Chapter 4

Numerical solution of the heat equation (Project B)

4.1 Introduction

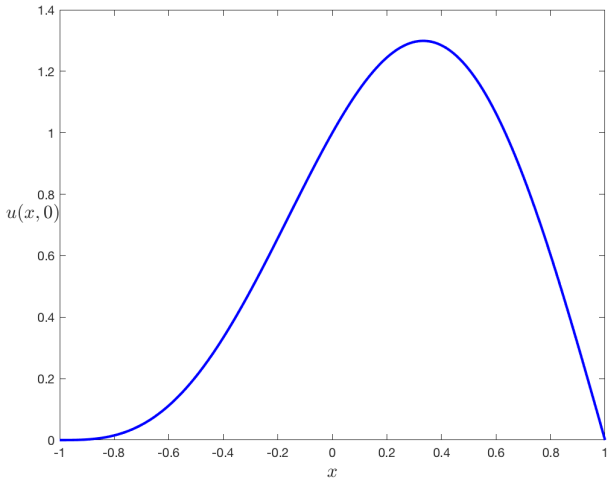
The *heat equation* is a partial differential equation (PDE) which models the diffusion of heat through an object over time. If we assume that temperature varies in only one spatial direction, we have the simplest case of the 1D heat equation. Writing $u(x,t)$ for the temperature at position x and time t, the 1D heat equation is

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \quad P \leq x \leq Q, \;\; 0 \leq t \leq T,$$

where α is the *thermal diffusivity* constant. For simplicity, we will take $\alpha = 1$, $P = -1$ and $Q = 1$.

As one would expect, the solution depends upon the *initial conditions*. We will assume that the initial heat distribution is given by the function

$$u(x,0) = \cos\frac{\pi x}{2} + \frac{1}{2}\sin \pi x.$$



We can then make the solution unique by also imposing *boundary conditions*. We will take the boundary conditions at $x = 0$ and $x = 1$ to be $u(-1,t) = 0$ and $u(1,t) = 0$ for all $0 \leq t \leq T$.

These conditions are often referred to as *Dirichlet boundary conditions*. Summarizing, we want to solve

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \quad -1 \le x \le 1, \;\; 0 \le t \le T, \tag{4.1}$$

subject to

$$\begin{aligned} u(x,0) &= \cos\tfrac{\pi x}{2} + \tfrac{1}{2}\sin \pi x \quad -1 \le x \le 1 \\ u(-1,t) &= u(1,t) = 0 \qquad\qquad 0 \le t \le T. \end{aligned} \tag{4.2}$$

While simple closed form solutions exist for some initial heat distributions $u(x,0)$, there are no simple closed form solutions for general $u(x,0)$. For this reason, it is common to obtain numerical approximations to the solution. The idea is to solve the equation on a discrete grid, with grid spacings of h in the x-direction and k in the t-direction. If T is the maximum time to which we want to compute the solution, setting $h = 2/M$ and $k = T/N$ for some integers M and N, we denote

$$\begin{aligned} x_m &= mh \quad m = 0, \ldots, M \\ t_n &= nk \quad n = 0, \ldots, N, \end{aligned}$$

and we let U_m^n be the solution computed at the gridpoint (x_m, t_n). We want/hope that $U_m^n \approx u(x_m, t_n)$.

The basic idea is to use local *finite difference* approximations for the partial derivatives involved, namely

$$\begin{aligned} \frac{\partial u}{\partial t}(x_m, t_n) &\approx \frac{U_m^{n+1} - U_m^n}{k}, \\ \frac{\partial^2 u}{\partial x^2}(x_m, t_n) &\approx \frac{U_{m+1}^n - 2U_m^n + U_{m-1}^n}{h^2}. \end{aligned} \tag{4.3}$$

Substituting these formulae into (4.1) gives the approximate relation

$$\frac{U_m^{n+1} - U_m^n}{k} = \frac{U_{m+1}^n - 2U_m^n + U_{m-1}^n}{h^2},$$

which, writing $\lambda = k/h^2$, we may rewrite as

$$U_m^{n+1} = U_m^n + \lambda(U_{m+1}^n - 2U_m^n + U_{m-1}^n). \tag{4.4}$$

In other words, the numerical solution at (x_m, t_{n+1}) can be found if we already know the solutions at (x_m, t_n) and $(x_{m\pm1}, t_n)$, allowing us to 'step forward' in time.

If we think of the inner part of the solution (minus the boundary points) at a given time $t = nk$ as a vector $\mathbf{U}^n$, we can express the formula (4.4) using vectors and matrices. Writing

$$\mathbf{U}^n = \begin{bmatrix} U_1^n \\ \vdots \\ U_{M-1}^n \end{bmatrix} \qquad \mathbf{L} = \begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix},$$

the recursion (4.4) becomes

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \lambda \mathbf{L}\,\mathbf{U}^n = (\mathbf{I} + \lambda \mathbf{L})\mathbf{U}^n = \mathbf{A}\,\mathbf{U}^n, \tag{4.5}$$

where $\mathbf{A} = \mathbf{I} + \lambda\mathbf{L}$. Note that the first and last rows of the matrix $\mathbf{L}$ implicity assume the boundary conditions in (4.2). $\mathbf{L}$ is a tridiagonal matrix, which can be efficiently stored in MATLAB as a *sparse matrix*. Repeatedly applying (4.5) generates a sequence of vectors $\{\mathbf{U}^n\}_{n\ge0}$, an approximate solution to the heat equation as we step forward in time.

4.2 Investigating stability

Download the MATLAB script `heat_eqn.m` from the course website and study the code. This function solves the heat equation using the method outlined above, and animates the solution as it evolves over time. Note the use of sparse matrices: you may wish to read a little documentation about sparse matrices, and in particular the `spdiags` and `speye` commands. Note also the use of the `pause` command as a simple animation tool. The function takes three input arguments: the maximum time T, the spatial resolution M, and the parameter λ. These three parameters then determine the time step length k and the number of time steps N.

Exercise 4.1. *Run the code with $T = 1$, $M = 20$, and for (a) $\lambda = 0.3$ and (b) $\lambda = 1$. Describe the behaviour of the numerical solutions.*

For the numerical solution to remain bounded and stable, we require that all of the eigenvalues of the matrix $\mathbf{A} = \mathbf{I} + \lambda\mathbf{L}$ have absolute value less than or equal to 1, which means $\lambda = k/h^2$ has to be suitably small.

Exercise 4.2. *Write a function `stable_test.m` which takes a single input argument λ and outputs `true` if all the eigenvalues of $\mathbf{A} = \mathbf{I} + \lambda\mathbf{L}$ have absolute value less than or equal to 1, and `false` otherwise. Run `stable_test.m` for $\lambda = 0.1, 0.2, 0.3, \ldots, 1$. Roughly how small does λ have to be for the numerical solution to be stable? Hint: the eigenvalues with the six largest absolute values of a sparse real symmetric matrix can be found using the `eigs` command. Continue to take $M = 20$.*

So we see that the value of λ matters for stability, the precise details of which come in a more advanced course.

4.3 Investigating approximation accuracy

Next, we fix λ and investigate the effect of the grid point spacing on the approximation error.

Though this is not always the case, the initial/boundary conditions in (4.2) were selected to ensure there is a simple closed form solution, namely

$$\hat{u}(x,t) = e^{-\frac{\pi^2 t}{4}} \cos\frac{\pi x}{2} + \frac{1}{2} e^{-\pi^2 t} \sin \pi x.$$

Knowing the true solution allows us to assess the accuracy of the method.

Let us write $\hat{\mathbf{U}}^n$ for the true solution vector at time $t = nk$.

Exercise 4.3. *Alter `heat_eqn.m` to animate the true solution as well as the numerical solution. Animate both solutions in the same figure, with the true solution in red to distinguish it from the numerical solution. Add a second output argument `u_exact` which gives the length-$(M+1)$ true solution vector at the final time point. Call the new code `heat_eqn1.m`.*

Next, we need to decide on a measure of approximation error. We will measure approximation error at time $t = nk$ using $\mathcal{E}^n = \|\mathbf{U}^n - \hat{\mathbf{U}}^n\|_\infty$, that is, the maximum absolute value of the error between the approximate solution vector and the true solution vector at $t = nk$. More specifically, we will be interested in the approximation error at $t = T$, the final time point, which we will simply denote by $\mathcal{E}$.

Exercise 4.4. *Fix $\lambda = 1/2$ and $T = 1$. Use your `heat_eqn1.m` code to calculate the final approximation error when $M = 20$.*

Exercise 4.5. *Continue to take $\lambda = 1/2$ and $T = 1$. Write code which loops over spatial step size parameters in the range $M \in \{20, 21, 22, \ldots, 100\}$ and returns the final approximation error $\mathcal{E}$ in each case. You may wish to use a duplicate of* `heat_eqn1.m` *but with the animation suppressed (call it whatever you like). Plot a graph of $\ln \mathcal{E}$ against $\ln h$ (note: $h = 2/M$). Fit a straight line to your graph using* `polyfit` *and hence estimate the gradient. What can you deduce about the relationship between $\mathcal{E}$ and h?*

4.4 A more stable finite difference scheme

We can improve the stability of the algorithm by using a different finite difference approximation for one of the partial derivatives. In particular, we will use

$$\begin{aligned} \frac{\partial u}{\partial t}(x_m, t_n) &\approx \frac{U_m^{n+1} - U_m^n}{k}, \\ \frac{\partial^2 u}{\partial x^2}(x_m, t_n) &\approx \frac{1}{2}\left\{ \frac{U_{m+1}^n - 2U_m^n + U_{m-1}^n}{h^2} + \frac{U_{m+1}^{n+1} - 2U_m^{n+1} + U_{m-1}^{n+1}}{h^2} \right\}. \end{aligned} \tag{4.6}$$

instead of (4.3). Note that the only change is that we approximate the second derivative in x by averaging the approximations at the nth and $(n+1)$th time points, rather than simply using the nth time point. Substituting these formulae into (4.1) as before gives the approximate relation

$$\frac{U_m^{n+1} - U_m^n}{k} = \frac{U_{m+1}^n - 2U_m^n + U_{m-1}^n}{2h^2} + \frac{U_{m+1}^{n+1} - 2U_m^{n+1} + U_{m-1}^{n+1}}{2h^2}, \tag{4.7}$$

which we may rewrite as

$$U_m^{n+1} - \frac{\lambda}{2}(U_{m+1}^{n+1} - 2U_m^{n+1} + U_{m-1}^{n+1}) = U_m^n + \frac{\lambda}{2}(U_{m+1}^n - 2U_m^n + U_{m-1}^n).$$

Using the same vector notation as before, we have

$$\left(\mathbf{I} - \frac{\lambda}{2}\mathbf{L}\right)\mathbf{U}^{n+1} = \left(\mathbf{I} + \frac{\lambda}{2}\mathbf{L}\right)\mathbf{U}^n.$$

But this is an implicit relation: we want to express $\mathbf{U}^{n+1}$ in terms of $\mathbf{U}^n$. Since $\mathbf{I} - \frac{1}{2}\lambda\mathbf{L}$ is invertible, we can invert the formula to get

$$\mathbf{U}^{n+1} = \left(\mathbf{I} - \frac{\lambda}{2}\mathbf{L}\right)^{-1}\left(\mathbf{I} + \frac{\lambda}{2}\mathbf{L}\right)\mathbf{U}^n = \mathbf{B}\mathbf{U}^n,$$

where $\mathbf{B} = \left(\mathbf{I} - \frac{\lambda}{2}\mathbf{L}\right)^{-1}\left(\mathbf{I} + \frac{\lambda}{2}\mathbf{L}\right)$.

Exercise 4.6. *By altering the code of* `heat_eqn.m`*, produce a* MATLAB *function which implements the new method described above; call it* `heat_eqn2.m`*. The function should have the same input and output arguments, and it should generate an animation of the solution as before. Note you do not need to include the true solution this time.*

Exercise 4.7. *Repeat exercise 4.1 for the new method.*

It can be shown that the eigenvalues of $\mathbf{B}$ are never greater than 1, which means that the method is stable for any λ.

4.5 The time-dependent Allen-Cahn Equation

In this final section, we explore adding an additional term to the heat equation. Consider the differential equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \beta(u - u^3) \quad -1 \le x \le 1, \;\; 0 \le t \le T, \tag{4.8}$$

subject to

$$\begin{array}{ll} u(x,0) = \cos\frac{\pi x}{2} + \frac{1}{2}\sin \pi x & -1 \le x \le 1 \\ u(-1,t) = u(1,t) = 0 & 0 \le t \le T. \end{array} \tag{4.9}$$

Here $\beta > 0$ is a constant which can be varied. Using the finite difference approximations for the partial derivatives given in (4.6), we obtain

$$\frac{U_m^{n+1} - U_m^n}{k} = \frac{U_{m+1}^n - 2U_m^n + U_{m-1}^n}{2h^2} + \frac{U_{m+1}^{n+1} - 2U_m^{n+1} + U_{m-1}^{n+1}}{2h^2} + \beta U_m^n - \beta \left(U_m^n\right)^3, \tag{4.10}$$

which is the same as (4.7) except for the additional terms.

Exercise 4.8. *By altering the code of* `heat_eqn2.m`*, produce a* MATLAB *function which numerically solves the time-dependent Allen-Cahn Equation; call it* `allen_cahn.m`*. The function should have the same input arguments as before along with an additional fourth argument for the parameter β, it should have the same output arguments as before, and it should generate an animation of the solution as before.*

Exercise 4.9. *Vary β over the integers $\{1, 2, 3, 4, 5\}$. Describe what you observe about the limiting solution (as $t \to \infty$) in these cases.*