Numerical Analysis – Computational Session

The Heat Equation

Marco Sutti

December 11, 2025

During the theoretical classes and computational sessions, we have encountered several examples of ordinary differential equations (ODEs). The heat equation, instead, is a *partial* differential equation (PDE). A PDE is a differential equation that involves two or more independent variables, an unknown function (dependent on those variables), and partial derivatives of the unknown function with respect to the independent variables. The heat equation is a linear, second-order, parabolic PDE.

In one spatial dimension (x) and time (t), the heat equation reads

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0, \qquad x \in [0,1], \quad t > 0, \tag{0.1}$$

with $u \equiv u(x,t)$, boundary conditions $u(0,t) = u(1,t) = 0$, and an initial condition $u(x,0) = g(x)$.

1 Spatial discretization

For the numerical solution, we begin by discretizing the space $[0,1]$ with a uniform grid of $N+2$ points

$$x_j = j\Delta x, \qquad j = 0, \dots, N+1,$$

where $\Delta x = 1/(N+1)$ is the spatial discretization step.

We approximate $u(x_j,t)$ by $u_j(t)$ considering the 2nd-order central finite difference approximation of the second derivative[1], i.e.,

$$\frac{\partial^2 u}{\partial x^2}(x_j,t) \approx \frac{u(x_{j-1},t) - 2u(x_j,t) + u(x_{j+1},t)}{\Delta x^2} \approx \frac{u_{j-1}(t) - 2u_j(t) + u_{j+1}(t)}{\Delta x^2}.$$

We obtain the following system of N *coupled* ODEs

$$\frac{\partial u_j(t)}{\partial t} = \frac{u_{j-1}(t) - 2u_j(t) + u_{j+1}(t)}{\Delta x^2}, \qquad j = 1, \dots, N, \tag{1.1}$$

with boundary conditions $u_0(t) = u_{N+1}(t) \equiv 0$, for all t. This is the *semi-discrete form* of (0.1) (we have discretized in space only).

[1] The Wikipedia page https://en.wikipedia.org/wiki/Finite_difference_coefficient gives us the coefficients for the central, forward, and backward finite differences, for the first and higher-order derivatives, for several orders of accuracy and with uniform grid spacing.

The semi-discrete form (1.1) can be written in matrix form, setting $\boldsymbol{u}(t) = (u_1(t), \dots, u_N(t))^\top$, we obtain

$$\boldsymbol{u}'(t) = L\boldsymbol{u}(t), \qquad \text{with} \qquad L = \frac{1}{\Delta x^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}. \tag{1.2}$$

This is the semi-discrete form, in matrix form, of the heat equation (0.1).

The initial condition $u(x, 0) = g(x)$ becomes

$$\boldsymbol{u}(0) = \begin{bmatrix} g(x_1) & g(x_2) & \cdots & g(x_N) \end{bmatrix}^\top.$$

The matrix L is called *discretized Laplacian*, as it discretizes the Laplacian, i.e., the second-order partial derivative operator $\partial^2/\partial x^2$. It is a real symmetric matrix, therefore, according to the spectral theorem, it is diagonalizable in an orthonormal basis Q, i.e., $L = Q\Lambda Q^\top$, with real eigenvalues one the diagonal of Λ. Moreover, the eigenvalues of L lie in the open interval $(-4/\Delta x^2, 0)$. This can be shown analytically. You can try it as an exercise.

With the change of variables $\boldsymbol{u}(t) = Q\boldsymbol{v}(t)$, the above system (1.2) can be written as a system of N *decoupled* scalar ODEs:

$$\boldsymbol{v}'(t) = \Lambda\boldsymbol{v}(t), \qquad \boldsymbol{v}(0) = Q^\top\boldsymbol{u}(0). \tag{1.3}$$

In component form, we have

$$\begin{pmatrix} v_1'(t) \\ v_2'(t) \\ \vdots \\ v_{N-1}'(t) \\ v_N'(t) \end{pmatrix} = \begin{bmatrix} \lambda_1 & & & & \\ & \lambda_2 & & & \\ & & \ddots & & \\ & & & \lambda_{N-1} & \\ & & & & \lambda_N \end{bmatrix} \begin{pmatrix} v_1(t) \\ v_2(t) \\ \vdots \\ v_{N-1}(t) \\ v_N(t) \end{pmatrix}.$$

The solution is therefore

$$v_i(t) = e^{\lambda_i t} \cdot v_i(0), \qquad i = 1, \dots, N.$$

Since all eigenvalues are real and less than 0, we also have that the norms of the (continuous) solutions vanish for $t \to \infty$, i.e.,

$$\|\boldsymbol{u}(t)\|_2 = \|\boldsymbol{v}(t)\|_2 \to 0, \qquad t \to \infty.$$

2 Time discretization

Let us now consider the explicit Euler with constant step size method applied to the system (1.2) (and (1.3)) with time discretization step h. We obtain

$$\boldsymbol{u}_{n+1} = (I + hL)\,\boldsymbol{u}_n, \quad \text{for (1.2)}, \qquad \text{and} \qquad \boldsymbol{v}_{n+1} = (I + h\Lambda)\,\boldsymbol{v}_n, \quad \text{for (1.3)},$$

where $\boldsymbol{v}_n = Q^\top\boldsymbol{u}_n$ for all n. Thus, the numerical solution remains bounded if $|R(h\lambda_i)| = |1 + h\lambda_i| \leqslant 1$, for all eigenvalues λ_i, with $\lambda_i \in (-4/\Delta x^2, 0)$. This gives the stability condition

$$h < \frac{1}{2}\,\Delta x^2.$$

We observe that **this condition is very restrictive**, as it requires $h \ll \Delta x$ for the numerical solution to remain bounded, which makes the method very expensive. **The problem is therefore stiff**.

3 Your exercise

Consider the problem (0.1) with the following initial condition

$$g(x) = x^2(1-x), \quad \text{for all } x \in (0,1).$$

1. Using MATLAB, apply the explicit Euler method to solve this problem using the space and time discretizations described in the previous sections, with $N_x = 30$ grid points in space, and $N_t = 1000$ time steps. Consider as final time $T = 2$, and $h = T/N_t$. Use the MATLAB command `diag` to create the discretized Laplacian L (ask me for help or look it up in the documentation). Your function should save the numerical solution at every time step.

2. Plot the norm of the numerical solution versus time. What do you observe?

3. In another figure, plot the stability region of the explicit Euler method.

   - You can use the script `computeRKstabilityRegion` from last week's computational session.
   - As an alternative, for the explicit Euler method, you can simply plot a unitary circle centered at the point $(-1, 0)$.
   - You can also use the technique described on page 4, which solves an ODE for the parametrized border of the stability region (please do it as an exercise).

   On the same figure, plot $h \cdot \lambda_{\min}(L)$ (the smallest eigenvalue of L times h). What do you observe?

4. Try increasing N_t from 1000 to 2000, then 4000. How do the plots change? Why?

5. Now, apply the Dormand–Prince method with adaptive step-size control for the same problem. Modify the function `RK45` from the previous sessions so that it can plot, at every time step, the stability region of the Dormand–Prince method and $h \cdot \lambda_{\min}(L)$. Note that now h is no longer constant, that is why you need to plot $h \cdot \lambda_{\min}(L)$ at every iteration. What do you observe?

This section is a follow-up to the computational session on plotting the stability regions of RK methods. It proposes an alternative method to what we saw last week. If you have time, please do it as an exercise.

4 Plotting Stability Regions via the "ODE Method"

Let $R(z)$ be the stability function of a Runge–Kutta (RK) method. The stability region is defined as

$$\mathcal{S} = \{z \in \mathbb{C} \colon |R(z)| \leq 1\}.$$

The boundary of the stability region is the curve $\partial\mathcal{S}$ such that $|R(z)| = 1$.

4.1 Parametric characterization of the boundary

A way to parametrize the boundary $\partial\mathcal{S}$ is to require

$$R\big(z(t)\big) = e^{\mathrm{i}t}, \qquad t \in [0, 2\pi k],$$

for some integer k (usually $k = 1$), i being the imaginary unit. Differentiating both sides with respect to t gives

$$R'\big(z(t)\big)\, z'(t) = \mathrm{i}\, e^{\mathrm{i}t}.$$

Assuming $R'(z) \neq 0$ along the curve, we obtain the ODE

$$z'(t) = \frac{\mathrm{i}\, e^{\mathrm{i}t}}{R'(z(t))}. \tag{4.1}$$

Thus, the solution $z(t)$ of this ODE traces the boundary of the stability region.

At $t = 0$, we have $e^{\mathrm{i}0} = 1$, hence the initial point must satisfy $R(z_0) = 1$. For many RK methods, $R(0) = 1$, so the initial condition is simply $z(0) = 0$.

4.2 Numerical integration

Now, we need to integrate the ODE (4.1) for $t \in [0, 2\pi k]$ using any suitable complex-valued ODE integrator (e.g., `ode45` in MATLAB). The resulting points $z(t)$ trace the boundary of the stability region.

4.3 Computing the stability function for general RK methods

For an s-stage RK method with RK matrix A and weights $\boldsymbol{b}$, the stability function is

$$R(z) = 1 + z\, \boldsymbol{b}^\top (I - zA)^{-1} \boldsymbol{e},$$

where $\boldsymbol{e} = (1, \ldots, 1)^\top$. If no closed-form expression is available (e.g., Dormand–Prince), evaluate $R(z)$ via this formula.

The derivative may be computed analytically if possible, otherwise numerically, e.g., using centered differences.

4.4 Plotting

Extract the real and imaginary parts of the solution, $z(t) = x(t) + \mathrm{i}\, y(t)$, and plot the parametric curve $(x(t), y(t))$ in the complex plane. This curve approximates the boundary $\partial\mathcal{S}$ of the stability region.