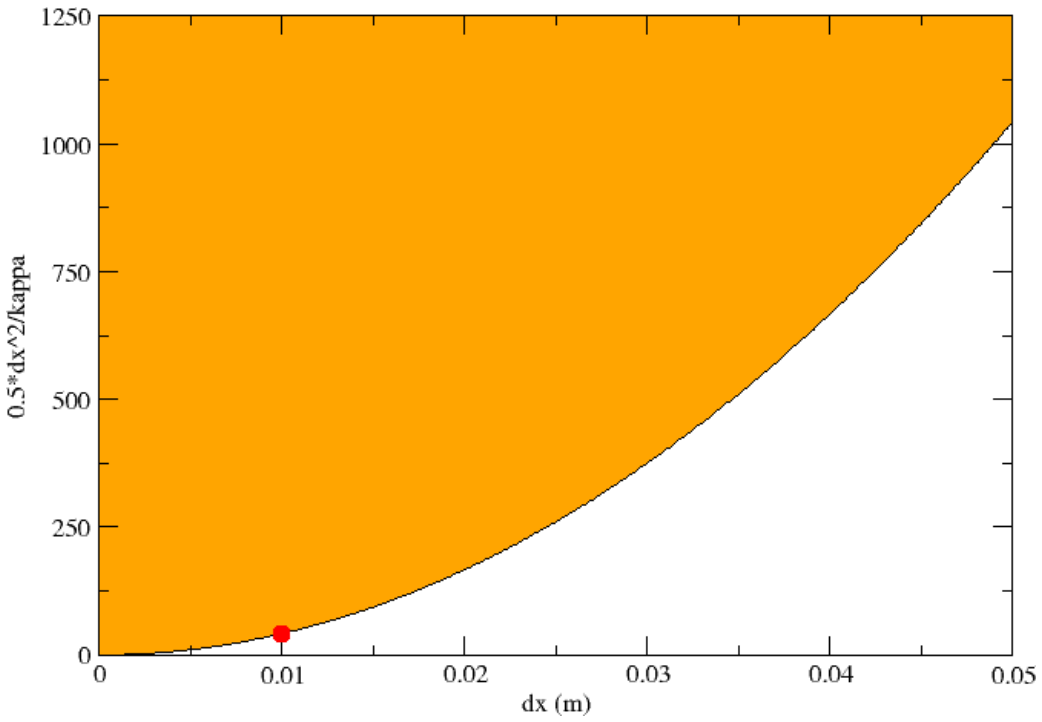

Figure 2: Divergence domain in orange as a function of dx.

6 Implicit scheme

Write a new script (named `heat_diff_fd-is-dd-1.py`) that solve the same partial derivative equation as in eq. (1), but with implicit scheme. Test that this code cannot diverge whatever the values of dt and dx.

7 Additional problems

7.1 Extra heat source in the medium

Using the case of heat propagation in a wall of infinite height and length and finite thickness (Fig. 1), write a new version of the script in order to model the presence of a vertical plate within the wall. The parameters and the boundary conditions are the same as in question 1. This extra heat source is located at $x = 0.4$ m and the constant temperature is $T = 16\,°\text{C}$.
It implies to define n_1 and n_2, the amount of points for each subdomain (located at left and right with respect to the extra heat source), and assuming that $n_1 + n_2 = n - 1$. The initial conditions have to be modified slightly. Create a figure to visualise the time evolution for such a problem.

7.2 Two-layer wall

Using the case of heat propagation in a wall of infinite height and length and finite thickness (Fig. 1), write a new version of the script in order to model the presence of a two-layer wall using these input parameters

```
1 # Length L (m)
10.0 # Temperature at x=0, T0 (°C)
20.0 # Temperature at x=L, TL (°C)
0.05 # x-axis discretization length dx (m)
0.25 # Time discretization, dt (s)
0.00001 # precision for convergence
```

In the x-axis direction, the wall is composed by $1/3$ of a medium defined by $\kappa = 0.005\ \text{m}^2\,\text{s}^{-1}$ and $2/3$ of another medium characterised by $\kappa = 0.0005\ \text{m}^2\,\text{s}^{-1}$.

7.3 Neumann boundary condition

The studied problem is the heat diffusion through a semi-infinite medium such as the Earth's crust. The temperature varies only as a function of time and z-axis, the depth (positive towards the Earth's centre).
Write a new Python script to model the heat diffusion by imposing a constant temperature at the surface ($z = 0$ km, $T = 5°\text{C}$) and a thermal gradient of $\dfrac{dT}{dz} = 30\,°\text{C/km}$ at the other domain boundary. At first approach use these input parameters

```
1.6e-6 # kappa, granite thermal diffusivity (m^2/s)
100 # Depth (m)
5.0 # Temperature at z=0, T0 (°C)
0.03 # Temperature gradient at max depth (C/m)
-100.0 # Initial temperature (except z=0)
1 # z-axis discretization length dz (m)
50000 # Time discretization, dt (s)
0.000001 # precision for convergence
```

Whatever the chosen scheme (implicit or implicit), there are two ways to treat this problem:

- the matrix M is kept unchanged and the Neumann boundary condition is imposed after the call to the function `prodMT` by modifying the last cell of the temperature vector, thanks to the uncentered derivate definition;
- the last row of the matrix M is modified to take into account the gradient boundary condition and also the last cell of the temperature vector after the call to the function `prodMT`.

The evolution of temperature through the first 100 metres of crustal depth should look something like Fig. 3.

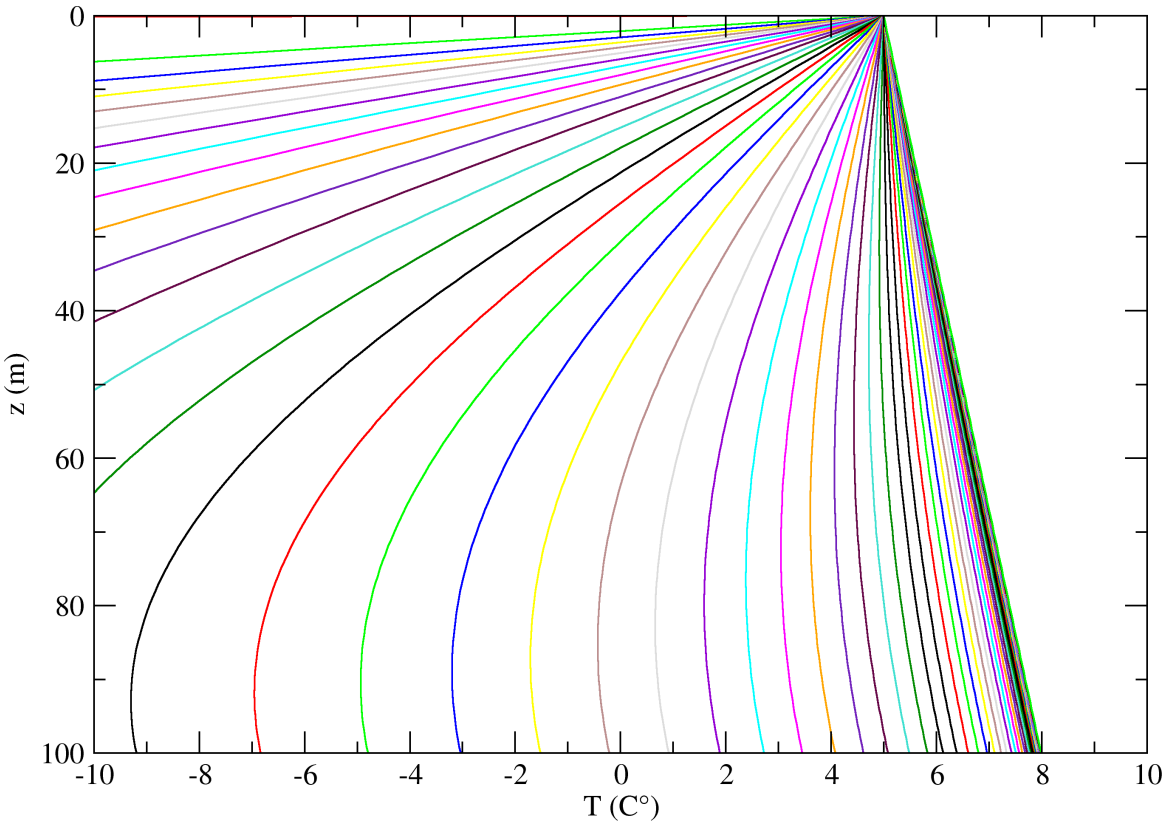

Figure 3: Heat diffusion through a granitic crust assuming a thermal gradient of 30 ° C/km.

To go further: add the possibility for the temperature at the surface ($z = 0$) to oscillate as a function of time with a characteristic period of 24 h. Define the depth at which these oscillations become negligible.