20/20

Very nice solution!

Solutions to Problem Set 7

Fundamentals of Simulation Methods 8 ECTS
Heidelberg University WiSe 20/21

Elias Olofsson
ub253@stud.uni-heidelberg.de

January 13, 2021

1 1-D Heat Diffusion Problem (10 pts)

For a 1-D block of radioactive material, radiating heat at the rate ϵ Joules per meter, the steady state temperature profile $T(x)$ obeys the 2^{nd} order PDE for heat diffusion as per

$$-D\frac{\partial^2 T(x)}{\partial x^2} = \epsilon, \quad \checkmark \tag{1}$$

where D is the heat diffusion coefficient of the material. At the domain boundaries, we assume a Dirichlet boundary condition, where both egdes at $x = \pm L$ are kept constant at a temperature T_0.

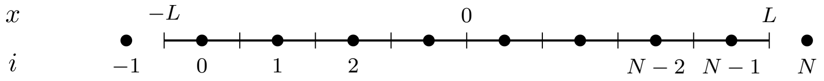
In order to write down a numeric form of Eq.(1), we divide the domain $x \in [-L, L]$ into N cells, and assign a grid point at the center of each cell. The temperature profile is then approximated by the temperature at the N grid points within the domain. We can approximate the 2^{nd} order spatial derivative in Eq.(1) at grid point i through the finite difference

$$\left(\frac{\partial^2 T}{\partial x^2}\right)_i \simeq \frac{T_{i+1} - 2T_i + T_{i-1}}{h^2}, \quad \checkmark \tag{2}$$

where we use the temperature in the neighbouring cells $i \pm 1$, and where $h = 2L/N$ is the grid spacing, which then leads to the numerical equation

$$T_{i+1} - 2T_i + T_{i-1} = -\frac{\epsilon}{D}h^2, \tag{3}$$

corresponding to Eq.(1). To make the numerical boundary value problem complete, we also however need to consider the boundary conditions, and how these should be incorporated into Eq.(3). One way of achieving this is to use a method of "ghost cells" where auxiliary grid points are introduced outside the domain, to force the temperature to a specific value at the boundary. In our specific 1-D case, we introduce one additional cell at each end of the domain, as depicted in Fig.(1). With these two cells, we can impose two constraints that specify the temperature at the boundaries to T_0, by



Sketch is also helpful

Figure 1 – The 1-D domain $x \in [-L, L]$ of the heat diffusion PDE in Eq.(1), and the corresponding discretization into N cells of equal size, with N grid points $i \in \{0, ..., N-1\}$ at the corresponding cell centers. Also shown is the two "ghost cells" outside the domain, indicated with grid points $i = -1$ and $i = N$, used for the implementation of the boundary conditions.

good! ?

requiring that

$$\frac{T_{i=-1} + T_{i=0}}{2} = T_0 \qquad \Longrightarrow \qquad T_{i=-1} = 2T_0 - T_{i=0} \tag{4}$$

$$\frac{T_{i=N-1} + T_{i=N}}{2} = T_0 \qquad \Longrightarrow \qquad T_{i=N} = 2T_0 - T_{i=N-1}. \tag{5}$$

Then, using these constraints to reformulate Eq.(3) for the cells $i = \{0, N-1\}$ at the edges of the domain, we obtain the discretization of the full boundary value problem as

$$T_{i+1} - 2T_i + T_{i-1} = -\frac{\epsilon}{D}h^2 \qquad \text{for} \quad i \in \{1, ..., N-2\} \tag{6}$$

$$T_{i+1} - 3T_i = -\frac{\epsilon}{D}h^2 - 2T_0 \qquad \text{for} \quad i = 0 \tag{7}$$

$$-3T_i + T_{i-1} = -\frac{\epsilon}{D}h^2 - 2T_0 \qquad \text{for} \quad i = N-1. \tag{8}$$

This set of equations can most favourably be written in matrix notation as

$$\boldsymbol{A}\boldsymbol{T} = \boldsymbol{b}, \tag{9}$$

where

$$\boldsymbol{T} = \begin{bmatrix} T_0 \\ T_1 \\ \vdots \\ T_N - 1 \end{bmatrix}, \qquad \boldsymbol{b} = -\frac{\epsilon}{D}h^2 - 2T_0 \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \tag{10}$$

and

$$\boldsymbol{A} = \begin{bmatrix} -3 & 1 & & & & 0 \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ 0 & & & & 1 & -3 \end{bmatrix}. \tag{11}$$

✓

For details on the implementations of the subroutines for a sparse matrix format, and matrix multiplication of such a matrix with an arbitrary vector, please reference the attached Juptyer Notebook `fsm_ex7.ipynb`. Here, I have created a sparse matrix format for tridiagonal matrices which only store the three main diagonals. More specifically, the matrix $\boldsymbol{A}$ above is in this format represented by

$$\boldsymbol{A}_{\text{sparse}} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 & 1 & 0 \\ -3 & -2 & -2 & \dots & -2 & -2 & -3 \\ 0 & 1 & 1 & \dots & 1 & 1 & 1 \end{bmatrix}, \tag{12}$$

where each row from top to bottom corresponds to the upper, middle and lower diagonal respectively.

Tests to verify the matrix multiplication of this sparse implementation are performed and passed. A comparison with the native `numpy` matrix multiplication through the @-operator determines that the resulting vector after a matrix multiplication has a element-wise maximum relative difference on the order of 10^{-16} between the two methods. ✓ For more details, please refer to the attached code. → that's then rounding / machine precision

Using the pseudo-algorithm given in the Wikipedia article for the forward-elimination backward-substitution method, also known as the Thomas algorithm [1], we implement another subroutine that solves a linear matrix equation on the form in Eq.(9) for an unknown vector $\boldsymbol{T}$. To verify correctness of the implemented matrix solver, we similarly to above compare the result of this method with the standard solver in `numpy.linalg.solve()`, and find that the solutions are identical for the two methods. ✓

Thus, applying our matrix solver to the boundary value problem specified in Eq.(9), where we also adopt the parameter values

$$D = 0.5, \quad \epsilon = 1, \quad T_0 = 1, \quad L = 1, \tag{13}$$

for $N = 100$ grid points, we obtain the steady state temperature profile shown in Fig.(2).

Multiplying the obtained numerical solution $\boldsymbol{T}$ with the matrix $\boldsymbol{A}_{\text{sparse}}$ and subtracting the constant vector $\boldsymbol{b}$, we obtain the residual of the solution. We can quickly note that it is very close, but not exactly equal to zero. This discrepancy I believe is due to the finite numerical precision inherent in floating point numbers, and it would likely decrease with increasing numerical precision.

For implementation details regarding the solver using the Jacobi iteration method, please refer to the attached code. At its core, the implementation uses the Jacobi iteration scheme

$$\mathbf{x}^{(n+1)} = \mathbf{D}^{-1}\mathbf{b} + \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{(n)}, \tag{14}$$

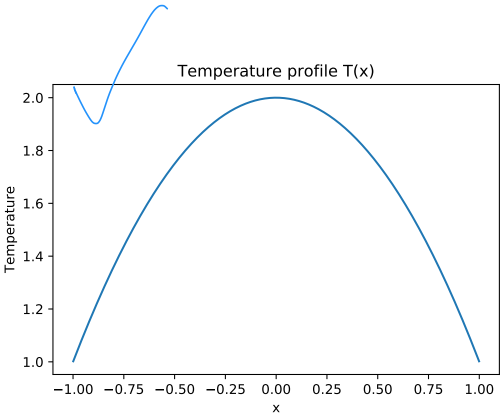


Figure 2 – Steady state solution of Eq.(1) describing the temperature profile in the 1-D radioactive block, numerically solved using the forward-elimination backward-substitution method and parameters specified in Eq.(13) with $N =$ 100 grid points.

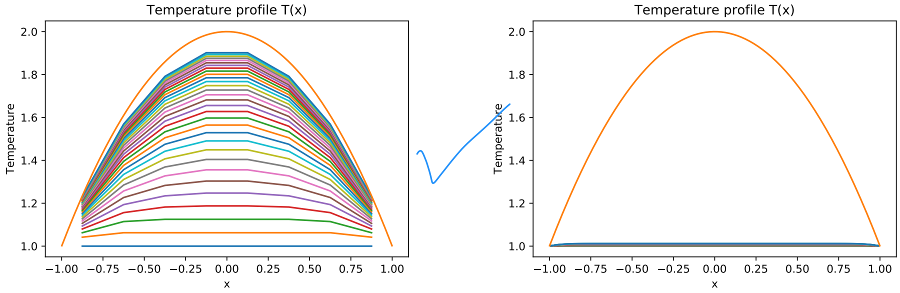
given in the FSM Lecture Script [2], where $\mathbf{x}^{(n+1)}$ and $\mathbf{x}^{(n)}$,$\mathbf{U}$, $\mathbf{D}$, $\mathbf{L}$ are the upper, middle and lower diagonal parts of matrix $\boldsymbol{A}$, respectively. Note however that $\mathbf{U}$ and $\mathbf{L}$ are the *negative* parts, such that

$$\boldsymbol{A} = \mathbf{D} - (\mathbf{L} + \mathbf{U}), \tag{15}$$

holds true. By iterating this scheme on some initial guess $\mathbf{x}^{(0)}$, we will slowly converge on the true solution $\mathbf{x}^*$ which satisfies $\boldsymbol{A}\mathbf{x}^* = \mathbf{b}$.

Implementing this iteration scheme by utilizing our matrix multiplication function, enables us to solve the same numerical boundary value problem from above in Eq.(9) using the parameters in Eq.(13). We solve for two cases, where $N = 8$ and $N = 100$ grid points, using an initial guess of $T(x) = 1$ and plot the solution after each step of 30 successive Jacobi iterations. The results can be seen in Fig.(3), where the "true" solution from the forward-elimination backward-substitution in Fig.(2) is over-plotted.

Comparing the two plots, we can see that the implementation of the Jaobi iteration is correct, since the solution is converging on the "true" temperature profile generated by the forward-elimination backward-substitution solver. We do however see a large difference in the rate of convergence between the two different grid resolutions, where the smaller resolution of $N = 8$ grid points are converging much faster than the case of $N = 100$. This highlights an unfavorable property of the Jacobi iteration method, that the convergence rate is inversely dependent on the number of grid points, which yields slower convergence for increasingly larger grids, rapidly adding more computational cost to obtain a numerical solution.



✓ ✓

(a) **(b)**

Figure 3 – Successive temperature profiles $T(x)$ following 30 Jacobi iterations from an inital guess of $T(x) = 1$. The true solution previously obtained using the forward-elimination backward-substitution method is over-plotted in orange for comparison. To the left and right, we have $N = 8$ and $N = 100$ grid points respectively.

10/10

2 1-D Multigrid Method (10 pts)

For this section, I realised I have to rewrite the numerical discretization in order to fit in with the specified grid sizes of 9, 5, 3 and 2 grid points. We then define a set of grids as in Fig.(4), where the number of grid points are dependent of a 'grid level' parameter n, as per

$$N = 2^n + 1, \tag{16}$$

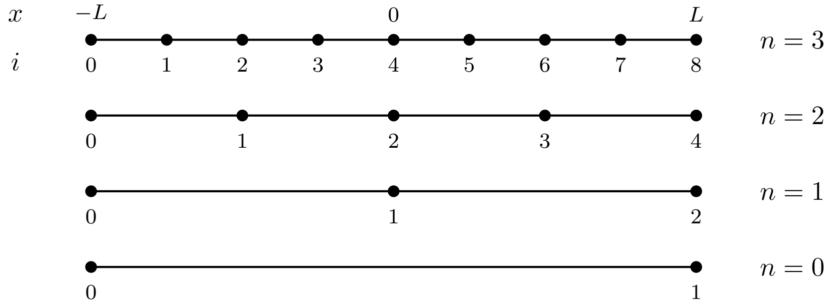
where N is the number of grid points. Specifically, we will here use grid levels $n = \{0, 1, 2, 3\}$, exactly as pictured in Fig.(4). Then, using the same approach as shown above in the previous exercise, an now instead implementing the boundary condition as letting the two grid points at the edges $i = \{0, N-1\}$ be fixed to the temperature T_0, we arrive at the linear system of equations

Good

$$\boldsymbol{AT} = \boldsymbol{b}, \tag{17}$$

where

$$\boldsymbol{T} = \begin{bmatrix} T_0 \\ T_1 \\ \vdots \\ T_N - 1 \end{bmatrix}, \qquad \boldsymbol{b} = -\frac{\epsilon}{D}h^2 \begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 0 \end{bmatrix} + T_0 \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \tag{18}$$



✓

Figure 4 – The 1-D domain $x \in [-L, L]$ of the heat diffusion PDE in Eq.(1), with a different discretization compared to Exercise 1. Here we let the grid points start precisely at the boundaries and follow a resolution doubling symmetry.

and

✓

$$\boldsymbol{A} = \begin{bmatrix} 1 & & & & & 0 \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ 0 & & & & & 1 \end{bmatrix}. \tag{19}$$

yes, this very odd reshaping is just annoying (but a good consistency test!)

Just to verify that this system of equations is equivalent to the Eq.(9), I solve this system using `numpy`'s built-in solver and plot the resulting temperature profile, please see the attached code `fsm_ex7.ipynb`. Sure enough, this gives the same reuslts and we can continue with the exercise.

We construct the restriction matrices $\mathbf{R}^{(3)}, \mathbf{R}^{(2)}, \mathbf{R}^{(1)}$ and prolongation matrices $\mathbf{P}^{(2)}, \mathbf{P}^{(1)}, \mathbf{P}^{(0)}$, as can be seen in the attached code. Since I was not entirely sure if these restriction and prolongation matrices are fully correct, I used two approaches to solve this exercise, differentiated by how the matrix $\boldsymbol{A}$ was calculated for lower grids than the finest resolution. First off, I used the approach of direct coarse grid approximation, i.e. I simply used the same discretization method but adjusted the number of gridpoints. With this method, one arrives matrices $\boldsymbol{A}^{(2)}, \boldsymbol{A}^{(1)}$ and $\boldsymbol{A}^{(0)}$ which are direct scales of $\boldsymbol{A}$ in Eq.(19), just with sizes $(5 \times 5), (3 \times 3)$, and (2×2). Secondly, I used the Galerkin coarse grid approximation, where $\boldsymbol{A}^{(3)}$ on the finest grid is used together with the restriction and prolongation matrices to calculate $\boldsymbol{A}$ for lower grid levels, as per

$$\boldsymbol{A}^{(2h)} = \mathbf{R}^{(h)} \boldsymbol{A}^{(h)} \mathbf{P}^{(h)}. \tag{20}$$

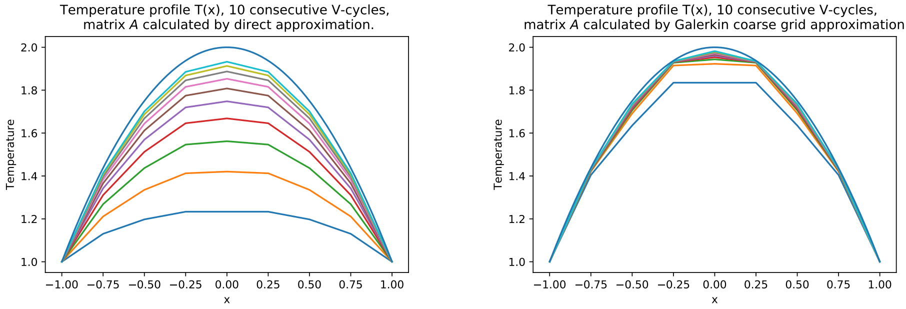
Matrices $\boldsymbol{A}$ produced with this method are for me not precisely proportional to the ones derived with the first approach, since elements close to the edges

(to check, you could also just functionally prolong or restrict, i.e. in a loop using $T_{j+1}^{h} = \frac{1}{2}[T_j^{2h} + T_{j+1}^{2h}]$ and not using matrices)

I think eq. 17 is correct (using odd number of points on the edges)

are slightly off. To see the specific matrices for this approach, please reference the the attached Jupyter notebook. I'm not sure if these discrepancies are due to an incorrect construction of the restriction and/or prolongation matrices, or if my discretization in Eq.(17) is not correct, or if this is inherent to the theory. Whatever the reason, I still seem to arrive at reasonable results anyway, using both of these approaches.

Implementing the recursive function for the multigrid V-cycle, and performing 10 consecutive V-cycles down to the lowest grid resolution for both approaches previously described, for an initial guess $T(x) = 1$, we plot the resulting temperature profiles in Fig.(5). Futhermore, the 'true' solution previ-



(a) **(b)**

Interesting, seems to converge faster

Figure 5 – Successive temperature profiles $T(x)$ following 10 multigrid V-cycles down to the lowest grid resolution, from an inital guess of $T(x) = 1$. The true solution previously obtained using the forward-elimination backward-substitution method is over-plotted in blue for comparison. To the left, the matrices $\boldsymbol{A}$ for lower grid resolution were calculated directly from discretization at the lower grids, while to the right these matrices were calculated using Galerkin coarse grid approximation, as detailed in Eq.(20).

ously obtained from the forward elimination, backward substitution method in the first exercise have been overplotted in the figure for comparison. Thus, we see that both solutions are approaching the static solution, and also that the rate of convergence are in both cases way faster than the normal Jacobi iterations previously performed in Exercise 1. We can also see that the second approach of obtaining matrices $\boldsymbol{A}$ for lower grids, using the Galerkin approximation, is converging a lot faster than the first approach. Just for my own investigations (not pictures here or in the code), I also tried 100 and 1000 iterations of V-cycles for both approaches, and both methods are really converging on the true solution. But as to why the Galerkin approach is that much faster I have no idea. → Sorry, me neither...

10/10

References

[1] Wikipedia. Tridiagonal matrix algorithm — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm`, 2021. [Online].

[2] Springel, V., Gräter, F. *Lecture Notes for MVComp1, Fundamentals of Simulation Methods.* 2020.