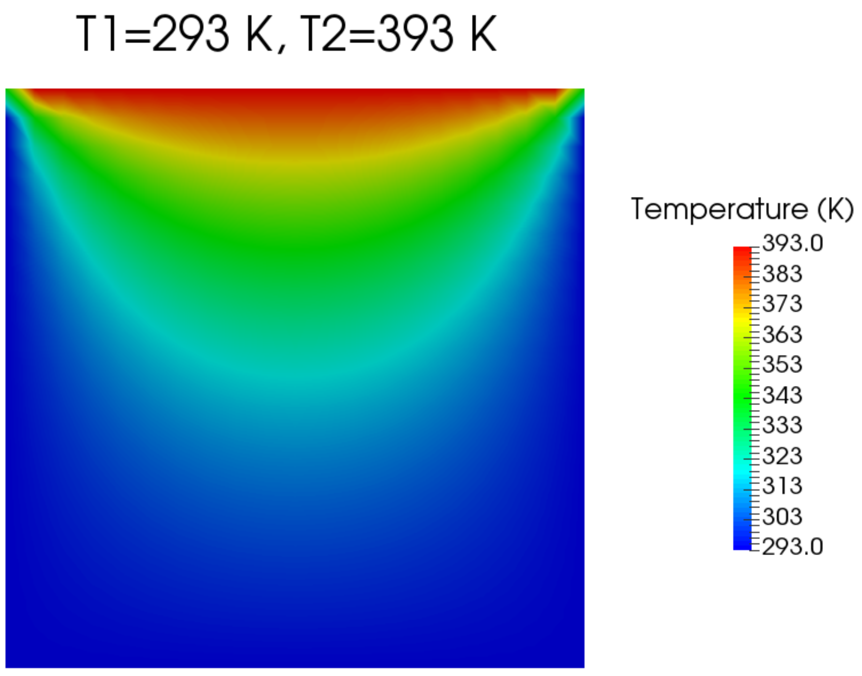
Computer Modelling Techniques

MMME3086 UNUK, 2023/24

Numerical Methods: Solution of the 1D steady diffusion equation

Author: Mirco Magnini
Office: Coates B100a
Email: mirco.magnini@nottingham.ac.uk



Contents

1 Partial Differential Equations

A Partial Differential Equation (PDE) is an equation that contains one or more partial derivatives of an unknown function that depends on at least two variables, as opposed to an Ordinary Differential Equation (ODE) where the unknown function depends on one variable only. The order of the highest derivative is the *order* of the PDE. We say that the PDE is *linear* if it is of the first degree in the unknown function and its partial derivatives, otherwise we call it *nonlinear*. Most of the PDE encountered in engineering are linear, second-order equations. For two independent variables, such equations can be expressed in the following general form:

$$A\frac{\partial^2\phi}{\partial x^2} + B\frac{\partial^2\phi}{\partial x\partial y} + C\frac{\partial^2\phi}{\partial y^2} + D = 0, \tag{1}$$

where A, B, and C are functions of x and y and D is a function of x, y, ϕ, $\partial\phi/\partial x$ and $\partial\phi/\partial y$. Note that x and y can identify both space and time. Depending on the discriminant $B^2 - 4AC$, the PDE can be classified into one of three categories:

- $B^2 - 4AC < 0 \rightarrow$ *Elliptic*, for example the *Laplace equation*:

  $$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0, \tag{2}$$

  which governs steady-state diffusion problems such as, for example, heat conduction, in two (or more) dimensions.

- $B^2 - 4AC = 0 \rightarrow$ *Parabolic*, for example the one-dimensional *heat equation*:

  $$\frac{\partial\phi}{\partial t} = \alpha\frac{\partial^2\phi}{\partial x^2}, \tag{3}$$

  which governs unsteady one-dimensional heat conduction problems; α is assumed to be constant.

- $B^2 - 4AC > 0 \rightarrow$ *Hyperbolic*, for example the one-dimensional *wave equation*:

  $$\frac{\partial^2\phi}{\partial^2 t} = c^2\frac{\partial^2\phi}{\partial x^2}, \tag{4}$$

  which governs the motion of a vibrating string.

The classification above is useful because each category relates to specific and distinct engineering problem contexts that demand special solution techniques. In this module, we will only work with elliptic equations, which are distinctive of steady-state diffusion problems in two or more dimensions (1D steady-state problems are simply governed by an ODE, not a PDE), and parabolic equations, which are distinctive of unsteady diffusion problems in one or more dimensions.

This lecture focuses on steady-state diffusion problems, which are introduced in Sec. 2. The principles behind the steps for obtaining a numerical solution are discussed in Sec. 3. In Sec. 4, we focus on the discretisation of the steady 1D heat equation (an ODE) using the finite-volume method, followed by a brief introduction to 2D and 3D cases (thus, PDEs) in Sec. 5, 6 and 7. Section 8 will present basic principles concerning with the solution of the linear system deriving from the discretisation of the equation which, later on, will be subject of a dedicated lecture (and notes). Finally, in Sec. 9 we provide some indicators to judge the accuracy of the numerical solution, and we conclude with a tutorial of the implementation of a numerical method in Matlab in Sec. 10.

Further reading on PDEs: Chapra and Canale [2], Part 8.

2 Diffusion problems in engineering

Diffusion problems identify physical phenomena where particles, energy, or other physical quantities are transferred inside a physical system due to diffusion only, i.e. when the bulk velocity in the direction of transmission is zero. The generic diffusion equation in steady-state conditions (independent of time), in a one-dimensional geometry, is an ODE and takes the following form:

$$\boxed{\frac{d}{dx}\left(\Gamma\frac{d\phi}{dx}\right) + S(x,\phi) = 0, \qquad 0 < x < L,} \tag{5}$$

where x is the coordinate direction along which the equation is solved and independent variable of the problem, ϕ is the variable being transported and dependent variable of the problem, Γ is a diffusion constant, L is the extension of the domain along x, and $S(x,\phi)$ represents a generic source term, that can be a generic function of x and ϕ. Note that the diffusion constant may be a function of the spatial variable (e.g. in a non-uniform medium), this is the reason why Γ is not taken off the external derivative. The first term at the left-hand side of Eq. (5) describes diffusion. Imagine, for instance, that ϕ is the concentration of a chemical. When concentration is low somewhere compared to the surrounding areas, the substance will diffuse in from the surroundings, so the concentration will increase. Conversely, if concentration is high compared to the surroundings, then the substance will diffuse out and the concentration will decrease (source: Wikipedia). The second term in Eq. (5) describes the creation or destruction of the quantity. For example, if ϕ is the concentration of a molecule, then S describes how the molecule can be created or destroyed by chemical reactions. S may be a function of ϕ and of other parameters.

Equation (5) needs boundary conditions at $x = 0$ and $x = L$ in order to be solved, for example Dirichlet:

$$\phi(x = 0) = a, \tag{6}$$

or Neumann boundary conditions:

$$\frac{d\phi}{dx}(x = 0) = b, \tag{7}$$

where a and b are two generic constants identifying the value of the function, or its gradient, at the boundary $x = 0$.

There are many engineering problems governed by diffusion equations, see the schematic in Fig. 2, for example heat conduction through a solid, the cross-stream diffusion of shear stress within a boundary layer, or the diffusion of a chemical species. In the case of heat conduction, the diffusion equation is usually written as:

$$\frac{d}{dx}\left(\lambda\frac{dT}{dx}\right) + S(x,T) = 0, \qquad 0 < x < L, \tag{8}$$

where T is the temperature in the solid and λ the thermal conductivity of the solid. $S(x,T)$ may be any energy source term within the solid, for example the heat generated by Joule effect due to electric current across a wire. In the case of the diffusion of shear stress within a fluid in motion, we can recall the Navier-Stokes equation for a laminar flow in a pipe, far from the entrance region, here written as

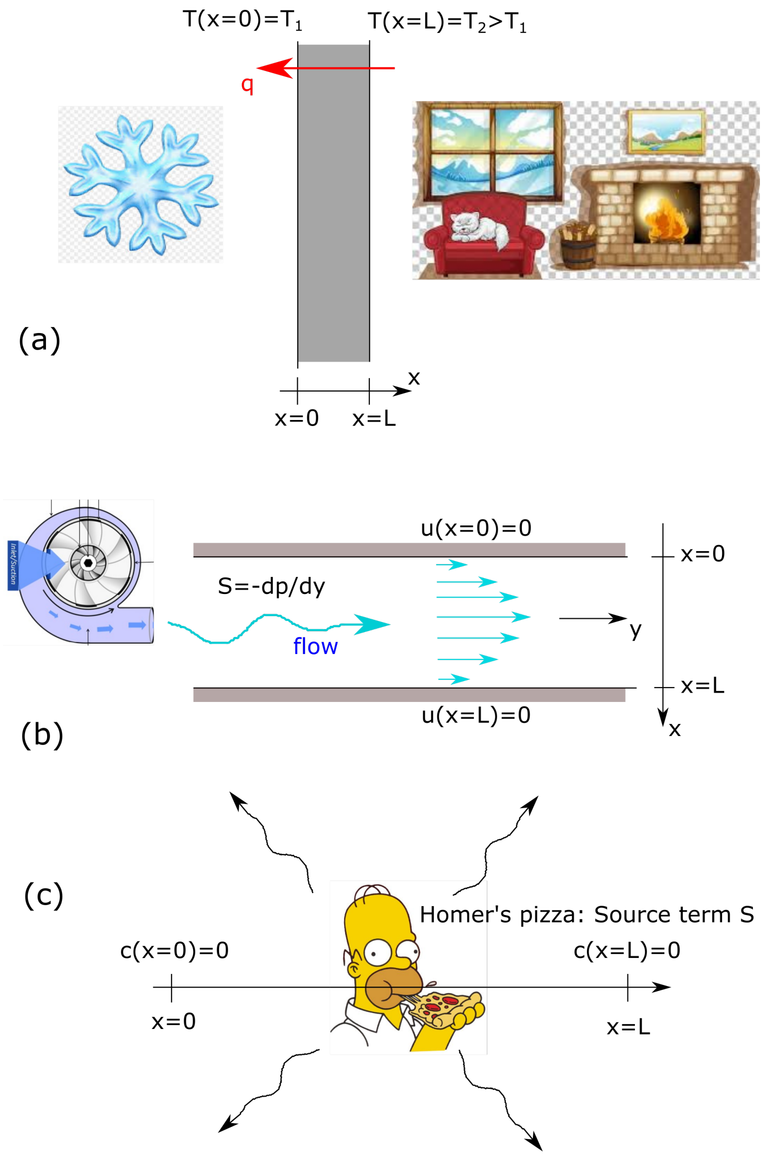


Figure 1: Sketch of traditional engineering problems governed by diffusion. (a) Diffusion of heat (q) across a wall due to temperature gradients. (b) Cross-stream diffusion of shear stress from the wall of a pipe, where the no-slip condition ($u = 0$) applies, through the fluid flowing in it; the streamwise pressure gradient dp/dy is due to the viscous shear stress in the fluid and acts a momentum source. (c) Diffusion of a chemical species c through the atmosphere, where Homer's pizza is the source, and the boundary conditions $c = 0$ at $x = 0$ and $x = L$ are purely arbitrary.

the streamwise component (here along y) of the momentum equation:

$$\frac{d}{dx}\left(\mu\frac{du}{dx}\right) - \frac{dp}{dy} = 0, \qquad 0 < x < L, \tag{9}$$

where x indicates the cross-stream direction, y the streamwise direction, u is the streamwise component of the velocity, μ is the dynamic viscosity of the fluid and dp/dy is the streamwise pressure gradient, with $dp/dy < 0$ (pressure always decreases along the channel), which acts as a momentum source; note that the fluid velocity component in the cross-stream direction x is zero (the streamlines are horizontal), and therefore the cross-stream transport of shear is still a diffusion problem. If you are not familiar with the Navier-Stokes equations in fluid mechanics, refer to the notes available at this link: MMME2047-Navier-Stokes. Likewise, the diffusion of a chemical species across a still environment, in 1D, is governed by the equation:

$$\frac{d}{dx}\left(D\frac{dc}{dx}\right) + S(x,c) = 0, \qquad 0 < x < L, \tag{10}$$

where c is the species concentration, D a species-diffusion constant, and $S(x,c)$ represents a generic source term for the species c, as example a chemical reaction. As you can see, all the equations above take the same form as Eq. (5), and therefore any solution method, be it analytical or numerical, applies to any of the three distinct problems.

Further reading: Chapra and Canale [2], Ch. 29; Burden and Douglas Faires [1], Ch. 12.

3 The discretisation concept

Let's now consider, without any loss of generality, the steady one-dimesional heat conduction equation that describes the transport of heat between two boundaries $x = 0$ and $x = L$, subject to two arbitrary Dirichlet boundary conditions:

$$\begin{aligned} \frac{d}{dx}\left(\lambda\frac{dT}{dx}\right) + S(x,T) &= 0, \qquad 0 < x < L, && \text{(11a)} \\ T(x=0) &= T_a, \qquad T(x=L) = T_b. && \text{(11b)} \end{aligned}$$

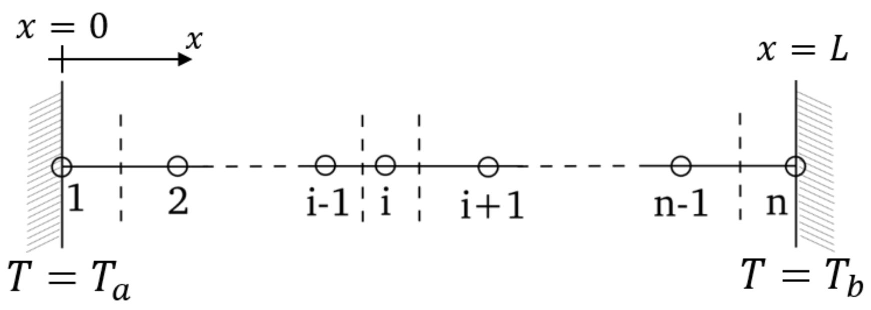


Figure 2: Sketch of the computational nodes (circles) used to discretise the one-dimensional domain across which the heat conduction equation is solved. The domain can be, for example, the grey wall separating the room with the fireplace from the outside in Fig. 2(a).

The situation is sketched in Fig. 2. An analytical method seeks for the continuous function $T(x)$ satisfying the differential problem for any location x within $[0, L]$. A numerical method seeks for an approximate solution $T_1, ..., T_n$ valid only at the discrete locations $x_1, ..., x_n$, which represent our *discretisation grid*, or *mesh*. Therefore, we have n unknowns $(T_1, ..., T_n)$ and to close the problem we need n equations. These n equations are derived from the differential equation Eq. (11) by means of appropriate discretisation methods (that we will see), and each $i - th$ equation describes how the temperature of the $i - th$ node depends on the discrete temperature values at the neighbours nodes. To differentiate it from the continuous equation Eq. (11), the $i - th$ node equation is called *discretisation equation*, because it involves the unknown values of T at chosen discrete grid points. In general, we prefer to solve systems of algebraic (i.e. polynomial) linear (where the unknowns appear with exponent one) equations of the kind:

$$\begin{cases} a_{1,1}T_1 + a_{1,2}T_2 + ... + a_{1,n}T_n = b_1 \\ \vdots \\ a_{n,1}T_1 + a_{n,2}T_2 + ... + a_{n,n}T_n = b_n \end{cases}$$

where the coefficients $a_{i,i}$ and b_i are constant. In compact notation, this means to solve the matricial problem $\mathbf{A} \times \mathbf{T} = \mathbf{B}$, where $\mathbf{A}$ is the $n \times n$ matrix of the coefficients, $\mathbf{T}$ is the $n \times 1$ vector of unknowns and $\mathbf{B}$ is the $n \times 1$ vector of known terms. This latter task, which is the most time-consuming aspect of the calculation, will be accomplished by a *linear solver*.

The specific discretisation method adopted, for example finite-difference, finite-volume or finite-element, identifies:

- How to derive the discretization equation from the initial differential equation;
- How to express derivatives as a function of the node values $T_1, ..., T_n$;
- How to deal with nonlinear terms.

However, the end point of the discretisation procedure, independently of the specific method chosen, is always to retrieve a system of linear algebraic equation as shown above. In this module, we will focus on the *finite-volume* method.

Further reading: Patankar [3], Ch. 3.

4 The finite-volume method – 1D

We aim to obtain the discretisation equation of the differential equation Eq. (11), for a generic node of the domain, using the finite-volume method. With the finite-volume (FV) method, we decompose the domain in n nonoverlapping control volumes (CV), see sketch in Fig. 3(a), with each CV being centered on the $i - th$ node. The discretisation equation is obtained by integrating the differential equation over each control volume. Each equation must be manipulated so that the only unknowns are the values of the temperature at the CV centres. The final system of equations is then obtained by assembling the n equations together. We will now see how this procedure is developed, starting with a generic CV internal to the domain, that is, not a boundary CV.

We begin with integrating Eq. (11) over a CV centred on a generic internal node:

$$\int_V \frac{d}{dx}\left(\lambda \frac{dT}{dx}\right) dV + \int_V S(x,T) dV = 0, \tag{12}$$

where V is the volume of the CV. We assume that the cross-section of the CV is uniform throughout the domain, its area denoted as A. Replacing dV with $dV = A\,dx$:

$$\int_w^e \frac{d}{dx}\left(\lambda \frac{dT}{dx}\right) A\,dx + \int_w^e S(x,T) A\,dx = 0, \tag{13}$$

where w and e denote the x-location of the west and east faces of the CV, see Fig. 3(b). The integrand of the first integral can then be simplified as follows:

$$\int_w^e d\left(\lambda \frac{dT}{dx}\right) A + \int_w^e S(x,T) A\,dx = 0. \tag{14}$$

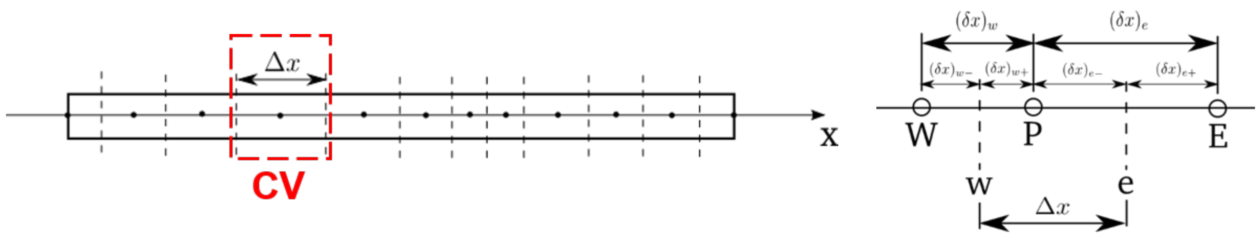
Now, A is constant and cancels out. The integral of a function differential (first term) is the function itself, to be evaluated at the integration boundaries. The second integral can be simplified by taking a spatial-average value of the source term within the control volume, denoted as $\bar{S}$. Therefore, we obtain:

$$\left[\lambda \frac{dT}{dx}\right]_w^e + \bar{S}\,[x]_w^e = 0, \tag{15}$$

which leads to:

$$\left(\lambda \frac{dT}{dx}\right)_e - \left(\lambda \frac{dT}{dx}\right)_w + \bar{S}\Delta x = 0. \tag{16}$$

where $\Delta x = x_e - x_w$ is the distance between the east and west faces, and it coincides with the CV width. Note that $\lambda(dT/dx)_e$ (and $\lambda(dT/dx)_w$) represents the heat flux, via Fourier's law of heat conduction, that is crossing the east boundary of the control volume. Therefore, Eq. (16) expresses



(a) (b)

Figure 3: (a) Finite-volume decomposition of a 1D domain, where Δx is the width of a generic control volume (CV). (b) Close-up look of a generic CV and notation used in this document: circles refer to control volume centroids, denoted as P (central node), E (east neighbour), W (west neighbour); vertical dashed lines refer to the CV faces, with e and w denoting the locations of the east and west faces of the CV centred in P; δx identifies the distance between the CV centres, whereas $\Delta x = x_e - x_w$ is the distance between the east and west faces, which coincides with the CV width.

the integral conservation of energy for the control volume at steady-state: the algebraic sum of the in/outgoing fluxes must be balanced by the generation/dissipation of heat ($\bar{S}\Delta x$) within the control volume. If there is no source term, the sum of the in/outgoing fluxes must be zero. As such, the finite-volume discretisation always satisfies the conservation of the quantity being transported even for very coarse grids, unlike other methods.

Now, how do we discretise the derivative d/dx appearing in Eq. (16), which must be calculated at the CV faces? Like we said before, we want a discretisation equation where the only unknowns are the temperature values at the control volumes centres, and thus we want to express dT/dx as a function of CV-centre temperatures. In order to do this, we must make an assumption about the behavior of $T(x)$ between the CV centres.

4.1 Approximation of derivatives

Let's write the Taylor expansion of the function $T(x)$ around a generic location x_i:

$$T(x) = T(x_i) + (x - x_i)\left.\frac{dT}{dx}\right|_{x_i} + \frac{(x-x_i)^2}{2}\left.\frac{d^2T}{dx^2}\right|_{x_i} + \frac{(x-x_i)^3}{6}\left.\frac{d^3T}{dx^3}\right|_{x_i} + \text{higher-order}, \tag{17}$$

where the higher-order terms are the x^4, $x^5, \ldots$ terms. We can use the expression above to write the expansion around the location x_e, now for convenience truncated at the second-order term, and use it to express $T(x_P)$:

$$T(x_P) = T(x_e) - \delta x_{e-}\left.\frac{dT}{dx}\right|_{x_e} + \frac{(\delta x)_{e-}^2}{2}\left.\frac{d^2T}{dx^2}\right|_{x_e} - \frac{(\delta x)_{e-}^3}{6}\left.\frac{d^3T}{dx^3}\right|_{x_e}, \tag{18}$$

where x_P identifies the CV centre and x_e the location of the east face, see Figs. 3(b) and 4; note that $\delta x_{e-} = x_e - x_P$ is the distance between the CV centre and its east face. The last term at the right-hand side is in red because it is third-order and is not actually present in the second-order expansion, but we retain it for now because it quantifies the *truncation error*, i.e. the error that we are committing by truncating an infinite sum (the Taylor expansion) and approximating it by a finite sum (e.g. the Taylor expansion truncated at the second-order term). We can use the same truncated expansion around x_e to express $T(x_E)$:

$$T(x_E) = T(x_e) + \delta x_{e+}\left.\frac{dT}{dx}\right|_{x_e} + \frac{(\delta x)_{e+}^2}{2}\left.\frac{d^2T}{dx^2}\right|_{x_e} + \frac{(\delta x)_{e+}^3}{6}\left.\frac{d^3T}{dx^3}\right|_{x_e}, \tag{19}$$

where now $\delta x_{e+} = x_E - x_e$ is the distance between the CV centre of the east neighbour and the east face of the CV centred in P. Again, the red term is not actually present in the second-order expansion, for now we leave it there just to remind us of the error committed when truncating the expansion to second-order.

Since we are looking for a discretised expression of dT/dx, we can subtract Eq. (18) from Eq. (19). Upon the simplification that $(\delta x)_{e+} = (\delta x)_{e-}$, i.e. the CV faces are located half-way between the centroids, we obtain:

$$\left.\frac{dT}{dx}\right|_{x_e} = \frac{T(x_E) - T(x_P)}{(\delta x)_e} - \frac{(\delta x)_e^2}{24}\left.\frac{d^3T}{dx^3}\right|_{x_e}, \tag{20}$$

where the second-order derivatives cancel out because of opposite signs, and the red term at the right-hand side is not actually present in the discretisation of the derivative but we keep it as a reminder of the truncation error. Note that $(\delta x)_e = x_E - x_P = (\delta x)_{e-} + (\delta x)_{e+}$ denotes the distance between the P and E control volume centres, see notation in Fig. 3(b). Equation (20) indicates that, using a Taylor expansion for $T(x)$ truncated at the second-order, we obtain an expression for dT/dx where the truncation error is proportional to $(\delta x)_e^2$. This means that, if we halve the node spacing, the error reduces by about $2^2 = 4$ times (further reading: Chapra and Canale [2], Ch. 4). In compact form, the truncation error $-\frac{(\delta x)_e^2}{24}\frac{d^3T}{dx^3}\Big|_{x_e}$ is expressed using the Big-O notation (see Big-O notation in Wikipedia) as $\mathcal{O}\left[(\delta x)_e^2 T_e'''\right]$, which signifies that the truncation error is proportional to $(\delta x)_e^2$, and to T_e''', with $T_e''' = (d^3T/dx^3)_{x_e}$, that is, the error is proportional also to the third-order derivative of the solution; this means that if the exact solution of the problem is linear or quadratic, such that $T''' = 0$, then the approximation of the derivative is exact and the truncation error is zero. The exponent of $(\delta x)_e$ in the truncation error defines the order of accuracy of the discretisation of the derivative, i.e. Eq. (20) discretises the first-order derivative with a second-order accurate scheme: the accuracy of the discretisation of the derivative becomes 4 times better if we halve the distance between the nodes. The profile assumption for $T(x)$ between the cell centroids deriving from this approximation is a *linear profile*, i.e. the derivative at x_e is calculated by assuming a linear variation of T between x_P and x_E, as sketched in Fig. 4. This discretisation scheme for the first-order derivative is often referred to as *central-difference*. Higher-order profiles exist, for example we can assume a quadratic variation of T between x_P and x_E by retaining third-order terms in the Taylor expansion; the discretisation would be more accurate (e.g. third-order, with truncation error $\mathcal{O}\left[(\delta x)_e^3\right]$), but more grid nodes would be involved in the calculation of the derivative, which is often undesired. In the CFD practice, the linear profile assumption sketched in Fig. 4 is often preferred when solving engineering problems.

4.2 Linearisation of source terms

Within Eq. (16), the term $\bar{S}$ may be a complex function of the dependent variable T. Since our objective is to eventually derive a system of algebraic equations which are linear in T, the only kind of dependency that we can handle is a linear dependency, which enables us to express the average value

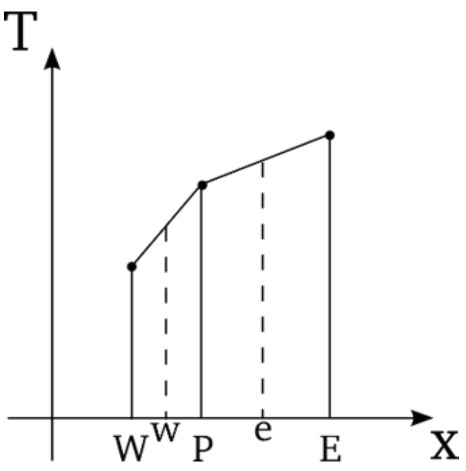


Figure 4: Piecewise-linear profile of temperature between CV centres.

$\bar{S}$ as:

$$\bar{S} = S_C + S_P T(x_P), \tag{21}$$

where S_C and S_P are independent of T, although they may be dependent on x (this does not pose any extra difficulty). If the physical phenomenon that we want to model has a non-linear source term, then we must derive a suitable linearised expression for it, e.g. by a Taylor expansion. The specific strategies for linearisation of source terms are out of the scope of this module (further reading on this: Patankar [3], Sec. 4.2.5), here we will only handle linear source terms where S_C and S_P are known. For completeness, we conclude by saying that S_C represents the explicit part of the source term because it does not depend on the temperature, whereas $S_P T(x_P)$ is the implicit part, owing to the dependence on temperature.

4.3 Final discretisation equation

Going back to Eq. (16), now we know how to express the derivatives in e and w:

$$\left(\lambda \frac{dT}{dx}\right)_e = \lambda_e \frac{T_E - T_P}{\delta x_e}, \quad \left(\lambda \frac{dT}{dx}\right)_w = \lambda_w \frac{T_P - T_W}{\delta x_w}, \tag{22}$$

where λ_e and λ_w denote the thermal conductivities evaluated at x_e and x_w, respectively, and we have lightened the notation so that, for example $T_P \equiv T(x_P)$. Combining this with the expression for the linearised source term written in Eq. (21), Eq. (16) can be rewritten as:

$$\lambda_e \frac{T_E - T_P}{\delta x_e} - \lambda_w \frac{T_P - T_W}{\delta x_w} + (S_C + S_P T_P)\,\Delta x = 0, \tag{23}$$

and, rearranging:

$$-\frac{\lambda_w}{\delta x_w} T_W + \left(\frac{\lambda_w}{\delta x_w} + \frac{\lambda_e}{\delta x_e} - S_P \Delta x\right) T_P - \frac{\lambda_e}{\delta x_e} T_E = S_C \Delta x. \tag{24}$$

This equation can be written in the compact form:

$$\boxed{a_W T_W + a_P T_P + a_E T_E = b, \tag{25}}$$

with the constant coefficients:

$$\boxed{a_W = -\frac{\lambda_w}{\delta x_w}, \quad a_P = \frac{\lambda_w}{\delta x_w} + \frac{\lambda_e}{\delta x_e} - S_P \Delta x, \quad a_E = -\frac{\lambda_e}{\delta x_e}, \quad b = S_C \Delta x. \tag{26}}$$

Equation (25) is our final discretisation equation for an internal control volume, i.e. it is valid for nodes $i = 2, 3, ..., n-1$. As intuition suggests, the temperature at the CV-centre P only depends on the temperature values at the two neighbour CV-centres W and E. Below, we will derive the discretisation equations for the boundary nodes.

4.4 Boundary conditions

Now, we want to derive the discretisation equations for the two boundary control volumes. As anticipated before, there is a vast range of boundary conditions that can apply to the system's boundaries, with the most common being Dirichlet (fixed value), Neumann (fixed gradient) and Robin boundary condition (mix of the two). Here, we will see how to derive the discretisation equation for the first two cases. If you want to know more: Patankar [3], Sec. 4.2.6.

Remember that we want to write discretisation equations which involve unknowns evaluated at the control volume centres. In the following, we consider that the boundary control volume centre ($i = 1, n$) coincides with the boundary itself, see Fig. 2 or 5. This simplifies the imposition of a Dirichlet boundary condition: it will be sufficient to set the temperature at the centre of the boundary control volume equal to the known boundary temperature. An alternative is to offset the centre of the boundary control volume, such that it is the CV face to coincide with the domain boundary, whereas its centre is internal to the domain. This complicates the imposition of the Dirichlet boundary condition, because we will have to transfer the boundary information from the boundary to the CV centre, but simplifies the imposition of a Neumann boundary condition, because gradients are naturally defined on CV faces (remember Eq. (22)). Both options are straightforward, here we decide to follow the first approach.

4.4.1 Dirichlet boundary conditions

As indicated in Fig. 5, the boundary control volume centre coincides with the physical boundary of the domain, and therefore we simply have $T_B = T_a$, with T_a being the boundary value and T_B the temperature evaluated at the boundary node, which can be interpreted as the discretisation equation for the first control volume on the left of the domain. Since these equations will be eventually solved by means of an algorithm that we have to implement, it is always good to derive a sort of unified manner to write them, be it a boundary or an internal control volume. Therefore, a convenient way to write the discretisation equation for a boundary control volume subject to a Dirichlet boundary condition is, for the first (leftmost) control volume:

$$\boxed{\mathbf{CV_1})\quad a_P T_P + a_E T_E = b, \qquad \text{with} \quad a_P = 1,\, a_E = 0,\, b = T_a, \qquad (27)}$$

which essentially means $T_P = T_a$, note that a_W does not appear because there is no west neighbour. For the last (rightmost) control volume:

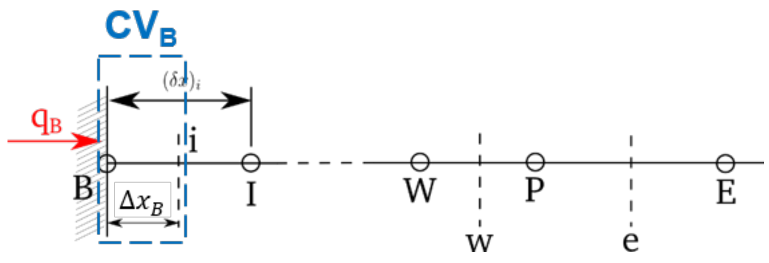


Figure 5: Illustration of the mesh near the left boundary. The boundary control volume CV_B extends from the boundary node B, which is also taken as the centre of CV_B, to the internal face i. The extension of CV_B is denoted as Δx_B. q_B denotes the heat flux imposed at the wall in the case of Neumann boundary condition.

$$\boxed{\mathbf{CV_n}) \quad a_W T_W + a_P T_P = b, \qquad \text{with} \quad a_P = 1,\, a_W = 0,\, b = T_b,} \tag{28}$$

where T_b is the temperature imposed at the $x = L$ boundary. Therefore, the two equations above can be simply merged with those for the internal nodes, Eqs. (25) and (26), to build the final linear system, as we will see later.

4.4.2 Neumann boundary conditions

In heat transfer, but many other fields as well, we may have a boundary condition where the temperature gradient is known, and not the temperature itself. Think, for example, of the external wall of a house, subject to solar radiation; the solar radiation generates a heat flux that is known and is on the order of $1000\,\mathrm{W/m^2}$. We may want to impose this boundary condition to study heat conduction through the wall that is hit by the solar radiation on one side, and is maintained at a constant temperature on the other side, where for example a thermostat keeps the room at a set temperature. How do we convert the information about the heat flux into an information about the temperature gradient at the wall? Very simple, we use the Fourier's law of heat conduction, in 1D:

$$q = -\lambda \frac{dT}{dx}. \tag{29}$$

In order to derive a discretisation equation for the boundary control volume subject to a constant heat flux (Neumann) condition, let's restart from Eq. (12), now written for a control volume centred at the boundary node B, bounded by B at west and by the internal face i at east (see Fig. 5):

$$\int_B^i \frac{d}{dx}\left(\lambda \frac{dT}{dx}\right) A\, dx + \int_B^i S(x,T) A\, dx = 0. \tag{30}$$

Following the same procedure as before:

$$\left(\lambda \frac{dT}{dx}\right)_i - \left(\lambda \frac{dT}{dx}\right)_B + \bar{S}\Delta x_B = 0. \tag{31}$$

Now, the first term can be expressed as a function of values of the temperature calculated at CV-centres as indicated in Eq. (22); the second term, representing the heat flux at B, coincides with the known boundary condition q_B, so that we obtain:

$$\lambda_i \frac{T_I - T_B}{\delta x_i} + q_B + (S_C + S_P T_B)\,\Delta x_B = 0, \tag{32}$$

where T_I is the temperature at the centre of the leftmost internal control volume (see Fig. 5), T_B is the temperature at the left boundary control volume which now is unknown (unlike the case of Dirichlet boundary condition), and q_B is the known heat flux imposed at the boundary. For convenience, we can rearrange the equation above to make it look the same as those for Dirichlet conditions (the differences are hidden in the coefficients):

$$\boxed{a_P T_P + a_E T_E = b, \qquad \text{with} \quad a_P = \frac{\lambda_e}{\delta x_e} - S_P \Delta x_B,\, a_E = -\frac{\lambda_e}{\delta x_e},\, b = S_C \Delta x_B + q_B,} \tag{33}$$

where the west neighbour is absent and the known heat flux q_B enters the known term vector on the right-hand side of the equation. You have to be careful with Δx_B because, if the grid is made of equidistant nodes, the width of the boundary control volume is half that of the internal node, so that $\Delta x_B = \Delta x/2$, with Δx denoting the width of internal control volumes. You can easily rework the procedure in the case that the constant heat flux is applied to the right boundary, paying attention to the signs. However, note that you cannot have Neumann-Neumann conditions (Neumann on both boundaries), because the value of the temperature must be set at least on one boundary to allow solution of the mathematical problem. For all the remaining internal nodes, the discretisation equations are still Eqs. (25) and (26).

4.5 Assembling the linear system of equations

Now that we have seen how to derive discretisation equations for all the boundary and internal CVs, and we have seen that we can always express these in the general form $a_W T_W + a_P T_P + a_E T_E = b$, regardless of the control volume location or specific boundary condition set, we can readily assemble our system of n linear algebraic equations. Let's help ourselves with a more handy notation:

$$\mathbf{CV_1})\quad a_{1,P}T_{1,P} + a_{1,E}T_{1,E} = b_1 \Rightarrow a_{1,1}T_1 + a_{1,2}T_2 = b_1$$

$$\mathbf{CV_i})\quad a_{i,W}T_{i,W} + a_{i,P}T_{i,P} + a_{i,E}T_{i,E} = b_i \Rightarrow a_{i,i-1}T_{i-1} + a_{i,i}T_i + a_{i,i+1}T_{i+1} = b_i$$

$$\mathbf{CV_n})\quad a_{n,W}T_{n,W} + a_{n,P}T_{n,P} = b_n \Rightarrow a_{n,n-1}T_{n-1} + a_{n,n}T_n = b_n$$

where, for example, $a_{1,E}$ refers to the east coefficient (a_E) of the discretisation equation for the first control volume and $T_{1,E}$ is the unknown temperature of the east neighbour to the first control volume; with this notation, you can imagine that $T_{1,E}$ coincides with $T_{2,P}$, i.e. the east neighbour temperature for control volume 1 is the central control volume temperature when discretising the equation for control volume 2. In the rightmost set of equations above, the subscripts W, P, E are replaced, respectively, by the subscripts $i-1$, i and $i+1$, by considering that the control volumes will be numbered in ascending order from left to right. This notation is more suitable for coding the algorithm that will fill the coefficient matrix.

The system of linear equations written above, using the notation adopted in the rightmost set of equations above, can be expressed in matricial form as follows:

$$\begin{pmatrix}
a_{1,1} & a_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
a_{2,1} & a_{2,2} & a_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
0 & a_{3,2} & a_{3,3} & a_{3,4} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & \cdots & 0 & a_{i-1,i-2} & a_{i-1,i-1} & a_{i-1,i} & 0 & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & a_{i,i-1} & a_{i,i} & a_{i,i+1} & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & 0 & a_{i+1,i} & a_{i+1,i+1} & a_{i+1,i+2} & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & \cdots & 0 & 0 & 0 & 0 & 0 & a_{n-2,n-3} & a_{n-2,n-2} & a_{n-2,n-1} & 0 \\
0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\
0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{n,n-1} & a_{n,n}
\end{pmatrix}
\times
\begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_{i-1} \\ T_i \\ T_{i+1} \\ \vdots \\ T_{n-2} \\ T_{n-1} \\ T_n \end{pmatrix}
=
\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{i-1} \\ b_i \\ b_{i+1} \\ \vdots \\ b_{n-2} \\ b_{n-1} \\ b_n \end{pmatrix}$$

and the task of the computer is to solve this linear system of equations in order to find the values of the n unknowns T_i at the n discrete locations composing the computational domain. Recall that the matricial problem can be written in compact form as

$$\mathbf{A} \times \mathbf{T} = \mathbf{B} \tag{34}$$

where $\mathbf{A}$ is the $n \times n$ matrix of the coefficients, $\mathbf{T}$ is the $n \times 1$ vector of unknowns and $\mathbf{B}$ is the $n \times 1$ vector of known terms. You can note that $\mathbf{A}$ has a rather regular appearance: the generic $a_{i,j}$ element, where i (index of rows) identifies the CV which the equation refers to, and j (index of columns) identifies the contribution of the $j-th$ CV to i, is $a_{i,j} \neq 0$ if j is a neighbour of i, and $a_{i,j} = 0$ if j is not a neighbour of i; the elements along the diagonal, $a_{i,i}$, must always be nonzero. Therefore, in a 1D configuration, where each CV has only two neighbours which contribute to its energy balance (via the heat fluxes at the CV faces), $\mathbf{A}$ is a *tridiagonal* matrix, where only 3 diagonals have nonzero elements, while all the remaining coefficients of the matrix are zero. Intuitively, if we were using a higher-order scheme (e.g. quadratic) to discretise the temperature gradients (recall Sec. 4.1), requiring more points to calculate derivatives, then the discretisation equation would have involved more neighbours, e.g. west-west and east-east neighbours, and thus $\mathbf{A}$ would have had more nonzero elements, e.g. becoming pentadiagonal (5 nonzero diagonals).

We will present some methods to solve matricial problems in the next lectures. Without going into details, it is for now important to know that, for the stability of the numerical schemes involved in the solution:

- The diagonal coefficients ($a_{i,i}$) are positive,
- The off-diagonal coefficients ($a_{i,i-1}$, $a_{i,i+1}$) are negative.

Take a look at the expressions for the coefficients in Eq. (26). Since the thermal conductivity λ and grid spacing δx are always positive, the off-diagonal coefficients $a_W, a_E < 0$ in agreement with the second condition above. The diagonal coefficient a_P is positive if $S_P \leq 0$, but it can become negative if $S_P > 0$. In this latter case, that we will not treat in this module, the entire source term is treated explicitly and the matricial system must be solved iteratively (further reading: Patankar [3], Sec. 4.2.5).

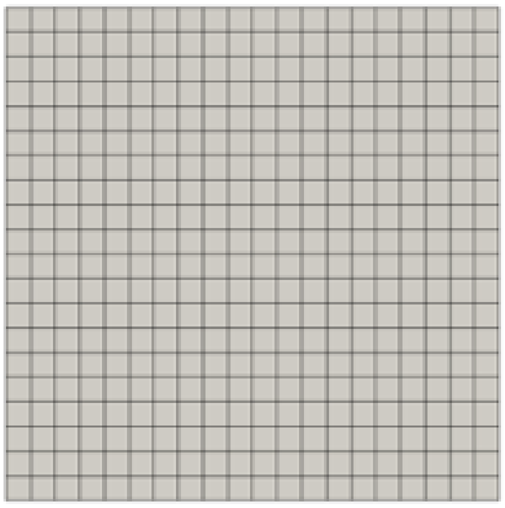
1D finite-volume method – Further reading: Patankar [3], Ch. 4.2.

5 The finite-volume method – 2D, structured mesh

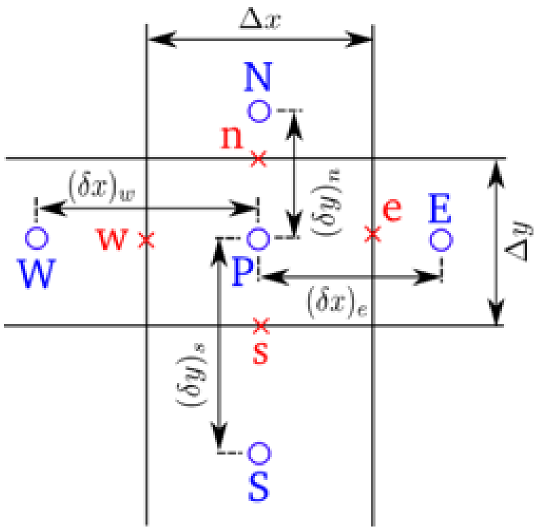
We extend now the derivation of the discretisation equation to the steady-state heat equation in the 2D case. This is shown here for completeness, but **you are not required to know this for the assessment**, so it is only for your information. The steady-state heat conduction equation in 2D is now a PDE:

$$\boxed{\frac{\partial}{\partial x}\left(\lambda \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(\lambda \frac{\partial T}{\partial y}\right) + S(x, y, T) = 0,} \tag{35}$$

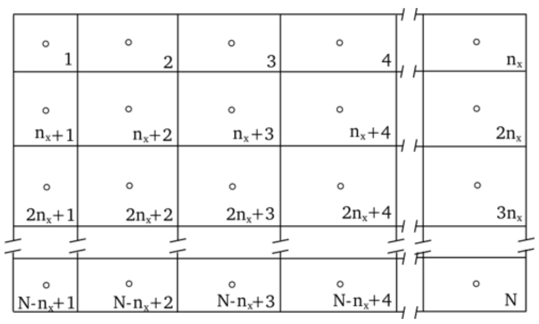
where now the ordinary derivative symbol d has been replaced by the partial derivative symbol ∂; note that Eq. (35) is an elliptic PDE and is often referred to as *Poisson equation*, which differs from the Laplace equation (Eq. (2)) only for the presence of the source term. For simplicity, we will consider the discretisation of the equation above over a simple, square/rectangular domain, see Fig. 6, which can be decomposed into well-ordered square/rectangular control volumes. This way, the



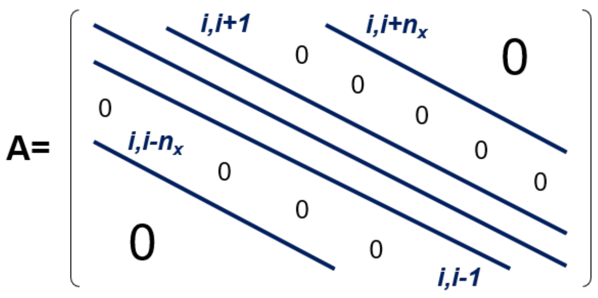
(a)



(b)



(c)



(d)

Figure 6: (a) An example of a 2D structured mesh. (b) Control volume configuration for 2D structured case. (c) Example of indexing of the control volumes in the mesh, where n_x is the number of elements along x and $n = n_x \times n_y$ (n_y: the number of elements along y) is the total number of elements of the domain (called N in the figure). (d) Architecture of the matrix of the coefficients for a 2D structured mesh case.

control volumes can be easily indexed in ascending order following, for example, a west-to-east and north-to-south approach as depicted in Fig. 6(b). The advantage of this indexing strategy is that each CV knows exactly who its east, west, north and south neighbours are based on its own index (as we will see below), and there is no need to store the mesh *connectivity*, i.e. the information about who the neighbours of each cell are. This kind of computational mesh, whose architecture is such that there is no need to store the CV connectivity, is called *structured mesh*. Another great advantage of structured meshes is on the resulting layout of the coefficient matrix, as will see below.

So, without going into the details (but you can try to work it out yourself), the finite-volume discretisation of Eq. (35) on a structured mesh can be done using a 4-neighbour stencil (north, east, west, south) as shown in Fig. 6(b), and following the same procedure seen for the 1D case, but now using double integrals (along both x and y). The procedure leads to the following final discretisation equation:

$$a_N T_N + a_W T_W + a_P T_P + a_E T_E + a_S T_S = b, \tag{36}$$

which, at a first glance, differs from the 1D case, Eq. (25), only for the presence of the extra north (N) and south (S) neighbour contributions. Of course, the coefficients $a_N, a_W, ...,$ now write slightly differently, but the main point that we want to make here is that the discretisation equation for the 2D case is very similar to that for the 1D case. The only extra terms come from the fact that, in 2D, we have two more CV neighbours whose energy fluxes impact the temperature of the central cell in P.

Let's now think of assembling the system of equations made of Eq. (36) for each CV of the domain. Let's consider that our square/rectangular domain is decomposed with a structured mesh of n_x cells along the horizontal direction, and n_y along the vertical direction, so $n = n_x \times n_y$ mesh cells in total. For the generic $i - th$ cell, we can rewrite Eq. (36) to express the indexes in a code-friendly manner as we did before. First, let's clarify the indexes of the neighbours. If we adopt the west-to-east and north-to-south indexing approach as depicted in Fig. 6(b), we have:

$$P \rightarrow i, \quad E \rightarrow i+1, \quad W \rightarrow i-1, \quad N \rightarrow i-n_x, \quad S \rightarrow i+n_x. \tag{37}$$

Note the "power" of the structured mesh: from i, we can retrieve the indexes of the 4 neighbours simply by adding/subtracting 1 or n_x. Therefore, for the $i - th$ control volume, the code-friendly version of Eq. (36) will look like:

$$a_{i,i-n_x} T_{i-n_x} + a_{i,i-1} T_{i-1} + a_{i,i} T_i + a_{i,i+1} T_{i+1} + a_{i,i+n_x} T_{i+n_x} = b_i, \tag{38}$$

where, just like for the 1D case, the first subscript refers to the CV which the equation refers to, and the second subscript refers to the contributing neighbour. Assembling the n equations for the n control volumes, we will obtain again a matricial problem $\mathbf{A} \times \mathbf{T} = \mathbf{B}$, where $\mathbf{A}$ is the $n \times n$ matrix of the coefficients, $\mathbf{T}$ is the $n \times 1$ vector of unknowns and $\mathbf{B}$ is the $n \times 1$ vector of known terms; n is the total number of cells of the domain ($n = n_x \times n_y$). The matricial system to be solved looks analogously to the 1D case, but there is an important difference in the layout of $\mathbf{A}$, which is schematically sketched in Fig. 6(d): there are still three nonzero diagonals just like in the 1D case but, more externally, there two additional nonzero diagonals that were not there in 1D. Intuitively, these two extra nonzero diagonals are due to the north (diagonal $i, i - n_x$) and south (diagonal $i, i + n_x$) neighbour contributions. $\mathbf{A}$ is a so-called *pentadiagonal* matrix, due to the 5 nonzero diagonals.

The figure on the title page of this document has been obtained by solving Eq. (35) with the finite-volume method, with Dirichlet boundary conditions at all four boundaries (right, bottom, left: $T = 293\,\mathrm{K}$, top: $T = 393\,\mathrm{K}$) and $S(x, y, T) = 0$, using the opensource finite-volume solver Openfoam.

2D finite-volume method – Further reading: Patankar [3], Ch. 4.4.1.

6 The finite-volume method – 3D, structured mesh

The extension to 3D and structured meshes is now trivial. Again, this is shown here for completeness, but **you are not required to know this for the assessment**, so it is only for your information. The steady-state heat conduction equation in 3D reads as follows:

$$\frac{\partial}{\partial x}\left(\lambda\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(\lambda\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(\lambda\frac{\partial T}{\partial z}\right)+S(x,y,z,T)=0, \tag{39}$$

For simplicity, we will consider the discretisation of the equation above over a simple, cuboid domain, see Fig. 7(a), which can be decomposed into well-ordered cubic control volumes. The finite-volume discretisation of Eq. (39) on a structured mesh can be done using a 6-neighbour stencil (north, east, west, south, top, bottom), and following the same procedure seen for the 1D/2D cases, but now using triple integrals (along x, y and z). The procedure leads to the following final discretisation equation:

$$a_T T_T + a_N T_N + a_W T_W + a_P T_P + a_E T_E + a_S T_S + a_B T_B = b, \tag{40}$$

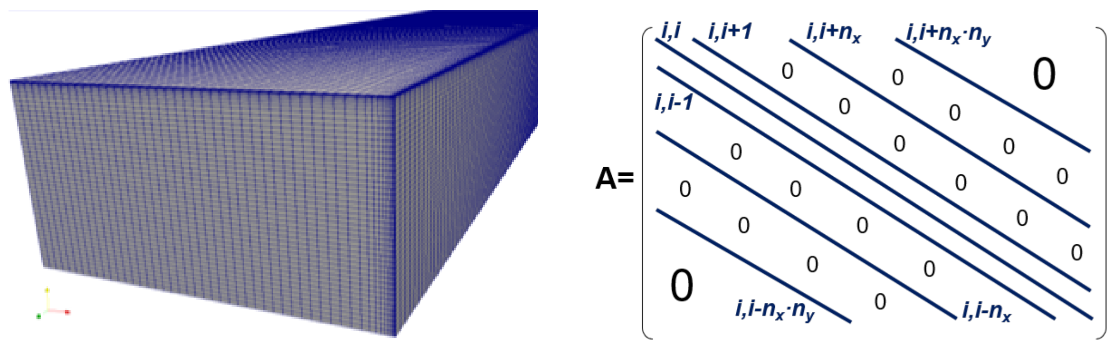
which, compared to the 2D case, incorporates the extra contribution of the top (T) and bottom (B) neighbours. Considering a computational mesh with $n = n_x \times n_y \times n_z$ cells, indexed from west-to-east, north-to-south and top-to-bottom, such that $T \rightarrow i - n_x \times n_y$ and $B \rightarrow i + n_x \times n_y$, the code-friendly version of Eq. (40) for the $i-th$ control volume will look like:

$$a_{i,i-n_x\times n_y}T_{i-n_x\times n_y} + a_{i,i-n_x}T_{i-n_x} + a_{i,i-1}T_{i-1} + \tag{41}$$

$$+ a_{i,i}T_i + a_{i,i+1}T_{i+1} + a_{i,i+n_x}T_{i+n_x} + a_{i,i+n_x\times n_y}T_{i+n_x\times n_y} = b_i, \tag{42}$$

Therefore, assembling the n equations for the n control volumes, we will obtain again a matricial problem $\mathbf{A} \times \mathbf{T} = \mathbf{B}$, where $\mathbf{A}$ is the $n \times n$ matrix of the coefficients, $\mathbf{T}$ is the $n \times 1$ vector of unknowns and $\mathbf{B}$ is the $n \times 1$ vector of known terms; n is the total number of cells of the domain ($n = n_x \times n_y \times n_z$). The layout of $\mathbf{A}$ resulting from the 3D structured mesh is schematically sketched in Fig. 7(b): there are still five nonzero diagonals just like in the 2D case but, more externally, there are two additional nonzero diagonals that were not there in 2D. Intuitively, these two extra nonzero diagonals are due to the top (diagonal $i, i - n_x \times n_y$) and bottom (diagonal $i, i + n_x \times n_y$) neighbours contributions. $\mathbf{A}$ is a so-called *eptadiagonal* matrix, due to the 7 nonzero diagonals.

3D finite-volume method – Further reading: Patankar [3], Ch. 4.4.2.



(a) (b)

Figure 7: (a) An example of a 3D structured mesh. (b) Architecture of the matrix of the coefficients for a 3D structured mesh case.

7 The finite-volume method – Unstructured mesh

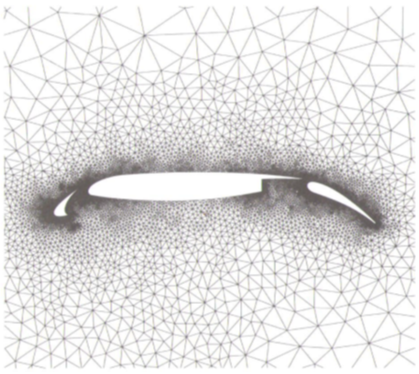
A clear limitation of structured grids is that they are not suitable for complex domains, such as those often encountered in engineering problems, see for example the airfoil in Fig. 8(a). In such cases, it is preferable to employ an *unstructured mesh*, that is, a mesh that does not show any regular structure (Fig. 8(a)) but that, for the same reason, can easily follow the complex boundaries of the domain. Below, we will briefly introduce how to extend the finite-volume method to deal with an unstructured grid; bear in mind that **you are not required to know this for the assessment**, so it is only for your information.

The steady-state heat conduction equation is now written in its most general form by using vector calculus notation:

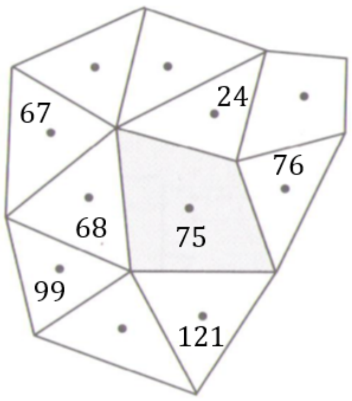
$$\boxed{\nabla \cdot (\lambda \nabla T) + S(\mathbf{x}, T) = 0,} \tag{43}$$

where the nabla operator ∇ writes differently depending on the system of coordinates used. Therefore, Eq. (43) is valid for any 1D/2D/3D case. The finite-volume discretisation of Eq. (43) in the case of a control volume with non-orthogonal boundaries, e.g. a skewed triangle or polygon, is not trivial but, based on the experience built with the previous sections, we can imagine that the discretisation equation for a generic CV will involve contributions from the neighbour cells, i.e. cells that share a face with the CV. Consider the situation depicted in Fig. 8(b), where the 68*th* cell has the three cells 67*th*, 75*th* and 99*th* as neighbours. Intuitively, the finite-volume discretisation of Eq. (43) for the 68*th* cell may look like:

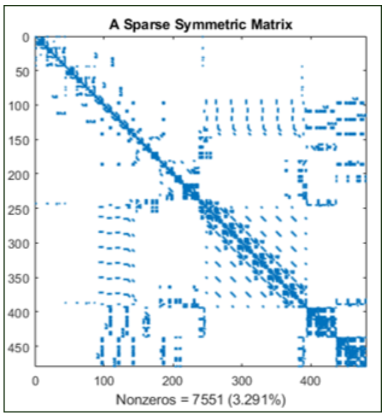
$$a_{68,67}T_{67} + a_{68,68}T_{68} + a_{68,75}T_{75} + a_{68,99}T_{99} = b_{68}, \tag{44}$$



(a)



(b)



(c)

Figure 8: (a) An example of an unstructured (2D) mesh; source: Versteeg and Malalasekera [5]. (b) Control volume configuration and indexing for unstructured case. (c) Architecture of the matrix of the coefficients for an example unstructured mesh case, the blue dots indicate nonzero coefficients, the white spaces indicate zeros; source: Mathworks.

whereas for the 75*th* cell:

$$a_{75,24}T_{24} + a_{75,68}T_{68} + a_{75,75}T_{75} + a_{75,76}T_{76} + a_{75,121}T_{121} = b_{75}. \tag{45}$$

Note that now the cell connectivity is quite random, and therefore the information about the indexes of the neighbours of each control volume will need to be stored somewhere, with consequent increase of the required memory, compared to structured grids where this is not necessary. When we assemble the system of n linear equations for the n control volumes making up the discretised domain, we still obtain a matricial problem $\mathbf{A} \times \mathbf{T} = \mathbf{B}$, where $\mathbf{A}$ is the $n \times n$ matrix of the coefficients, $\mathbf{T}$ is the $n \times 1$ vector of unknowns and $\mathbf{B}$ is the $n \times 1$ vector of known terms. However, the consequence of the irregular grid connectivity of the unstructured mesh is now readily apparent on the layout of the matrix of the coefficients, see the example in Fig. 8(c): the nonzero elements of the matrix are now sparsely distributed, while most of the coefficients are still zero. $\mathbf{A}$ is now a *sparse* matrix. More memory is required to store the data of sparse matrices, compared to *band-diagonal* (tri/penta/eptadiagonal) matrices. Consider that, to save space, we do not want to store the full $n \times n$ matrix (i.e. n^2 elements), but only the nonzero coefficients (which are $\sim n$). For band matrices this is easy: only the nonzero diagonals are stored, as separate vectors, and each vector is identified by one single index that "remembers" the location of the diagonal within the matrix. However, for sparse matrices, the nonzero elements (which are always very few, $\sim n$) are randomly distributed, and more sophisticated and memory/time-consuming methods are necessary in order to efficiently index and store only the nonzero elements. Since the linear solver (which solves $\mathbf{A} \times \mathbf{T} = \mathbf{B}$) needs continuous access to the elements of the matrix during the calculation, a slower access results in a more RAM- and time-consuming simulation (further reading: Press et al. [4], Sec. 2.4 and 2.7). Furthermore, linear solvers take advantage of the regular structure of $\mathbf{A}$ resulting from structured meshes, and tend to be more efficient than general purpose linear solvers (for sparse matrices).

We are now ready to summarise the main pros and cons of structured and unstructured grids:

- Structured meshes do not need to store the CV connectivity (whereas unstructured meshes do need it), thus saving disk space and reducing the number of operations during the solution procedure.
- Structured meshes need less memory space and allow faster access to the matrix elements during the solution of the linear system, compared to unstructured meshes, because all the nonzero elements are well-organised along diagonals.
- Linear solvers specific for band matrices (structured mesh) are more efficient than general purpose ones.
- Structured meshes are limited to very simple computational domains, while unstructured meshes are more suitable for the complex geometries encountered in engineering applications.

Finite-volume method for unstructured mesh – Further reading: Versteeg and Malalasekera [5], Ch. 11.6.

8 Brief introduction to the solution of the linear system

The brief introduction to the finite-volume method in 2D and 3D, and unstructured meshes, had the scope of showing that, regardless of the dimension of the problem (1D/2D/3D) and of the regularity of the grid (structured/unstructured), the numerical solution of a differential equation on a computational mesh with n cells always involves the solution of a linear system of n equations, $\mathbf{A} \times \mathbf{T} = \mathbf{B}$, with n unknowns $T_1, T_2, ..., T_n$, with:

$$\mathbf{A} = \begin{vmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{vmatrix}, \quad \mathbf{T} = \begin{vmatrix} T_1 \\ \vdots \\ T_n \end{vmatrix}, \quad \mathbf{B} = \begin{vmatrix} b_1 \\ \vdots \\ b_n \end{vmatrix}.$$

Depending on the morphology or regularity of $\mathbf{A}$, e.g. full, band-diagonal, triangular, symmetric, positive definite, etc. etc., there exist optimal solution algorithms. These can be grouped into:

- **Direct methods:** compute the solution $T_1, T_2, ..., T_n$ within a finite number of operations.
- **Iterative methods:** are based on the sequential evaluation of approximate solutions that are supposed to converge to the exact solution after ∞ iterations.

Direct and iterative algorithms will be the subject of the next class and are discussed in another document.

9 Indicators of the accuracy of the numerical solution

At this point, you should have understood that the numerical solution of a differential problem (for those treated in this module) involves the solution of a linear system of equations, $\mathbf{A} \times \mathbf{T} = \mathbf{B}$. Therefore, how can we make sure that the numerical solution $T_1, T_2, ..., T_n$, calculated at discrete grid points $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}$, now in bold font to account for 2D/3D geometries, is correct?

- The first tool that we have is, of course, our intuition. Look at the solution! For example, if there is no source term ($S(\mathbf{x}, T) = 0$), and you have Dirichlet conditions at all boundaries, it should be intuitive that the solution should be bounded between the boundary values. In the example depicted on the title page, where the boundaries are kept at 293 K and 393 K, it would not make sense to see temperatures $T < 293\,\mathrm{K}$ or $T > 393\,\mathrm{K}$.
- Are the boundary conditions respected? If you have a Dirichlet boundary condition at a boundary, is T on that boundary equal to the value that you imposed? If you have a Neumann

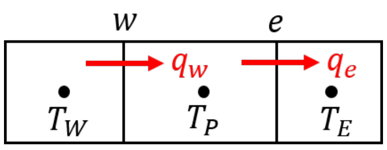


Figure 9: Sketch of the heat fluxes calculated at the faces of the control volume P, in a 1D geometry. Remember that $q_w = -\lambda_w(T_P - T_W)/\delta_{x_w}$ and $q_e = -\lambda_e(T_E - T_P)/\delta_{x_e}$.

boundary condition, is the temperature gradient (or heat flux) at the boundary equal to the set value?

- Always try to compare your numerical solution with benchmark data, although this is not always possible. If the problem is simple and it has a theoretical solution (like those that we will see in this module), then compare the numerical solution with the analytical solution. If this is not possible, try to gather validation data from experimental measurements or numerical results obtained by someone else, that can be found in the literature. The ERCOFTAC database is a valuable resource for fluid flow data.

- The three tools listed above verify that your solution is mathematically and physically correct. But is it numerically correct? The first thing you can look at is whether the problem $\mathbf{A} \times \mathbf{T} = \mathbf{B}$ has been solved correctly by your linear solver. Hence, the first check you can do is about the *residual error*:

$$R = \frac{|\mathbf{B} - \mathbf{A} \times \mathbf{T}|}{|diag(\mathbf{A}) \times \mathbf{T}|}, \tag{46}$$

  where the numerator is the absolute error which tells you how far you are from the exact solution, that is $\mathbf{B} - \mathbf{A} \times \mathbf{T} = \mathbf{0}$, and the denominator is a normalisation factor that makes the magnitude of the error independent of the scale of the problem. Numerical methods can never achieve the perfect $R = 0$ solution, but a good solution has $R \leq 10^{-6}$, depending on the complexity of the problem, as a rule of thumb.

- Is the energy balance respected? At steady-state, and for $S(\mathbf{x}, T) = 0$, we are expecting that the sum of the in/outgoing fluxes in each cell is zero. If we are solving the heat equation in 1D, this means that $q_w - q_e = 0$ in each cell, with q being the heat flux (remember Fourier: $q = -\lambda\, dT/dx$) which is always calculated on the control volume faces, according to the two neighbour cell-centred temperatures (recall Eq. (22)):

$$q_w - q_e = 0 \Rightarrow -\left[\lambda \frac{dT}{dx}\right]_w + \left[\lambda \frac{dT}{dx}\right]_e = 0 \Rightarrow -\lambda_w \frac{T_P - T_W}{\delta_{x_w}} + \lambda_e \frac{T_E - T_P}{\delta_{x_e}} = 0. \tag{47}$$

  Since we are solving the heat conduction equation numerically, the balance above will not be perfectly equal to zero. In order to obtain an indication of the total unbalance, we sum up the absolute values of the difference above, looping through all the n cells of the domain:

$$error = \frac{\sum_{i=1}^{i=n} |q_{i,w} - q_{i,e}|}{q_{ref}}, \tag{48}$$

  where q_{ref} is an arbitrary reference value that rescales the absolute error to make it independent of the scale of the problem. This can be an average of all the heat fluxes in all the cells, $q_{ref} = \sum |q_{i,w}|$, or a reference value based on the boundary conditions, e.g. if we have Dirichlet conditions, $q_{ref} = \lambda |T_a - T_b|/L$.

- The convergence order of the solution. We have seen in Sec. 4.1 that the truncation error committed when discretising first-order spatial derivatives with a central-difference scheme is

$\mathcal{O}[(\delta x)^2 T''']$, i.e. the central-difference scheme has convergence order 2 (or, equally, is a second-order method), and therefore we expect this error to decrease 4 times if we halve the grid spacing. The steady heat conduction equation requires discretisation of the first-order spatial derivative only, and therefore the overall truncation error committed when discretising the equation is that related to dT/dx; if other higher-order derivatives with respect to x were present, each requiring its own discretisation via a truncated Taylor expansion, then the overall error would be related to the lowest convergence order among the different terms of the equation. In the present case, where only dT/dx is present and is discretised using a second-order scheme, we expect our discrete temperature solution to become 4 times more accurate (in space) if we halve the grid spacing. Verification of this allows us to verify that the numerical method does what expected. How do we verify this? We can repeat the simulation with different mesh spacings, e.g. doubling and halving the starting value of δx, and keep record of the deviation between the numerical and exact solution at a specific point of the domain. Then, we plot a graph of this deviation vs δx in log-log coordinates, and we find the exponent of the curve $error \sim (\delta x)^k$, which represents the convergence order of our solution with respect to the spatial discretisation. If the convergence order that we expect from our method, based on the truncation error of the spatial discretisation schemes adopted, is close to k, then the method behaves correctly. We will see this in Worked example 2 below.

10 Matlab tutorials: finite-volume solution of the steady 1D heat equation

10.1 Worked example 1: Dirichlet boundary conditions, zero source term

Implement a FV code in Matlab to solve the steady 1D heat conduction equation, Eq. (11a), with the following conditions:

- $L = 1\,\mathrm{m}$.
- Zero source term, $S = 0$.
- Constant thermal conductivity, $\lambda = 400\,\mathrm{W/(m\,K)}$.
- $n = 21$ equidistant nodes.

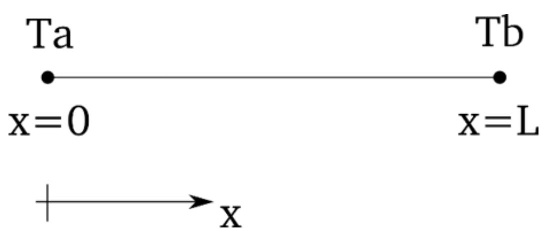


Figure 10: Sketch of the domain and boundary conditions for the tutorials.

- Dirichlet boundary conditions at both boundaries, $T(x=0) = T_a = 300\,\mathrm{K}$ and $T(x=L) = T_b = 320\,\mathrm{K}$.

To solve the linear system, make use of matlab's "backslash" operator: $T = A \setminus B$, which is based on a direct method (Mathworks). Perform the solution checks suggested in Sec. 9. The problem has the following theoretical solution:

$$T(x) = T_a + \frac{T_b - T_a}{L} x. \tag{49}$$

Solution

Ok, so all we need to do is to create the matrix $\mathbf{A}$ and vector $\mathbf{B}$, based on the finite-volume discretisation of the heat equation for this specific worked example. We have seen how to do this in Sec. 4.5.

Let's start with defining the geometry. We take $n = 21$ control volumes, with the centres of the two boundary control volumes matching the domain boundaries, as sketched in Fig. 3(a). Considering equidistant CV centres, the distance between adjacent centres is $\delta x = L/(n-1) = 1/(21-1) = 0.05\,\mathrm{m}$. By taking the CV faces midway between adjacent centres, the width of internal control volumes is $\Delta x = \delta x = 0.05\,\mathrm{m}$, whereas the width of the two boundary control volumes will be half of this, $\Delta x = 0.025\,\mathrm{m}$. However, for Dirichlet boundary conditions, we will not need the Δx of the boundary CV.

Let's consider the first control volume, whose centre coincides with the $x = 0$ boundary:

$$\mathbf{CV_1}) \quad a_{1,P}T_{1,P} + a_{1,E}T_{1,E} = b_1 \Rightarrow a_{1,1}T_1 + a_{1,2}T_2 = b_1, \tag{50}$$

and, owing to the Dirichlet boundary condition, Eq. (27), $a_{1,1} = 1$, $a_{1,2} = 0$ and $b_1 = T_a$.

For any internal control volume:

$$\mathbf{CV_i}) \quad a_{i,W}T_{i,W} + a_{i,P}T_{i,P} + a_{i,E}T_{i,E} = b_i \Rightarrow a_{i,i-1}T_{i-1} + a_{i,i}T_i + a_{i,i+1}T_{i+1} = b_i, \tag{51}$$

where the coefficients are, Eq. (26), $a_{i,i-1} = -\lambda/\delta x$, $a_{i,i} = 2\lambda/\delta x$, $a_{i,i+1} = -\lambda/\delta x$ and $b_i = 0$. Note that, compared to Eq. (26), we are disregarding the source term contributions to $a_{i,i}$ (that is, $-S_P\Delta x$) and b_i ($+S_C\Delta x$), but for this example it does not matter because the source terms are zero. Feel free to include them if you wish to write a more general solution code.

For the last control volume:

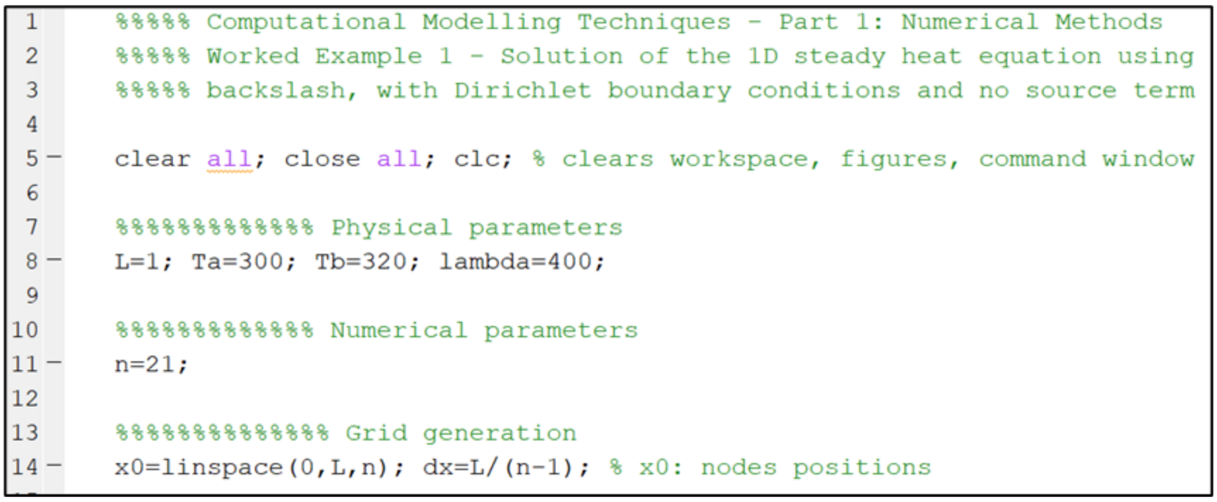
$$\mathbf{CV_n}) \quad a_{n,W}T_{n,W} + a_{n,P}T_{n,P} = b_n \Rightarrow a_{n,n-1}T_{n-1} + a_{n,n}T_n = b_n, \tag{52}$$

and, owing to the Dirichlet boundary condition, Eq. (28), $a_{n,n} = 1$, $a_{n,n-1} = 0$ and $b_n = T_b$.

Therefore, the matrices will look like:

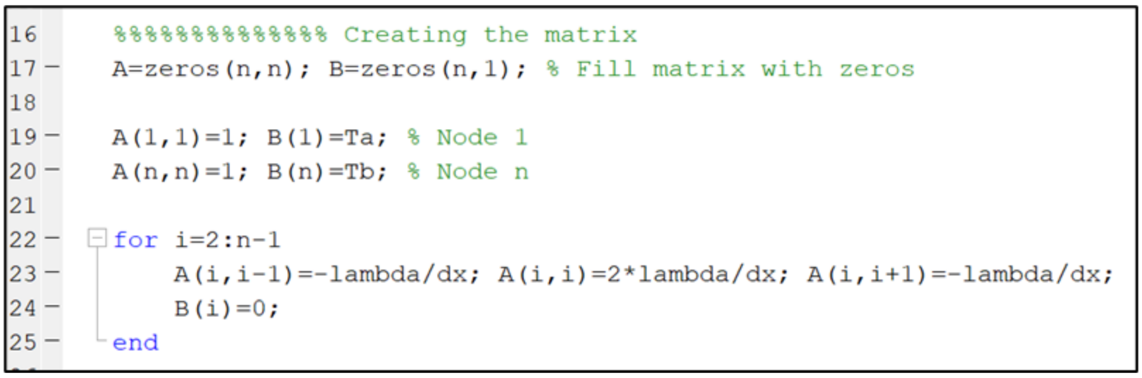
$$\mathbf{A} = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & 0 & -\frac{\lambda}{\delta x} & 2\frac{\lambda}{\delta x} & -\frac{\lambda}{\delta x} & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{vmatrix}, \quad \mathbf{B} = \begin{vmatrix} T_a \\ 0 \\ \vdots \\ 0 \\ T_b \end{vmatrix}.$$

Looking at our matlab code, we can start with the definition of the problem parameters, generation of a vector storing the nodes locations, and calculation of the distance between CV centres (δx):



```
1     %%%%% Computational Modelling Techniques - Part 1: Numerical Methods
2     %%%%% Worked Example 1 - Solution of the 1D steady heat equation using
3     %%%%% backslash, with Dirichlet boundary conditions and no source term
4
5     clear all; close all; clc; % clears workspace, figures, command window
6
7     %%%%%%%%%%%%%% Physical parameters
8     L=1; Ta=300; Tb=320; lambda=400;
9
10    %%%%%%%%%%%%%% Numerical parameters
11    n=21;
12
13    %%%%%%%%%%%%%%% Grid generation
14    x0=linspace(0,L,n); dx=L/(n-1); % x0: nodes positions
```

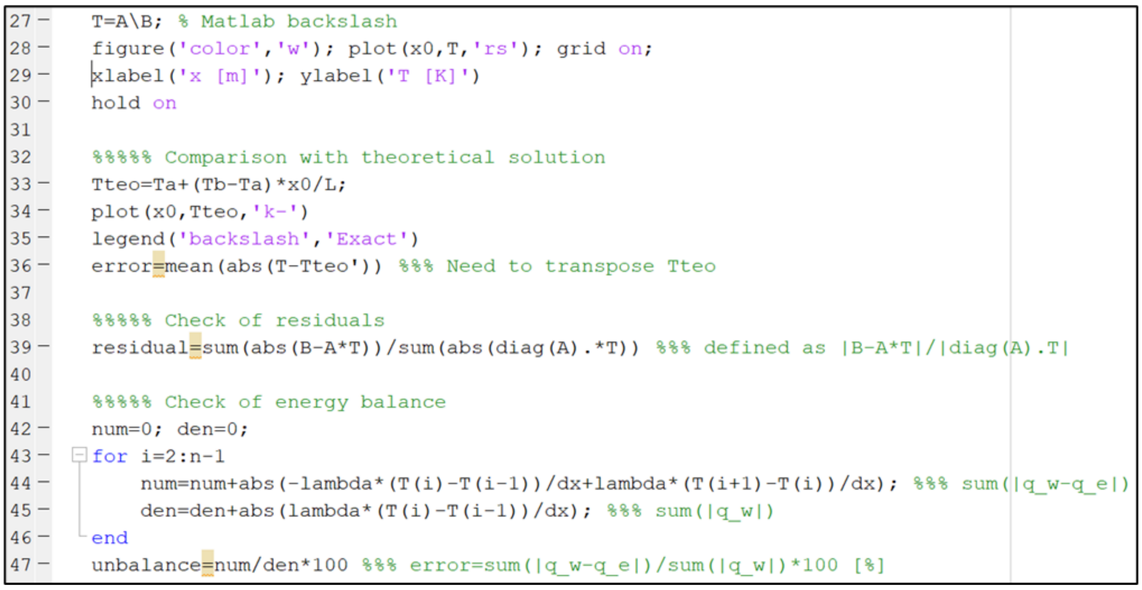
where `linspace` generates a linearly spaced vector (type `help linspace` on the command window for help with Matlab commands). Let's now create and fill the matrix **A** and vector **B**:



```
16    %%%%%%%%%%%%%%% Creating the matrix
17    A=zeros(n,n); B=zeros(n,1); % Fill matrix with zeros
18
19    A(1,1)=1; B(1)=Ta; % Node 1
20    A(n,n)=1; B(n)=Tb; % Node n
21
22    for i=2:n-1
23        A(i,i-1)=-lambda/dx; A(i,i)=2*lambda/dx; A(i,i+1)=-lambda/dx;
24        B(i)=0;
25    end
```

where in line 17 we first fill our matrices with zeros. You see in lines 19 and 20 that we start with introducing the elements related to the two boundary control volumes, i.e. $a_{1,1} \Rightarrow$`A(1,1)`, $b_1 \Rightarrow$`B(1)`, $a_{n,n} \Rightarrow$`A(n,n)` and $b_n \Rightarrow$`B(n)`. Then, we use a `for` loop from the 2nd to the (n-1)-th row to populate the nonzero elements of the matrices. Line 24 is not really necessary because all the elements of **B**, except the first and last, are already zero owing to line 17. After this step, you may potentially take a look at your **A** and **B** matrices by typing `A` or `B` in the command window. We are expecting that the first/last rows of **A** have 1 on the diagonal and all 0 elsewhere; the 2nd to one-to-last rows will show $2\lambda/\delta x = 16000$ on the main diagonal and $-\lambda/\delta x = -8000$ at its sides; **B** should simply exhibit $T_a = 300$ and $T_b = 320$ as first/last entries, and 0 otherwise.

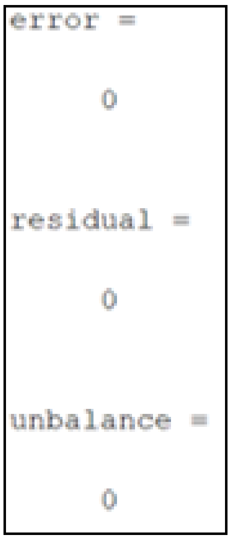
At this point, we are ready to solve the matricial problem $\mathbf{A} \times \mathbf{T} = \mathbf{B}$ using Matlab's backslash operator:



```
27 -    T=A\B; % Matlab backslash
28 -    figure('color','w'); plot(x0,T,'rs'); grid on;
29 -    xlabel('x [m]'); ylabel('T [K]')
30 -    hold on
31
32      %%%%% Comparison with theoretical solution
33 -    Tteo=Ta+(Tb-Ta)*x0/L;
34 -    plot(x0,Tteo,'k-')
35 -    legend('backslash','Exact')
36 -    error=mean(abs(T-Tteo')) %%% Need to transpose Tteo
37
38      %%%%% Check of residuals
39 -    residual=sum(abs(B-A*T))/sum(abs(diag(A).*T)) %%% defined as |B-A*T|/|diag(A).T|
40
41      %%%%% Check of energy balance
42 -    num=0; den=0;
43 -    for i=2:n-1
44 -        num=num+abs(-lambda*(T(i)-T(i-1))/dx+lambda*(T(i+1)-T(i))/dx); %%% sum(|q_w-q_e|)
45 -        den=den+abs(lambda*(T(i)-T(i-1))/dx); %%% sum(|q_w|)
46 -    end
47 -    unbalance=num/den*100 %%% error=sum(|q_w-q_e|)/sum(|q_w|)*100 [%]
```

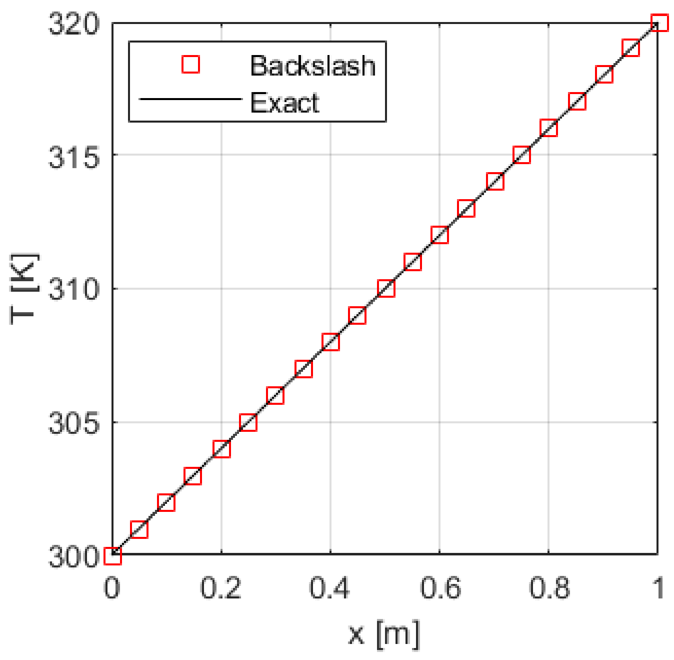
where line 27 is the very last line involved in the solution of the problem. Lines from 28 on have to do with the postprocessing. We can now go through the checklist outlined in Sec. 9:

- Is the solution bounded between the boundary values? Lines 28-35 generate the figure shown in Fig. 11(b), where the numerical and exact solutions are compared. We can conclude that yes, the solution seems to be bounded between $T_a = 300$ and $T_b = 320$.
- Are the boundary conditions respected? We can click on the boundary points in the matlab



```
error =

     0


residual =

     0


unbalance =

     0
```

(a)



(b)

Figure 11: Worked example 1: (a) Output on the command window and (b) figure generated comparing the numerical and exact solution.

figure (or output **T** in the command window) and we see that the boundary values match the boundary conditions.

- Comparison with a known solution. In this simple case, the problem has analytical solution, Eq. (49), which is plotted in Fig. 11(b). Numerical and theoretical solutions seem to match quite well. Line 36 calculates an average error which is then displayed on screen, see Fig. 11(a), and the error is actually zero. This is a very special case, that happens because the solution of our problem is very simple.
- Are the residuals sufficiently small? Equation (46) for the calculation of the residuals is implemented in line 39, where the one-norm (sum of absolute values of the elements of a vector) is chosen to compute the absolute values of the vectors at both numerator and denominator. The output on screen shows that the error in zero. Again, this is a very special case, where the solution of the problem is very simple.
- Is the energy balance respected? The calculation of the overall unbalance, based on Eq. (48), is performed in lines 42-47, the output is zero. Note that the `for` loop in line 43 runs from `i=2:n-1`, thus for simplicity we are discarding the first and last control volumes, where one of the neighbours is missing.
- There is no point in calculating the convergence order of the solution. The exact solution of the problem is a linear function, Eq. (49), and therefore any derivative of $T(x)$ above the first-order is zero. As such, the truncation error associated with the second-order discretisation of dT/dx, $\mathcal{O}[(\delta x)^2 T''']$, is zero, which explains why the deviation between numerical and analytical solution is exactly zero. Repeating the simulation for different values of n would simply yield solutions with zero errors. It will not be so in the next worked example.

10.2 Worked example 2: Dirichlet boundary conditions, nonzero source term

Repeat the previous example, but this time with the nonzero source term $\bar{S} = S_C + S_P T_P$, with $S_C = 5000\,\mathrm{W/m^3}$ and $S_P = -100\,\mathrm{W/(m^3K)}$. The ODE governing the problem is now:

$$\lambda T'' + S_C + S_P T = 0, \tag{53}$$

with $T'' = d^2T/dx^2$. For Dirichlet boundary conditions, $T(x=0) = T_a$ and $T(x=L) = T_b$, this has the following analytical solution:

$$T(x) = c_1 e^{\mu_1 x} + c_2 e^{\mu_2 x} - \frac{S_C}{S_P}, \tag{54}$$

with:

$$\mu_{1,2} = \pm\sqrt{-\frac{S_P}{\lambda}}, \quad c_1 = \frac{T_b - \left(\frac{S_C}{S_P} + T_a\right) e^{\mu_2 L} + \frac{S_C}{S_P}}{e^{\mu_1 L} - e^{\mu_2 L}}, \quad c_2 = T_a + \frac{S_C}{S_P} - c_1. \tag{55}$$

Note that now the theoretical solution of the ODE is an exponential function of x, which has nonzero derivatives at all orders, and hence we are expecting a nonzero truncation error when using a central-difference discretisation scheme for the derivatives in our numerical method, unlike the previous example.

Solution

Compared to the previous example, the only differences in **A** and **B** are that now they must include the contributions due to the nonzero source term. The geometry is unchanged, so refer to the previous exercise for the definition of δx and Δx. The entries for the first control volume do not change:

$$\mathbf{CV_1})\quad a_{1,P}T_{1,P} + a_{1,E}T_{1,E} = b_1 \Rightarrow a_{1,1}T_1 + a_{1,2}T_2 = b_1, \tag{56}$$

with $a_{1,1} = 1$, $a_{1,2} = 0$ and $b_1 = T_a$. For any internal control volume:

$$\mathbf{CV_i})\quad a_{i,W}T_{i,W} + a_{i,P}T_{i,P} + a_{i,E}T_{i,E} = b_i \Rightarrow a_{i,i-1}T_{i-1} + a_{i,i}T_i + a_{i,i+1}T_{i+1} = b_i, \tag{57}$$

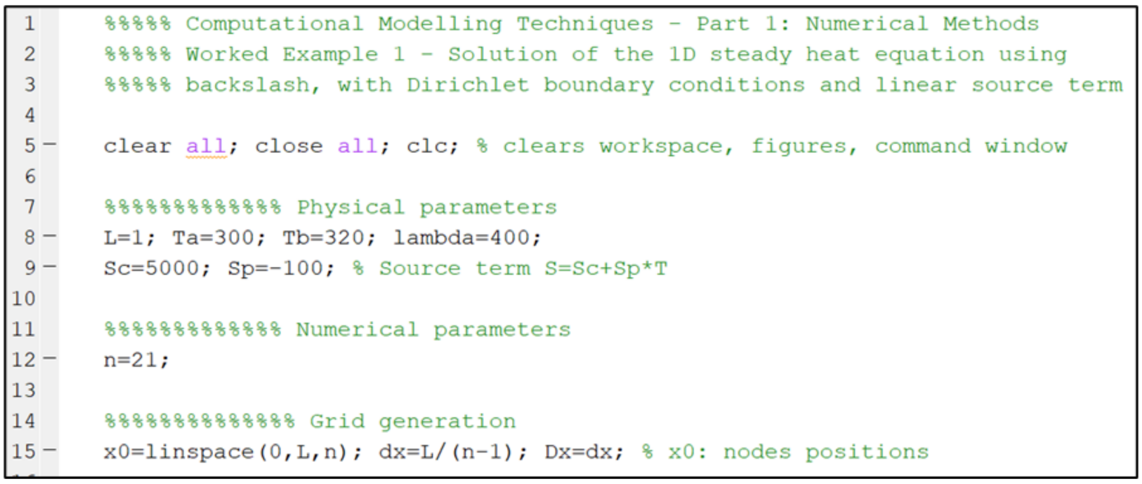
where the coefficients now are, Eq. (26), $a_{i,i-1} = -\lambda/\delta x$, $a_{i,i} = 2\lambda/\delta x - S_P\Delta x$, $a_{i,i+1} = -\lambda/\delta x$ and $b_i = S_C\Delta x$. So, you can see that the only changes compared to the previous exercise are in $a_{i,i}$ and b_i, which now embed the nonzero source term coefficients. The entries for the last control volume are unchanged:

$$\mathbf{CV_n})\quad a_{n,W}T_{n,W} + a_{n,P}T_{n,P} = b_n \Rightarrow a_{n,n-1}T_{n-1} + a_{n,n}T_n = b_n, \tag{58}$$

with $a_{n,n} = 1$, $a_{n,n-1} = 0$ and $b_n = T_b$. Therefore, the matrices will look like:

$$\mathbf{A} = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & -\frac{\lambda}{\delta x} & 2\frac{\lambda}{\delta x} - S_P\Delta x & -\frac{\lambda}{\delta x} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{vmatrix}, \quad \mathbf{B} = \begin{vmatrix} T_a \\ S_C\Delta x \\ \vdots \\ S_C\Delta x \\ T_b \end{vmatrix}.$$

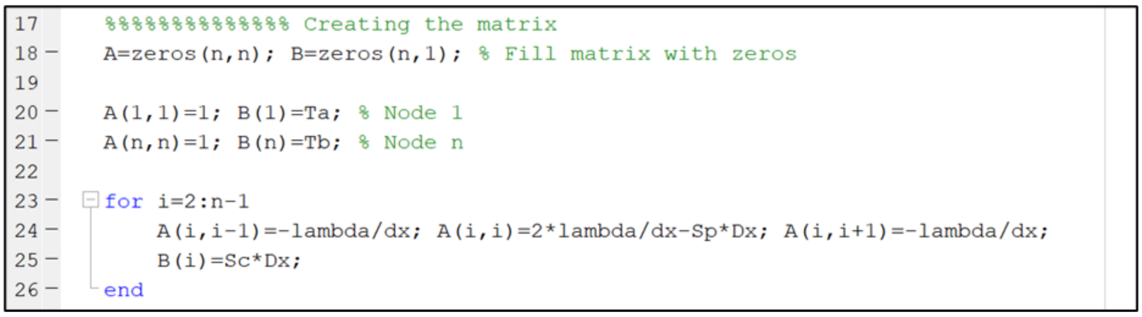
Let's now take a look at our Matlab code to solve this problem:



```
1     %%%%% Computational Modelling Techniques - Part 1: Numerical Methods
2     %%%%% Worked Example 1 - Solution of the 1D steady heat equation using
3     %%%%% backslash, with Dirichlet boundary conditions and linear source term
4
5 -   clear all; close all; clc; % clears workspace, figures, command window
6
7     %%%%%%%%%%%%%% Physical parameters
8 -   L=1; Ta=300; Tb=320; lambda=400;
9 -   Sc=5000; Sp=-100; % Source term S=Sc+Sp*T
10
11    %%%%%%%%%%%%%% Numerical parameters
12 -  n=21;
13
14    %%%%%%%%%%%%%%% Grid generation
15 -  x0=linspace(0,L,n); dx=L/(n-1); Dx=dx; % x0: nodes positions
```

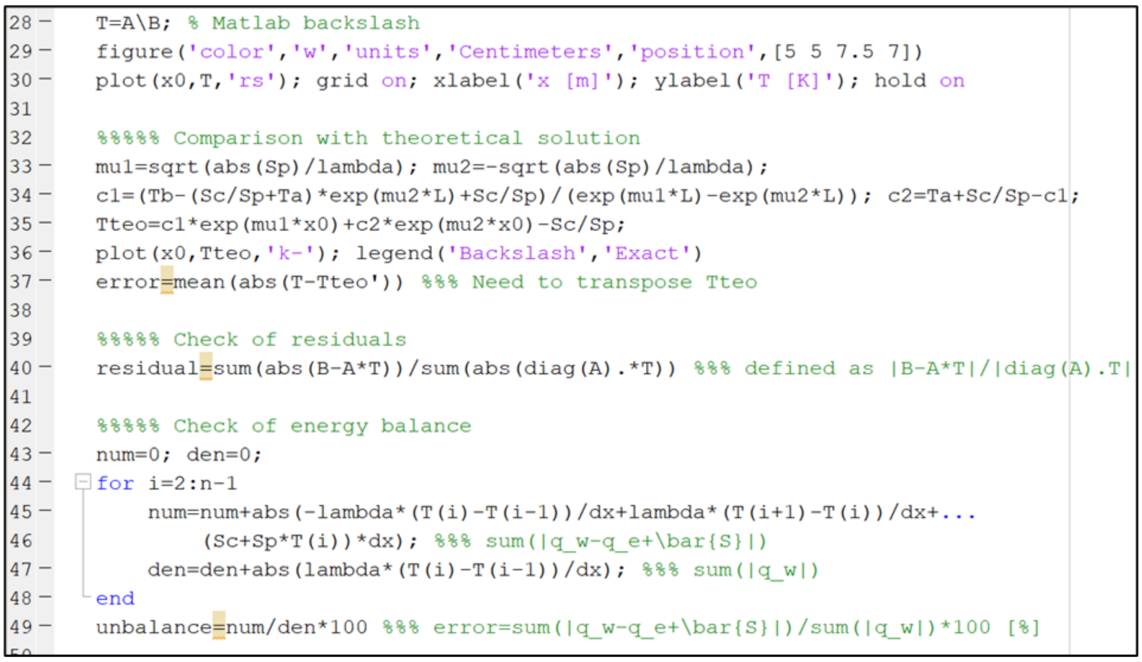
This initial part is essntially unchanged, you see that in line 9 now we define the constants involved in the source term, and in line 15 we add the definition of Δx (`Dx`), which actually refers to internal control volumes, because the boundary control volumes are half of this size. However, because of the Dirichlet boundary conditions, we do not need the width of the boundary control volumes, and therefore here we simply define Δx based on the internal control volumes. But be careful in the case you want to use this code for Neumann boundary conditions, because in that case you'll need to adapt it according to Eq. (33).

Let's now create and fill the matrix **A** and vector **B**:



```
17      %%%%%%%%%%%%%% Creating the matrix
18 -    A=zeros(n,n); B=zeros(n,1); % Fill matrix with zeros
19
20 -    A(1,1)=1; B(1)=Ta; % Node 1
21 -    A(n,n)=1; B(n)=Tb; % Node n
22
23 -    for i=2:n-1
24 -        A(i,i-1)=-lambda/dx; A(i,i)=2*lambda/dx-Sp*Dx; A(i,i+1)=-lambda/dx;
25 -        B(i)=Sc*Dx;
26 -    end
```

This is not much different from the past example, but now you can see in lines 24 and 25 that the contributions of the source terms have been included in `A(i,i)` and `B(i)`. After this step, you may potentially take a look at your **A** and **B** matrices by typing `A` or `B` in the command window. We are expecting that the first/last rows of **A** have 1 on the diagonal and all 0 elsewhere; the 2nd to one-to-last rows will show $2\lambda/\delta x - S_P\Delta x = 16005$ on the main diagonal and $-\lambda/\delta x = -8000$ at its sides; **B** has $T_a = 300$ and $T_b = 320$ as first/last entries, and $S_C\Delta x = 250$ otherwise. At this point, we are ready to solve the matricial problem $\mathbf{A} \times \mathbf{T} = \mathbf{B}$ using Matlab's backslash operator:



```
28 -    T=A\B; % Matlab backslash
29 -    figure('color','w','units','Centimeters','position',[5 5 7.5 7])
30 -    plot(x0,T,'rs'); grid on; xlabel('x [m]'); ylabel('T [K]'); hold on
31
32      %%%%% Comparison with theoretical solution
33 -    mu1=sqrt(abs(Sp)/lambda); mu2=-sqrt(abs(Sp)/lambda);
34 -    c1=(Tb-(Sc/Sp+Ta)*exp(mu2*L)+Sc/Sp)/(exp(mu1*L)-exp(mu2*L)); c2=Ta+Sc/Sp-c1;
35 -    Tteo=c1*exp(mu1*x0)+c2*exp(mu2*x0)-Sc/Sp;
36 -    plot(x0,Tteo,'k-'); legend('Backslash','Exact')
37 -    error=mean(abs(T-Tteo')) %%% Need to transpose Tteo
38
39      %%%%% Check of residuals
40 -    residual=sum(abs(B-A*T))/sum(abs(diag(A).*T)) %%% defined as |B-A*T|/|diag(A).T|
41
42      %%%%% Check of energy balance
43 -    num=0; den=0;
44 -    for i=2:n-1
45 -        num=num+abs(-lambda*(T(i)-T(i-1))/dx+lambda*(T(i+1)-T(i))/dx+...
46              (Sc+Sp*T(i))*dx); %%% sum(|q_w-q_e+\bar{S}|)
47 -        den=den+abs(lambda*(T(i)-T(i-1))/dx); %%% sum(|q_w|)
48 -    end
49 -    unbalance=num/den*100 %%% error=sum(|q_w-q_e+\bar{S}|)/sum(|q_w|)*100 [%]
```

The chunk of code above is very similar to what has been seen for the previous example. Line 28 finds the discrete temperature solution, which is plotted and compared with the theoretical solution, Eq. (55), using lines 29-36; the output figure is displayed in Fig. 12(b), where differences between numerical and exact solutions are negligible. The average error, calculated in line 37 and output on screen (Fig. 12(a)) is very little, on the order of 10^{-4} K. Note that the absolute value of the error is not a good indicator, it is always better to take a relative error, as example taking a boundary temperature as a reference value; in this case, this would make $10^{-4}/T_a \approx 10^{-6} - 10^{-7}$, which is anyway very small. Residuals are very small as well, which indicates that the linear system is solved accurately. Lines 43-49 compute the overall energy balance of the system. Owing to the nonzero source term, we cannot use Eqs. (47) or (48) as done in the previous example, but we must rewrite the energy balance of the control volume to account for the presence of the source term. From Eq. (23):

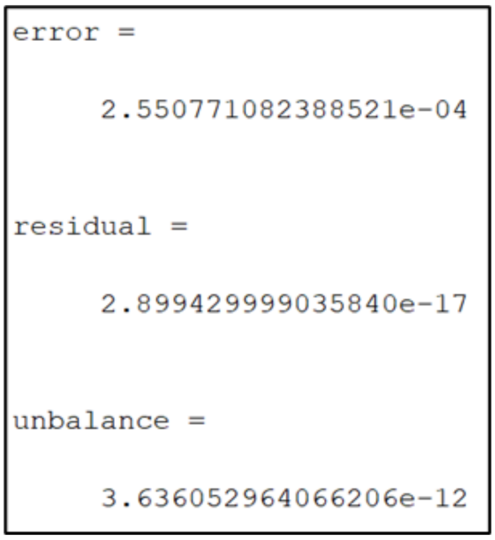
$$q_w - q_e + (S_C + S_P T_P)\,\Delta x = 0 \Rightarrow -\left[\lambda \frac{dT}{dx}\right]_w + \left[\lambda \frac{dT}{dx}\right]_e + (S_C + S_P T_P)\,\Delta x = 0 \tag{59}$$

$$\Rightarrow -\lambda_w \frac{T_P - T_W}{\delta_{x_w}} + \lambda_e \frac{T_E - T_P}{\delta_{x_e}} + (S_C + S_P T_P)\,\Delta x = 0, \tag{60}$$

which is the equation coded in lines 45-46 to calculate the numerator of the expression for the overall unbalance. The denominator is an arbitrary reference, which is taken to be the same as for the previous exercise. Again, in the `for` loop in line 44 we are discarding the first and last control volumes, where one of the neighbours is missing.

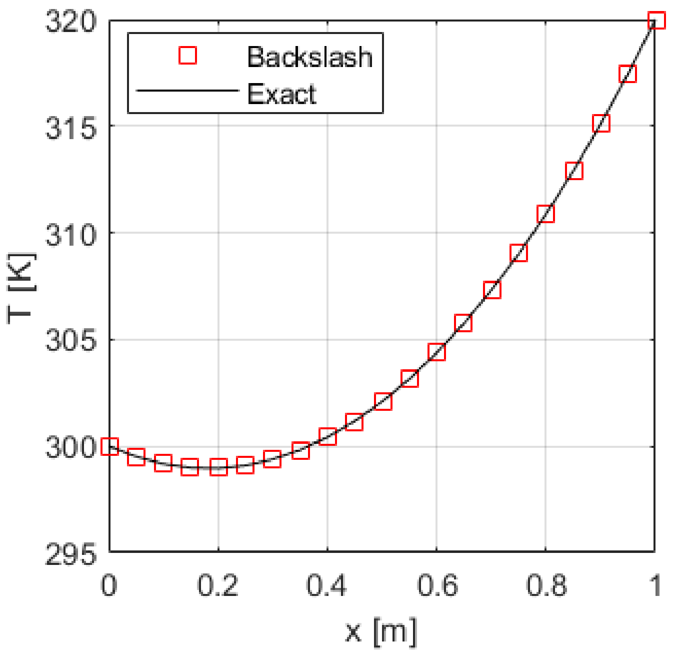
The overall unbalance, reported in Fig. 12(a), is very small, which is a very good result. This means that the overall energy of the system is well preserved by the solution method and algorithm.

Now, we have verified that our solution is quite good, because the deviation with the exact solution is very small. But we have done this for one value of n only, what does happen to our solution (or to the error) when we change the number of nodes? Intuitively, we expect the error to reduce as we increase n and thus decrease the grid spacing δx. Let's rerun the simulation for different values of n. We can do this by adding the following chunk of code:



```
error =

     2.550771082388521e-04


residual =

     2.899429999035840e-17


unbalance =

     3.636052964066206e-12
```

(a)



(b)

Figure 12: Worked example 2: (a) Output on the command window and (b) figure generated comparing the numerical and exact solution.

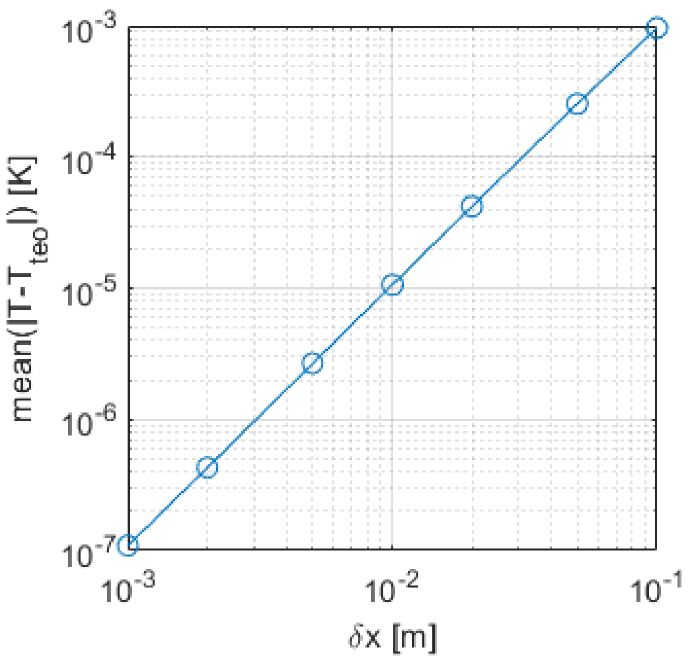
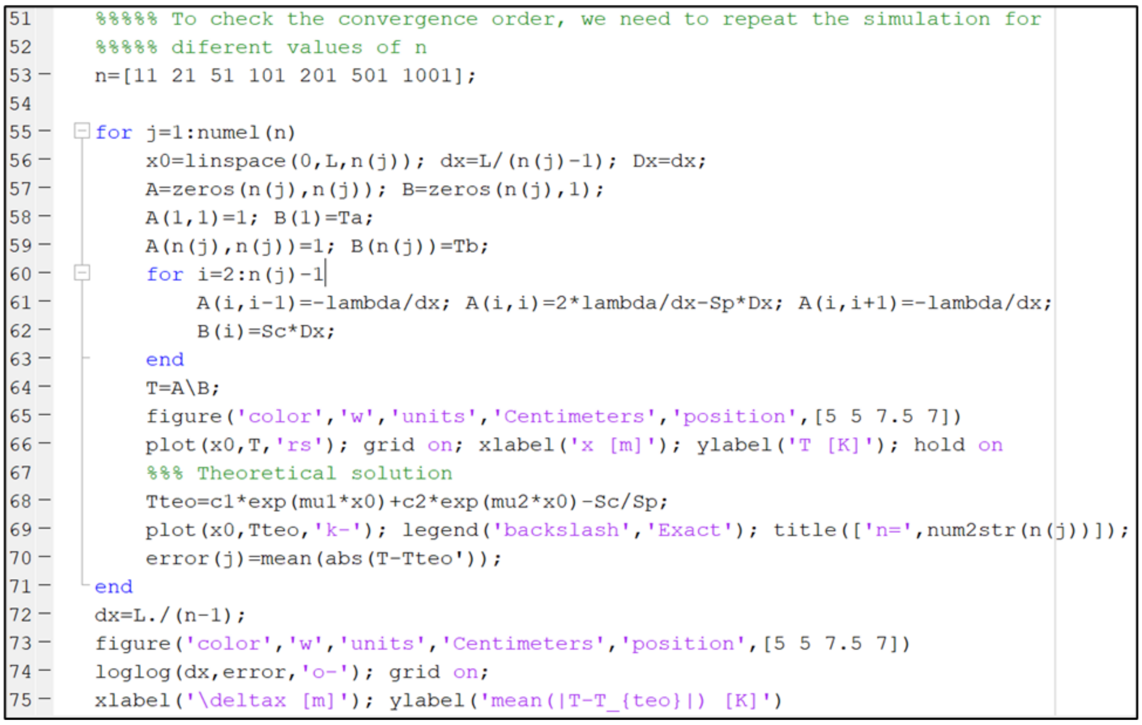


Figure 13: Worked example 2: analysis of the convergence of the solution error upon refinement of the mesh.



```
51      %%%%% To check the convergence order, we need to repeat the simulation for
52      %%%%% diferent values of n
53 -    n=[11 21 51 101 201 501 1001];
54
55 -    for j=1:numel(n)
56 -        x0=linspace(0,L,n(j)); dx=L/(n(j)-1); Dx=dx;
57 -        A=zeros(n(j),n(j)); B=zeros(n(j),1);
58 -        A(1,1)=1; B(1)=Ta;
59 -        A(n(j),n(j))=1; B(n(j))=Tb;
60 -        for i=2:n(j)-1
61 -            A(i,i-1)=-lambda/dx; A(i,i)=2*lambda/dx-Sp*Dx; A(i,i+1)=-lambda/dx;
62 -            B(i)=Sc*Dx;
63 -        end
64 -        T=A\B;
65 -        figure('color','w','units','Centimeters','position',[5 5 7.5 7])
66 -        plot(x0,T,'rs'); grid on; xlabel('x [m]'); ylabel('T [K]'); hold on
67          %%% Theoretical solution
68 -        Tteo=c1*exp(mu1*x0)+c2*exp(mu2*x0)-Sc/Sp;
69 -        plot(x0,Tteo,'k-'); legend('backslash','Exact'); title(['n=',num2str(n(j))]);
70 -        error(j)=mean(abs(T-Tteo'));
71 -    end
72 -    dx=L./(n-1);
73 -    figure('color','w','units','Centimeters','position',[5 5 7.5 7])
74 -    loglog(dx,error,'o-'); grid on;
75 -    xlabel('\deltax [m]'); ylabel('mean(|T-T_{teo}|) [K]')
```

where we are testing 7 different values of n, see line 53, which give δx from $\delta x = 0.1\,\text{m}$ ($n = 11$) to $\delta x = 0.001\,\text{m}$ ($n = 1001$). The entire process of grid generation (line 56), filling the matrix (lines 57-63), solution (line 64), plotting and comparison with the exact solution (lines 65-70) is now done within an external `for` loop that repeats the simulation for each desired value of n. The different plots comparing numerical and exact solutions for each n will not emphasise any difference, as the two are very close already for very coarse meshes. Lines 72-75 generate a summary figure for all values of n, Fig. 13, that plots the average error as a function of the nodes spacing, in log-log coordinates. By inspection of Fig. 13, we immediately notice that the error decreases as δx decreases, which is a first sanity check for our solution. But there is more than that. You see that the error follows a linear trend with δx in log-log coordinates, which suggests an exponential dependence of the kind $error \sim (\delta x)^k$. The value of k is easily deduced from Fig. 13: when δx is reduced by a factor 10^2 (from 0.1 to 0.001), the error decreases by a factor 10^4. This emphasises that $k = 2$, $error \sim (\delta x)^2$, which means that the numerical solution converges to the exact solution with the second-order with respect to the mesh size δx. Was this expected? Yes, indeed, this is the result of the central-difference discretisation scheme used for the first-order derivative, see Sec. 4.1. The central-difference scheme is obtained by calculating the first-order derivative starting from a Taylor expansion truncated at the second-order term, and Eq. (20) shows that the associated truncation error is $\mathcal{O}\left[(\delta x)^2\right]$. This coincides with what we see in Fig. 13, and enables us to positively tick the last bullet point of the verification of the accuracy of the numerical solution, see Sec. 9.

10.3 Worked example 3: Neumann boundary conditions, nonzero source term

Repeat Worked example 1, but this time with a Neumann boundary condition at $x = 0$, say $q = 10^4\,\mathrm{W/m^2}$. Follow the procedure illustrated in Section 4.4.2 to write the correct coefficients for the boundary node. You can try to solve this problem first with a zero source term, and then with a nonzero, temperature-independent source term $\bar{S} = S_C$ with $S_C = 50000\,\mathrm{W/m^3}$. In both cases, there exists an analytical solution, derived here. The ODE governing the problem is:

$$\lambda T'' + S_C = 0, \tag{61}$$

to be solved with boundary conditions $T'(x = 0) = -q/\lambda$ and $T(x = L) = T_b$. The solution can be obtained by integrating twice the equation above and finding the two integration constants thanks to the boundary conditions, this leads to:

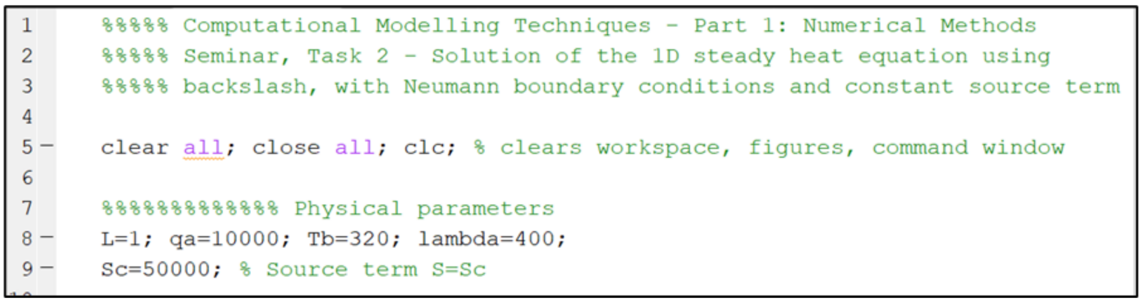
$$T(x) = T_b + \frac{q}{\lambda}(L - x) + \frac{S_C}{2\lambda}(L^2 - x^2), \tag{62}$$

which is valid also in the case that $S_C = 0$.

Solution

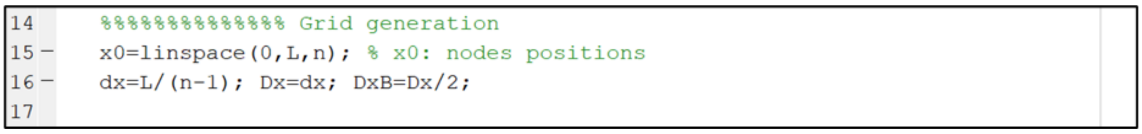
We show the solution algorithm for the general case in which $S = S_C = 50000\,\mathrm{W/m^3}$, the case with $S = 0$ can be solved simply setting $S_C = 0$ in the same code. The starting code can be the same as that of Worked example 1, to which we must add the new boundary condition and the nonzero source S_C; or that of Worked example 2, tho which we must add the new boundary condition and set $S_P = 0$.

We begin with defining the physical and numerical parameters:



```
1      %%%%% Computational Modelling Techniques - Part 1: Numerical Methods
2      %%%%% Seminar, Task 2 - Solution of the 1D steady heat equation using
3      %%%%% backslash, with Neumann boundary conditions and constant source term
4
5 -    clear all; close all; clc; % clears workspace, figures, command window
6
7      %%%%%%%%%%%%%% Physical parameters
8 -    L=1; qa=10000; Tb=320; lambda=400;
9 -    Sc=50000; % Source term S=Sc
```

with line 8, `qa=10000`, that now defines the heat flux at the $x = 0$ boundary. No new code is necessary with regards to the grid generation:

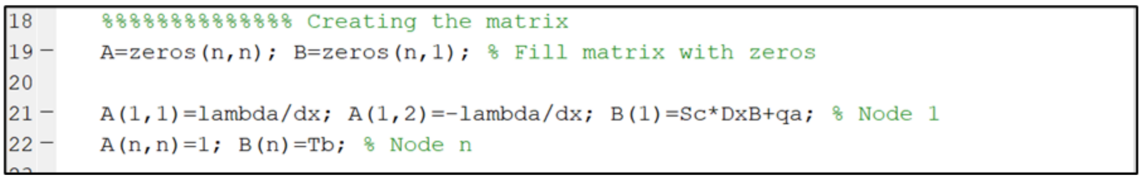


```
14     %%%%%%%%%%%%%%% Grid generation
15 -   x0=linspace(0,L,n); % x0: nodes positions
16 -   dx=L/(n-1); Dx=dx; DxB=Dx/2;
17
```

When creating the matrix, we need a new set of coefficients for the control volume 1, now subjected to a Neumann condition. The discretisation equation for the leftmost control volume subjected to a Neumann condition, Eq. (33) in this document, translates into:

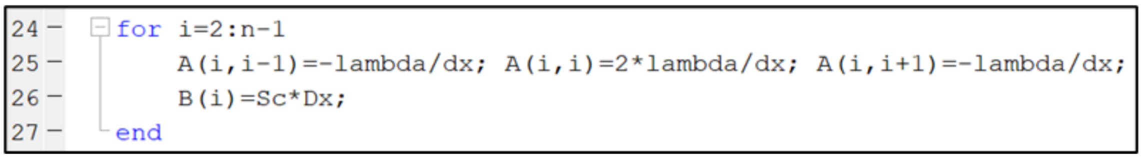
$$a_{1,1} = \frac{\lambda}{\delta x}, \quad a_{1,2} = -\frac{\lambda}{\delta x}, \quad b_1 = S_C \Delta x_B + q_a, \tag{63}$$

which coincides with the modified line 21:



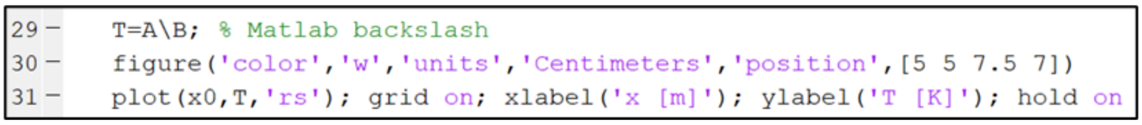
```
18     %%%%%%%%%%%%%%% Creating the matrix
19 -   A=zeros(n,n); B=zeros(n,1); % Fill matrix with zeros
20
21 -   A(1,1)=lambda/dx; A(1,2)=-lambda/dx; B(1)=Sc*DxB+qa; % Node 1
22 -   A(n,n)=1; B(n)=Tb; % Node n
```

Note that, since $S_P = 0$, this is not included into the expression for $a_{i,i}$. The for-cycle to fill the matrix is:



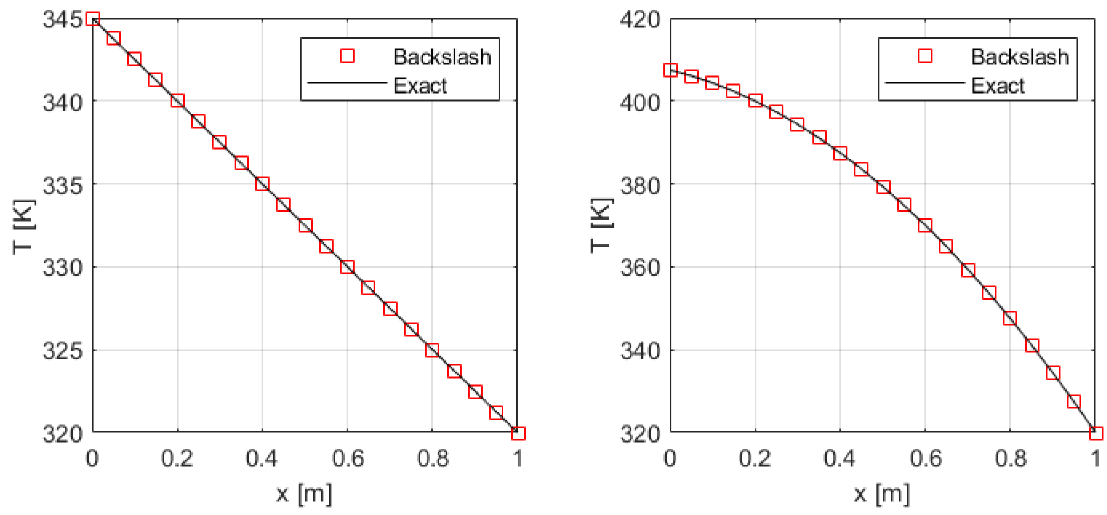
```
24 -   for i=2:n-1
25 -       A(i,i-1)=-lambda/dx; A(i,i)=2*lambda/dx; A(i,i+1)=-lambda/dx;
26 -       B(i)=Sc*Dx;
27 -   end
```

We use the backslash operator to solve the problem and plot the solution:



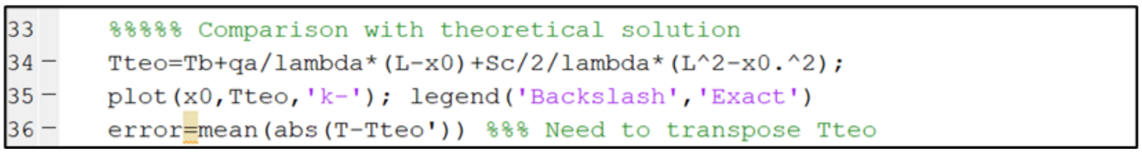
```
29 -   T=A\B; % Matlab backslash
30 -   figure('color','w','units','Centimeters','position',[5 5 7.5 7])
31 -   plot(x0,T,'rs'); grid on; xlabel('x [m]'); ylabel('T [K]'); hold on
```

The analytical solution of this problem was given in Eq. (62) and is implemented as:



(a) (b)

Figure 14: Worked example 3: Numerical and theoretical solutions for (a) $S_C = 0$ and (b) $S_C = 50000\,\mathrm{W/m^3}$.



```
%%%%% Comparison with theoretical solution
Tteo=Tb+qa/lambda*(L-x0)+Sc/2/lambda*(L^2-x0.^2);
plot(x0,Tteo,'k-'); legend('Backslash','Exact')
error=mean(abs(T-Tteo')) %%% Need to transpose Tteo
```

The figure generated by matlab is displayed in Fig. 14(b). The output on the command window is `error=0`. Comments:

- The numerical and theoretical solution match exactly. This should not be surprising. Remember from Sec. 4.1 that the truncation error of the central-difference scheme used to discretise the first-order derivatives in the discretisation equation is proportional to $(\delta x)^2 T'''$, with $T''' = d^3T/dx^3$ being the third-order derivative of the exact solution. However, in the present exercise $T''' = 0$ because the solution is a parabola, see Eq. (62), and therefore the truncation error is zero. As such, the numerical solution matches exactly the theoretical solution.

- The temperature at $x = 0$ is now a result of the imposed heat flux at $x = 0$. Note that, since $dT/dx = -q/\lambda$ and $q_a > 0$, at $x = 0$ we expect $dT/dx < 0$, i.e. the temperature should decrease along x, which is confirmed by the temperature plot in the figure above.

10.4 Suggested exercises

1. Repeat Worked example 1, but this time with a space-dependent source term $S(x) = 10^5 x^2$. You should simply rewrite the part of code of Worked example 1 that defines `B(i)`, paying attention to make this dependent on the position of the i-th node. The ODE with this source term has an analytical solution that can be derived easily. The ODE governing the problem is:

$$\lambda T'' + cx^2 = 0, \qquad (64)$$

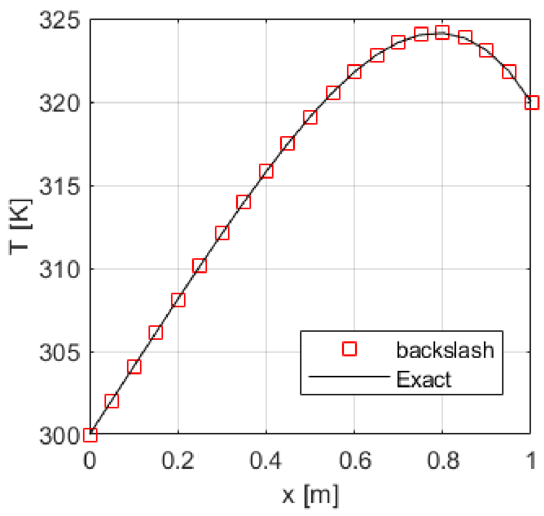


Figure 15: Suggested exercise 1: Numerical and theoretical solutions.

with $c = 10^5$, to be solved with boundary conditions $T(x = 0) = T_a$ and $T(x = L) = T_b$. The solution can be obtained by integrating twice the equation above and finding the two integration constants thanks to the boundary conditions, this leads to:

$$T(x) = T_a + \frac{T_b - T_a}{L}x + \frac{cx}{12\lambda}(L^3 - x^3). \tag{65}$$

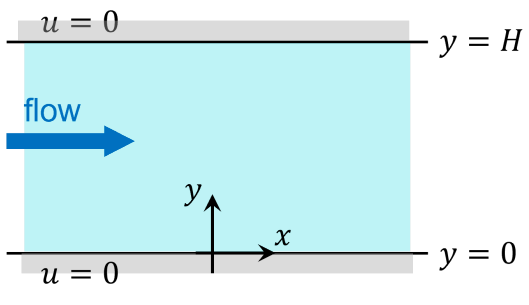
The solution is plotted in Fig. 15.

2. Consider water flowing along the horizontal direction between two parallel plates that form a gap of height H, see schematic in Fig. 16(a). Water flows because it is mobilised by a pump which induces a constant pressure gradient of magnitude dp/dx. Discarding the z-direction, the flow is governed by the two-dimensional Navier-Stokes equations (self-study: MMME2047-Navier-Stokes); however, under the assumption that the flow is fully-developed along the x-direction, and that the pressure gradient is a constant, the flow can be studied as a one-dimensional steady-state diffusion problem, governed by the momentum equation:

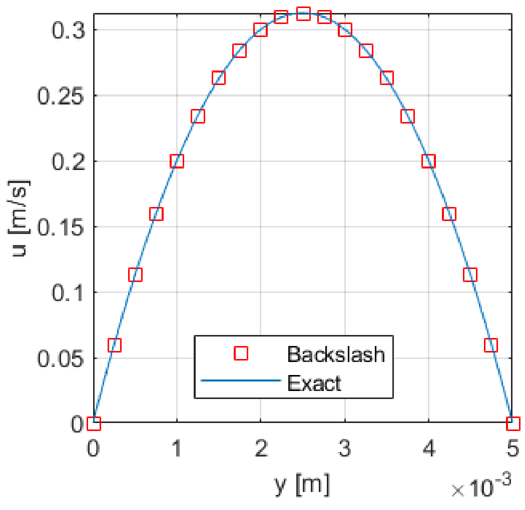
$$\frac{d}{dy}\left(\mu\frac{du}{dy}\right) - \frac{dp}{dx} = 0, \tag{66}$$

where u is the horizontal speed of the water, $-dp/dx$ can be treated as a constant source term (what we used to call S_C), and μ is the dynamic viscosity of the fluid. Note that Eq. (66) is the same diffusion equation that we have solved previously in the case of heat conduction, with u replacing the temperature T, μ replacing the thermal conductivity λ, and $-dp/dx$ being a constant source term S_C (with $S_P = 0$), and therefore you can apply the same numerical methods and solution techniques.

Solve Eq. (66) and compare your solution with the analytical solution for the following set of parameters: $H = 0.005\,\text{m}$, $dp/dx = -100\,\text{Pa/m}$, $\mu = 0.001\,\text{kg/(m s)}$, 21 equidistant nodes. As boundary conditions, the top and bottom walls cause the fluid to stop, and therefore: $u(y = 0) = 0$ and $u(y = H) = 0$. The analytical solution for this problem is a parabolic velocity profile:



(a)



(b)

Figure 16: Suggested exercise 2: (a) Schematic of water flowing in a pipe. (b) Numerical and theoretical solution.

$$u(y) = \frac{1}{2\mu}\frac{dp}{dx}\left(y^2 - yH\right). \tag{67}$$

The numerical and analytical solutions are compared in Fig. 16(b).

References

[1] R. L. Burden and J. Douglas Faires. *Numerical Analysis, 9th edition.* Brooks/Cole, Cengage Learning, Boston, USA, 2010. NUsearch; Download(may not work).

[2] S. C. Chapra and R. P. Canale. *Numerical Methods for Engineers, 7th edition.* McGraw-Hill Education, New York, USA, 2015. NUsearch; Download(may not work).

[3] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow.* Hemisphere Publishing, New York, 1980. NUsearch; Download(may not work).

[4] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes.* Cambridge University Press, Cambridge, England, 2007. NUsearch; Download(may not work).

[5] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics – The Finite Volume Method, 2nd edition.* Pearson Education Limited, Harlow, England, 2007. NUsearch; Download(may not work).