Elementary

- **Topic:** finite element method, heat equation
- **Task A:**
  1. Implement calculation of stiffness and mass matrices.
  2. Try running the simulation with different boundary temperatures.
  3. What should the temperature profile be like at a steady state? Does your solution match your expectation?
  4. Add some internal heating and see how that changes the result.
- **Task B:**
  1. Implement functions to calculate the propagator matrix and load vector.
  2. Implement the time advancing simulation step.
  3. Run time-dependent simulations with the same boundary temperatures and heating power as previously.
  4. The temperature must approach the limiting distribution you solved earlier. Check that it does.
- **Template:**
  - heat.py
- **Further reading:**
  - https://en.wikipedia.org/wiki/Heat_equation
  - https://en.wikipedia.org/wiki/finite_element_method

heat.py

heat.**animate**(*mesh*, *T_trajectory*) [source]

Animates the development of the temperature profile using `draw()`.

Parameters::
- **mesh** (*array*) – x-coordinates
- **T_trajectory** (*list*) – list of arrays of temperatures (at different times)

heat.**calculate_load**(*S*, *M*, *p*, *dt*) [source]

Calculates the load vector for a dynamic simulation.

Time is advanced according to the equation

$$T(t+\Delta t) = PT(t) + L.$$

This function calculates the vector L as

$$L = \left(M + \frac{1}{2}\Delta t S\right)^{-1} p\Delta t.$$

Parameters::
- **S** (*array*) – the stiffness matrix S
- **M** (*array*) – the mass matrix M
- **p** (*array*) – the power vector p
- **dt** (*float*) – time step Δt

Note

This function is incomplete!

Returns:: the load vector L
Return type:: array

heat.**calculate_mass_matrix**(*dx*, *n_mesh*, *printout=False*) [source]

Calculates and returns the mass matrix M.

M is a matrix whose elements are the overlap integrals of the FEM basis functions.

We define $\varphi_n(x)$ to be the basis function that is 1 at mesh point x_n and zero everywhere else. The elements of the stiffness matrix are then defined as

$$M_{i,j} = \int_0^L \varphi_i(x)\varphi_j(x)dx$$

except where boundary conditions override this definition.

Note

This function is incomplete!

Parameters::
- **dx** (*float*) – distance between mesh points
- **n_mesh** (*int*) – number of mesh points
- **printout** (*bool*) – If True, the matrix is printed on screen.

Returns:: the M matrix M
Return type:: array

heat.**calculate_power_vector**(*T_left*, *T_right*, *heat*, *M*, *printout=False*) [source]

Calculates the power vector p.

For a fixed boundary (temperature is kept constant):

- Replace the boundary node values of the heating power density vector ρ with the temperature values at the boundary. You get $\rho = [T(x_0), \rho(x_1), \rho(x_2), \ldots, \rho(x_N), T(x_{N+1})]$.
- Calculate the power vector as $p = M\rho$.

Parameters::
- **T_left** (*float*) – temperature at the left boundary
- **T_right** (*float*) – temperature at the right boundary
- **heat** (*array*) – heating power densities ρ
- **M** (*array*) – the mass matrix M
- **printout** (*bool*) – If True, the matrix is printed on screen.

Returns:: power vector p
Return type:: array

heat.**calculate_propagator**(*S*, *M*, *dt*) [source]

Calculates the propagator matrix for a dynamic simulation.

Time is advanced according to the equation

$$T(t+\Delta t) = PT(t) + L.$$

This function calculates the matrix P as

$$P = \left(M + \frac{1}{2}\Delta t S\right)^{-1}\left(M - \frac{1}{2}\Delta t S\right).$$

Note

This function is incomplete!

Parameters::
- **S** (*array*) – the stiffness matrix S
- **M** (*array*) – the mass matrix M
- **dt** (*float*) – time step Δt

Returns:: the propagator matrix P
Return type:: array

heat.**calculate_stiffness_matrix**(*dx*, *n_mesh*, *cond*, *printout=False*) [source]

Calculates and returns the stiffness matrix S.

S is a matrix whose elements are the overlap integrals of the derivatives of the FEM basis functions.

We define $\varphi_n(x)$ to be the basis function that is 1 at mesh point x_n and zero everywhere else. We also denote by $k_{n,m}$ the conductivity between mesh points x_n and x_m. The elements of the stiffness matrix are then defined as

$$S_{i,j} = k_{i,j}\int_0^L \varphi_i'(x)\varphi_j'(x)dx$$

except where boundary conditions override this definition.

Note

This function is incomplete!

Parameters::
- **dx** (*float*) – distance between mesh points
- **n_mesh** (*int*) – number of mesh points
- **cond** (*array*) – heat diffusivity k
- **printout** (*bool*) – If True, the matrix is printed on screen.

Returns:: the S matrix
Return type:: array

heat.**draw**(*frame*, *mesh*, *T_trajectory*, *Tlims*) [source]

Draws the temperature profile.

Parameters::
- **frame** (*int*) – index of the frame to be drawn
- **mesh** (*array*) – x-coordinates
- **T_trajectory** (*list*) – list of arrays of temperatures (at different times)
- **Tlims** (*list*) – the bottom and top values of the temperature axis: [Tmin, Tmax]

heat.**main**(*dx=0.1*, *n_mesh=11*, *T_left=300*, *T_right=400*, *heating_power=None*, *dt=0.001*, *recording_dt=0.01*, *simulation_time=0*) [source]

The main program.

If simulation_time is 0, only the equilibrium temperature distribution is calculated.

If simulation_time > 0, a dynamic simulation is run.

Parameters::
- **dx** (*float*) – mesh point separation
- **n_mesh** (*int*) – number of mesh points
- **T_left** (*float*) – temperature at the left boundary
- **T_right** (*float*) – temperateru at the right boundary
- **heating_power** (*array*) – heating power density at mesh points
- **dt** (*float*) – time step Δt
- **recording_dt** (*float*) – time between recorded temperature profiles
- **simulation_time** (*float*) – total time to simulate

heat.**run_simulation**(*temperatures*, *propagator*, *load*, *n_steps*) [source]

Runs a time dependent heat simulation.

Time is advanced according to the equation

$$T(t+\Delta t) = PT(t) + L.$$

The function does not change the original temperatures array. Instead, the final temperatures are returned as a new array.

Note

This function is incomplete!

Parameters::
- **temperatures** (*array*) – temperatures
- **propagator** (*array*) – the propagator matrix P
- **load** (*array*) – the load vector L
- **n_steps** (*int*) – simulation length in number of time steps

Returns:: temperatures at the end of the simulation
Return type:: array

heat.**show_history**(*mesh*, *T_trajectory*, *last=0*) [source]

Plots several temperature profiles, from different simulation times, in the same plot.

Parameters::
- **mesh** (*array*) – x-coordinates
- **T_trajectory** (*list*) – list of arrays of temperatures (at different times)
- **last** (*int*) – only plot this many temperature profiles from the end of the simulation (by default, all data is plotted)

heat.**show_temperature**(*mesh*, *temperature*) [source]

Plots the temperature profile.

Parameters::
- **mesh** (*array*) – x-coordinates
- **temperature** (*array*) – temperature at mesh points

heat.**solve_equilibrium**(*S*, *p*) [source]

Solves the equilibrium temperature profile.

The equilibrium satisfies $ST = p$ so the temperature is given by

$$T = S^{-1}p.$$

For fixed boundaries, the matrix S should always be non-singular and a solution can be found. For arbitrary boundary conditions this may not be true.

Parameters::
- **S** (*array*) – the stiffness matrix S
- **p** (*array*) – the power vector p

Returns:: equilibrium temperatures
Return type:: array