

Mat1062: Introductory Numerical Methods for PDE

Mary Pugh

January 29, 2009

1 Ownership

These notes are the joint property of Rob Almgren and Mary Pugh.

2 Rigorous convergence for heat equation

We have two purposes in doing this: First, it is interesting to know that it is possible to prove convergence rigorously, at least for a simple problem and under the assumption that the initial data is sufficiently nice. Second, the method of proof illustrates the fundamental interplay of consistency and stability: If you make only a small error on each step, and the errors don't amplify, then the final error is small.

We consider the heat equation

$$u_t = Du_{xx} \quad \text{on } -\infty < x < \infty, \quad t > 0,$$

with initial condition $u(x, 0) = u_0(x)$. Let $v(x, t)$ denote the true solution to this problem. (Please suspend your disbelief on why I'm calling the true solution v rather than u .)

We consider nice initial data; specifically we suppose that the function $u_0(x)$ has globally bounded 4th derivative:

$$\text{There is } M < \infty \text{ so that } |u_{0xxxx}(x)| \leq M \quad \text{for all } -\infty < x < \infty.$$

Because v is a solution of $u_t = Du_{xx}$, it follows that v_{xxxx} is also a solution of $u_t = Du_{xx}$. And so the maximum principle applies to v_{xxxx} , resulting in the uniform bound:

$$|v^{(4)}(x, t)| = |v_{xxxx}(x, t)| \leq M \quad \text{for all } x \text{ and all } t \geq 0. \quad (1)$$

Furthermore, since

$$v_{tt} = (v_t)_t = (Dv_{xx})_t = D(v_t)_{xx} = D(Dv_{xx})_{xx} = D^2v_{xxxx},$$

we also have a uniform bound on v_{tt} :

$$|v_{tt}(x, t)| \leq MD^2 \quad \text{for all } x \text{ and all } t \geq 0. \quad (2)$$

The bounds (1) and (2) will be used in proving convergence.

Now introduce a grid with space step h and time step k , and write $v_j^n = v(jh, nk)$ for samples of the true continuous solution on the grid. As part of proving consistency, we used the local asymptotic expression

$$\frac{1}{k} \left(v_j^{n+1} - v_j^n \right) \sim v_t(jh, nk) + \frac{1}{2}kv_{tt}(jh, nk) + \mathcal{O}(k^2), \quad k \rightarrow 0.$$

This means that there is some $K > 0$ such that

$$\left| \frac{1}{k} \left(v_j^{n+1} - v_j^n \right) - v_t(jh, nk) \right| \leq k \|v_{tt}\|_\infty$$

for all $k < K$. (See the explanation why at the end of this section.) But since we have the uniform bound (2) on $|v_{tt}|$, we have assert

$$\left| \frac{1}{k} \left(v_j^{n+1} - v_j^n \right) - v_t(jh, nk) \right| \leq kMD^2 \quad (3)$$

for all j, n, h and all $k < K$. Rather than using asymptotic expansions, this statement can also be proved using the Mean Value Theorem of calculus. It is not a particularly surprising conclusion, but I wanted to state it rigorously so you see how the bound on the second t -derivative is used.

Similarly, using the bound (1) on $|v_{xxxx}|$, we can show that there is a $H > 0$ so that

$$\left| \frac{1}{h^2} \left(v_{j-1}^n - 2v_j^n + v_{j+1}^n \right) - v_{xx}(jh, nk) \right| \leq h^2M \quad (4)$$

for all j, n, k and all $h < H$.

Now let us turn to our discrete computation. For any particular values of h and k , we define a set of numbers u_j^n for $-\infty < j < \infty$ and $n \geq 0$ by the rule

$$u_j^0 = u_0(jh), \quad \text{for } -\infty < j < \infty,$$

and

$$\begin{aligned} u_j^{n+1} &= u_j^n + \lambda(u_{j-1}^n - 2u_j^n + u_{j+1}^n) \\ &= (1 - 2\lambda)u_j^n + \lambda(u_{j-1}^n + u_{j+1}^n) \end{aligned}$$

for $-\infty < j < \infty$ and $n \geq 0$, where

$$\lambda = \frac{Dk}{h^2}.$$

This is the formula we implement in our computer program.

We are interested in whether the numbers u_j^n converge to the true solution $v(x, t)$. The rule that generated u_j^n depends on both h and k . So the first thing we do is to choose a “refinement path”. That is, we specify a relationship between h and k , so that we have a single parameter tending to zero. We choose h to be given in terms of k as

$$k = \lambda h^2/D \quad \implies \quad h = \sqrt{\frac{kD}{\lambda}} \quad (5)$$

where λ is fixed. If $\lambda \leq 1/2$ then taking $(h, k) \rightarrow (0, 0)$ along this path results in a convergent method. For Crank-Nicolson, Backward Euler, or one of the θ -methods with $\theta \geq 1/2$, we can choose $k = \mu h$, with μ fixed and get convergence. But this refinement path would not lead to convergence for Forward Euler.

With this choice, the u_j^n are determined by the single parameter k . Convergence then means the following thing. Pick any particular position and time (x, t) . For each value of k and hence of h , there will be a particular pair of indices $(j_*(k), n_*(k))$ so that the corresponding grid point (j_*h, n_*k) is the closest grid point to (x, t) . As $h, k \rightarrow 0$, this closest grid point will get closer and closer to (x, t) . We want to show that

$$u_{j_*(k)}^{n_*(k)} \longrightarrow v(x, t) \quad \text{as } h \rightarrow 0, \quad \text{where } j_*h \rightarrow x \text{ and } n_*k \rightarrow t.$$

As $k \rightarrow 0$, j_* and n_* will $\rightarrow \infty$, since you need more and more grid points to reach the same location in space and time.

We must prove this convergence without knowing the exact solution $v(x, t)$ (otherwise we would have no need to construct a numerical method). The amazing fact is that we can do this using only the fact that v satisfies the PDE, which gives *consistency*, and using the fact that the discrete scheme for u is *stable*.

First, we recall the **local truncation error**

$$\begin{aligned}\epsilon_j^n &= v_j^{n+1} - (1 - 2\lambda)v_j^n - \lambda(v_{j-1}^n + v_{j+1}^n) \\ &= k \left(\frac{1}{k} (v_j^{n+1} - v_j^n) - \frac{D}{h^2} (v_{j-1}^n - 2v_j^n + v_{j+1}^n) \right)\end{aligned}$$

This is the amount by which the true solution fails to satisfy the difference formula. Because Forward Euler is consistent and we have the bounds (1) and (2), we know that

$$\left| \epsilon_j^n - k \left(v_t(jh, nk) - Dv_{xx}(jh, nk) \right) \right| \leq k^2 MD^2 + kh^2 MD$$

(we suppose that $h < H$ and $k < K$ so that the bounds hold). But since v satisfies the PDE $u_t = Du_{xx}$, this gives

$$|\epsilon_j^n| \leq k^2 MD^2 + kh^2 MD \quad (6)$$

Furthermore, since $k = \lambda h^2/D$, we may write this as

$$|\epsilon_j^n| \leq h^2 k (\lambda MD + MD) = h^2 k MD (\lambda + 1)$$

The important part is that the right side is asymptotically smaller than k : we must make an error per step that is smaller than the time we spend, otherwise we have no hope of the cumulative error being small.

We may turn around the definition of the truncation error and write

$$v_j^{n+1} = (1 - 2\lambda)v_j^n + \lambda(v_{j-1}^n + v_{j+1}^n) + \epsilon_j^n.$$

This is the difference formula we use to generate the u_j^n , plus a small correction. That is, ϵ_j^n is the correction we should add into the difference scheme at each step to get the correct solution. From the consistency analysis, we know that this correction is uniformly small (6).

Now let us define the **actual error**

$$e_j^n = u_j^n - v_j^n.$$

We have $e_j^0 = 0$ for all j , since we took exact initial data. And e_j^n satisfies

$$\begin{aligned}e_j^{n+1} &= u_j^{n+1} - v_j^{n+1} \\ &= (1 - 2\lambda)u_j^n + \lambda(u_{j-1}^n + u_{j+1}^n) \\ &\quad - (1 - 2\lambda)v_j^n - \lambda(v_{j-1}^n + v_{j+1}^n) - \epsilon_j^n \\ &= (1 - 2\lambda)e_j^n + \lambda e_{j-1}^n + \lambda e_{j+1}^n - \epsilon_j^n.\end{aligned}$$

Taking absolute values on each side and using the triangle inequality,

$$|e_j^{n+1}| \leq |1 - 2\lambda| |e_j^n| + |\lambda| |e_{j-1}^n| + |\lambda| |e_{j+1}^n| + |\epsilon_j^n|. \quad (7)$$

We introduce the globally maximum error:

$$E^n = \|e^n\|_\infty = \max_j |e_j^n|.$$

Taking maximums on each side in (7), we find

$$\begin{aligned} E^{n+1} &\leq \max_j \left[|1 - 2\lambda| |e_j^n| + |\lambda| |e_{j-1}^n| + |\lambda| |e_{j+1}^n| + |\epsilon_j^n| \right] \\ &\leq |1 - 2\lambda| \max_j |e_j^n| + |\lambda| \max_j |e_{j-1}^n| + |\lambda| \max_j |e_{j+1}^n| + \max_j |\epsilon_j^n| \\ &\leq (|1 - 2\lambda| + 2|\lambda|) E^n + \max_j |\epsilon_j^n| \end{aligned}$$

We assumed that $\lambda \leq 1/2$ and so $|1 - 2\lambda| = 1 - 2\lambda$ hence

$$|1 - 2\lambda| + 2|\lambda| = 1.$$

(If $\lambda > \frac{1}{2}$, then this is false, which does not by itself prove instability; but we know from explicit Fourier solutions that it is unstable if $\lambda > \frac{1}{2}$.) Therefore, we have

$$E^{n+1} \leq E^n + \max_j |\epsilon_j^n|. \quad (8)$$

That is, the error at the previous step is *not amplified*, since E^n is not multiplied by a coefficient greater than 1. The error at later steps is simply the accumulation of errors introduced at earlier steps, and we now have enough information to show that the total accumulated error is small.

Indeed, we simply sum (8) over n (using $E^0 = 0$) to see

$$E^n \leq \sum_{\ell=0}^{n-1} \max_j |\epsilon_j^\ell| \leq \sum_{\ell=0}^{n-1} \widetilde{M} k h^2 = \widetilde{M} n k h^2 \quad (9)$$

where $\widetilde{M} = MD(\lambda + 1)$.

We have nearly finished the proof¹. We use the bound (9) to prove that given a fixed time T , the approximate solutions u_j^n converge to $u(x, t)$ for any $x \in \mathbb{R}$ and any $t \in [0, T]$. First, we use T to choose the time-step k

¹The rest of this paragraph will make sense to you only if you've taken a course in real analysis.

via $k = T/N$ where $N \in \mathbb{N}$ and N is big enough so that $k \leq K$ (to ensure bound (2)). Now that k is chosen, this determines h via $h = \sqrt{TD/(\lambda n)}$. We choose n larger still, if needed, so that $h \leq H$ (to ensure bound 1). For this h, k pair, we generate the approximate solutions u_j^n . By the above work, we know that

$$E^n \leq \widetilde{M} n k h^2 \leq \widetilde{M} N k h^2 = \widetilde{M} T h^2 = \widetilde{M} T \frac{TD}{\lambda n} \quad (10)$$

for any $n \leq N$. Now, fix a point (x, t) so that $t \leq T$. For each N there will be a grid point $(j_* h, n_* k)$ that is closest to (x, t) . As $N \rightarrow \infty$ these grid points will converge to (x, t) . As a result, given $\epsilon > 0$, there is N_0 so that for any $N \geq N_0$ we have $|v(j_* h, n_* k) - v(x, t)| < \epsilon/2$. This used the fact that $v(x, t)$ is a continuous function of x and t . Now by taking $n = n_*$ in the bound (10) we see that as $n_* \rightarrow \infty$ the right-hand side will go to zero. And so there is some $N_1 \geq N_0$ so that for $N \geq N_1$ the right-hand side will be less than $\epsilon/2$. Combining this,

$$|u_{j_*}^{n_*} - v(x, t)| \leq |u_{j_*}^{n_*} - v(j_* h, n_* k)| + |v(j_* h, n_* k) - v(x, t)| \leq \epsilon/2 + \epsilon/2 = \epsilon.$$

This finishes the proof.

Let's summarize the reasoning one more time. We did two things. First, we checked *consistency*; by plugging some hypothetical exact solution into the difference formula, we computed that the truncation error was small (6). Hence, using the discrete formula introduces a small error at each time step.

Next, we checked *stability* (8). This told us that the error did not amplify from one step to the next, except by the accumulation of the truncation errors. Stability combined with consistency gave us convergence.

As a postscript, let us remember that this result is not quite as powerful as it may sound. It depends essentially on having very smooth initial data, so that the solution remains smooth and we get calculus results like (3,4). But the heat equation has very strong smoothing properties: initial data that is not smooth quickly becomes so. Finite difference methods often work just fine on such problems, even though this theory does not apply. Much more is known about this kind of problems than we have time for.

One final note: in the above, I was very careful about my constants. I kept track of the D s and the λ s and ended up with a \widetilde{M} in which I could

see how things depended on what. This is not something I'd normally do — the mathematician in me would usually do the following: with each subsequent inequality I'd keep using the same notation M for the constant but I'd comment that each subsequent M might be larger than the prior M . After all, all I need is some upper bound in the end. If, for some reason, I found that I really needed to keep track of how things depended on λ and D then I could redo things and be super-careful. But normally if all I want to do is prove convergence, I wouldn't do this.

Some might argue that this is the difference between a theoretician: "There is a constant such that..." and someone who actually implements the theory: "Hey wait a minute! Your constant is huge and the numerical scheme's really slow, as a result!" This exchange could certainly happen if I'd just proven that the running time of the scheme is CNM^2 where N is the number of intervals in space and M is the number of intervals in time.

2.1 Our friend $\mathcal{O}(k^2)$

I want to show why

$$\frac{1}{k} \left(v_j^{n+1} - v_j^n \right) \sim v_t(jh, nk) + \frac{1}{2} k v_{tt}(jh, nk) + \mathcal{O}(k^2), \quad k \rightarrow 0. \quad (11)$$

implies that there is some $K > 0$ such that

$$\left| \frac{1}{k} \left(v_j^{n+1} - v_j^n \right) - v_t(jh, nk) \right| \leq k \|v_{tt}(jh, nk)\|_\infty$$

for all $k < K$.

The statement (11) can be rewritten as

$$\frac{1}{k} \left(v_j^{n+1} - v_j^n \right) - v_t(jh, nk) - \frac{1}{2} k v_{tt}(jh, nk) \sim \mathcal{O}(k^2), \quad k \rightarrow 0.$$

By definition of $\mathcal{O}(k^2)$, this means that there is a $C > 0$ and a $K_0 > 0$ such that

$$\left| \frac{1}{k} \left(v_j^{n+1} - v_j^n \right) - v_t(jh, nk) - \frac{1}{2} k v_{tt}(jh, nk) \right| \leq C k^2$$

for all $k < K_0$. By the triangle inequality, if A and B are numbers then $|A| = |A - B + B| \leq |A - B| + |B|$ hence $|A| - |B| \leq |A - B|$. Therefore we have

$$\left| \frac{1}{k} \left(v_j^{n+1} - v_j^n \right) - v_t(jh, nk) \right| - \left| \frac{1}{2} k v_{tt}(jh, nk) \right| \leq C k^2$$

and thus

$$\left| \frac{1}{k} \left(v_j^{n+1} - v_j^n \right) - v_t(jh, nk) \right| \leq \frac{1}{2}k |v_{tt}(jh, nk)| + Ck^2 \leq \frac{1}{2}k \|v_{tt}\|_\infty + Ck^2$$

Now, let $K_1 = \|v_{tt}\|_\infty / (2C)$. Then

$$k < K_1 \quad \implies \quad Ck^2 < \frac{1}{2}k \|v_{tt}\|_\infty$$

It then follows that if $K := \min\{K_0, K_1\}$ then

$$\left| \frac{1}{k} \left(v_j^{n+1} - v_j^n \right) - v_t(jh, nk) \right| \leq k \|v_{tt}\|_\infty$$

for all $k < K$.

3 How to use truncation errors to test your code

Let's step back and look at what we've done in proving convergence for the explicit euler scheme for the heat equation $u_t = Du_{xx}$. Our finite difference scheme had two parts: the spatial discretization and the temporal discretization. And so if $v(x, t)$ is a smooth solution of the heat equation then

$$\frac{v_{j+1}^n - 2v_j^n + v_{j-1}^n}{h^2} = v_{xx}(x_j, t_n) + \frac{1}{12}h^2 v_{xxxx}(\tilde{x}, t_n), \text{ for some } \tilde{x} \in (x_{j-1}, x_{j+1})$$

and

$$\frac{v_j^{n+1} - v_j^n}{k} = v_t(x_j, t_n) + \frac{1}{2}kv_{tt}(x_j, \tilde{t}), \text{ for some } \tilde{t} \in (t_n, t_{n+1})$$

If we further assumed that $v_{xxxx}(x, 0)$ is uniformly bounded on \mathbb{R} then there is some $M < \infty$ such that $|v_{xxxx}(x, t)| \leq M$ and $|v_{tt}(x, t)| \leq M$ for all $x \in \mathbb{R}$ and all $t > 0$. As a result,

$$\left| \frac{v_{j+1}^n - 2v_j^n + v_{j-1}^n}{h^2} - v_{xx}(x_j, t_n) \right| = \mathcal{O}(h^2)$$

and

$$\left| \frac{v_j^{n+1} - v_j^n}{k} - v_t(x_j, t_n) \right| = \mathcal{O}(k).$$

Recall that “ $A = \mathcal{O}(k)$ ” means there exists a K and a constant C such that $|A| \leq Ck$ for all $k < K$. Note that because the bounds on v_{xxxx} and v_{tt} are independent of x and t , the constants C in the $\mathcal{O}(h^2)$ and $\mathcal{O}(k)$ bounds are independent of j and n .

From the above,

$$\left| \frac{v_j^{n+1} - v_j^n}{k} - D \frac{v_{j+1}^n - 2v_j^n + v_{j-1}^n}{h^2} - (v_t(x_j, t_n) - Dv_{xx}(x_j, t_n)) \right| = \mathcal{O}(k + h^2)$$

If we make the further assumption that $k = \lambda h^2/D$ then the right-hand side is $\mathcal{O}(h^2)$.

Using these local truncation errors and the stability of the scheme, we were able to control the true error $e_j^n = u_j^n - v(x_j, t_n)$:

$$\|e^n\|_\infty = \max_j |e_j^n| \leq h^2 t_n MD(\lambda + 1)$$

where $t = nk$ and $k = \lambda h^2/D$ and h is small enough so that $h < H$ and $k < K$ where H and K are such that the bounds (1) and (2) from the January 24 notes hold.

What does this mean? It means that at any point (x_j, t_n) in space time, if k (and hence h) is small enough then

$$|u_j^n - v(x_j, t_n)| \leq h^2 t_n MD(\lambda + 1), \implies u_j^n = v(x_j, t_n) + \mathcal{O}(h^2).$$

I can use this to test if I've written my code correctly. To do this, I consider the heat equation $u_t = u_{xx}$ on $[0, 1]$ with Neumann boundary conditions and initial data $u_0(x) = \cos(3\pi x)$. The exact solution is

$$v(x, t) = e^{-(3\pi)^2 t} \cos(3\pi x)$$

I choose the initial time is 0 and the final time is $1/10$. I compute five discrete solutions: u_1 with $h_1 = 1/8$ and $k_1 = 1/160$, u_2 with $h_2 = h_1/2 = 1/16$ and $k_2 = k_1/4 = 1/640$, u_3 with $h_3 = h_2/2$ and $k_3 = k_2/4$, u_4 with $h_4 = h_3/2$ and $k_4 = k_3/4$, and u_5 with $h_5 = h_4/2$ and $k_5 = k_4/4$. That is, I'm dividing the space-step by two and the time-step by 4 with each refinement. They are chosen so that $\lambda = 2/5 < 1/2$.

We've proven that the scheme converges and because we know the exact solution, we can actually look at the true errors. Below, I present the errors at the final time $t = 1/10$:

	error	ratio
$\ u1^{16} - v(\cdot, 1/10)\ _\infty$	1.2022e-04	2.7052
$\ u2^{64} - v(\cdot, 1/10)\ _\infty$	4.4441e-05	3.6644
$\ u3^{256} - v(\cdot, 1/10)\ _\infty$	1.2128e-05	3.9160
$\ u4^{1024} - v(\cdot, 1/10)\ _\infty$	3.0970e-06	3.9790
$\ u5^{4098} - v(\cdot, 1/10)\ _\infty$	7.7834e-07	

(recall that it took 16 time steps for $u1$ to reach $t = 1/10$). The first thing you notice is that the errors are decreasing. This is good. But you can get more out of the errors. The second column is the ratio of subsequent errors. You note that as k (and hence h) gets smaller, the ratios appear to be getting closer to 4. This is what you're looking for.

What's going on? We know that

$$\|u1^{16} - v(\cdot, 1/10)\|_\infty = \max_j |u1_j^{16} - v(x_j, 1/10)| \sim (h1)^2$$

Similarly,

$$\|u2^{64} - v(\cdot, 1/10)\|_\infty \sim (h2)^2 = (h1)^2/4.$$

Combining the two,

$$\frac{\|u1^{16} - v(\cdot, 1/10)\|_\infty}{\|u2^{64} - v(\cdot, 1/10)\|_\infty} \sim \frac{(h1)^2}{(h2)^2} = 4.$$

And so as the space-step and time-step get smaller and smaller (but always by factors of 2 and 4, respectively) the ratios of the errors get closer and closer to 4. Note: there's nothing magical about factors of 2 and 4. If I'd taken the space-steps smaller and smaller by factors of 3 and the time-steps smaller by factors of 9 then the ratios get closer and closer to 9.

This looks all well and good, but what if I don't know the exact solution v ? In fact, you can still find what you're looking for:

	norm	ratio
$\ u1^{16} - u2^{64}\ _\infty$	7.5782e-05	2.3452
$\ u2^{64} - u3^{256}\ _\infty$	3.2313e-05	3.5782
$\ u3^{256} - u4^{1024}\ _\infty$	9.0307e-06	3.8948
$\ u4^{1024} - u5^{4098}\ _\infty$	2.3187e-06	

Again, the ratios are getting closer and closer to 4. Note: I should be clear about what I mean when I talk about the difference of $u1$ and $u2$. Recall

that u_1 is defined at 9 grid points ($x_j = j/8$) and u_2 is defined at 17 grid points ($x_j = j/16$). In the above, I take the differences on the coarser mesh:

$$\|u_1^{16} - u_2^{64}\|_\infty := \max_{0 \leq j \leq 8} |u_1^{16} - u_2^{64}|$$

Why are those ratios going to 4? Loosely speaking,

$$\begin{aligned} \|u_1^{16} - u_2^{64}\|_\infty &\sim \|u_1^{16} - v(x_j, 1/10)\|_\infty + \|u_2^{64} - v(x_j, 1/10)\|_\infty \\ &\sim (h_1)^2 + (h_1/2)^2 = (h_1)^2(1 + \frac{1}{4}) \end{aligned}$$

and

$$\begin{aligned} \|u_2^{64} - u_3^{256}\|_\infty &\sim \|u_2^{64} - v(x_j, 1/10)\|_\infty + \|u_3^{256} - v(x_j, 1/10)\|_\infty \\ &\sim (h_1/2)^2 + (h_1/4)^2 = (h_1/2)^2(1 + \frac{1}{4}) \end{aligned}$$

and so

$$\frac{\|u_1^{16} - u_2^{64}\|_\infty}{\|u_2^{64} - u_3^{256}\|_\infty} \sim \frac{(h_1)^2(1 + \frac{1}{4})}{(h_1/2)^2(1 + \frac{1}{4})} = 4$$

One can make the above more rigorous.

I will now try to convince you of the power of all those ratios. Let's imagine that there's a bug in my code, somewhere in the spatial discretization, for example. For example, let's imagine I have an even number of subintervals in space and that at the middle grid point $x_{N/2}$ I mistyped and as a result I've coded up

$$u_{xx}(x_{N/2}) \sim \frac{u_{N/2+1} - 2u_{N/2} + 0u_{N/2-1}}{h^2}$$

(Note that there's a coefficient of zero there, instead of 1.) As a result, one of the $N + 1$ equations determining u^{n+1} is incorrect. How big of an effect could that have? It's only at one point, after all! The result of this error is that at the $N/2$ grid point we'll have

$$\left| \frac{v_{N/2+1}^n - 2v_{N/2}^n + 0v_{N/2-1}^n}{h^2} - v_{xx}(x_j, t_n) \right| = \mathcal{O}(h)$$

That is, the consistency at that grid point is $\mathcal{O}(h)$ rather than the $\mathcal{O}(h^2)$ we have at all the other grid points. But recall that the convergence arguments were in the supremum norm and so not only will what's going on at this

one grid point matter but it'll be what's most important because $h^2 < h$ for h small. If we follow through the argument as before we'll find that this buggy code does converge to a solution but that the error estimate is now

$$\|e^n\|_\infty = \max_j |e_j^n| \leq htMD(\lambda + 1)$$

Note that right-hand side is now $\mathcal{O}(h)$ rather than $\mathcal{O}(h^2)$!

Is this all in theory or does it show up in practice? To check this, I coded up this buggy code and repeated the ratio testing from before:

	error	ratio
$\ u1^{16} - v(\cdot, 1/10)\ _\infty$	5.1947e-02	1.6507
$\ u2^{64} - v(\cdot, 1/10)\ _\infty$	3.1470e-02	1.8815
$\ u3^{256} - v(\cdot, 1/10)\ _\infty$	1.6726e-02	1.9584
$\ u4^{1024} - v(\cdot, 1/10)\ _\infty$	8.5404e-03	1.9843
$\ u5^{4098} - v(\cdot, 1/10)\ _\infty$	4.3041e-03	

Note that not only are the errors larger than before but the ratios are going to 2. Because the ratios are going to 2 rather than 4, we know with 100% confidence that our code is not doing what we wanted it to do. Time to go bug hunting!

Remember back when we looked at how to code up the Neumann and Robin boundary conditions? We used

$$v_x(0) \implies \frac{v_1 - v_{-1}}{2h} = 0 \implies v_{-1} = v_1$$

This finite-difference approximation of v_x is $\mathcal{O}(h^2)$ close to v_x . And it works great in the code – it's what I used at each endpoint in the code that generated the first set of tables. What if we hadn't done this? What if we'd done

$$v_x(0) \implies \frac{v_0 - v_{-1}}{h} = 0 \implies v_{-1} = v_0$$

instead? This finite-difference approximation of v_x is $\mathcal{O}(h)$ close to v_x . And so we can't hope for any thing better than in our buggy code above. Here's what we get when implementing the Neumann boundary condition sub-optimally at $x = 0$:

	error	ratio
$\ u1^{16} - v(\cdot, 1/10)\ _\infty$	1.1542e-01	1.9517
$\ u2^{64} - v(\cdot, 1/10)\ _\infty$	5.9142e-02	1.9864
$\ u3^{256} - v(\cdot, 1/10)\ _\infty$	2.9773e-02	1.9963
$\ u4^{1024} - v(\cdot, 1/10)\ _\infty$	1.4914e-02	1.9989
$\ u5^{4098} - v(\cdot, 1/10)\ _\infty$	7.4613e-03	

Again, note that not only are the errors larger than before but the ratios are going to 2. Because the ratios are going to 2 rather than 4.

Okay, so do we really care about $\mathcal{O}(h^2)$ versus $\mathcal{O}(h)$? If we want to achieve an error of 10^{-9} we can do this with either code. So what's the big deal? This is where the runtime² shows up:

	error	runtime (sec)
$\ u2^{64} - v(\cdot, 1/10)\ _\infty$	5.9142e-02	4.0368e-03
$\ u3^{256} - v(\cdot, 1/10)\ _\infty$	2.9773e-02	6.2507e-02
$\ u4^{1024} - v(\cdot, 1/10)\ _\infty$	1.4914e-02	1.1203e+00
$\ u5^{4098} - v(\cdot, 1/10)\ _\infty$	7.4613e-03	5.2689e+01
$\ u6^{16384} - v(\cdot, 1/10)\ _\infty$	3.7313e-03	1.6711e+03

Note that

$$\text{err}(h) \sim ch^\alpha \implies \log_{10}(\text{err}(h)) \sim \log_{10}(c) + \alpha \log_{10}(h).$$

Doing a least-squares fit on the above data, I find $\alpha = 1.0$ (as expected) and $c = 0.94$. This implies that to get an error of 10^{-8} I would have to take $h = 1.06e-08$. Turning to the runtime,

$$\text{run}(h) \sim dh^\beta \implies \log_{10}(\text{run}(h)) \sim \log_{10}(d) + \beta \log_{10}(h).$$

Doing a least-squares fit on the above data, I find $\beta = 4.7$ and $d = 5.5e-09$. And so to reach an error of $1e-08$ it would take $2.4e+29$ seconds ($7.6e+21$ years). Which is a long time.

If I do the same calculations with the code where the boundary conditions were both implemented in an $\mathcal{O}(h^2)$ manner then I would have gotten

²The runtime is done on my mac and I was using more RAM than CPU. Ideally, I'd want to get flop counts since this would be platform independent. But one can't get flop counts in matlab.

	error	runtime (sec)
$\ u2^{64} - v(\cdot, 1/10)\ _\infty$	4.4441e-05	3.6236e-03
$\ u3^{256} - v(\cdot, 1/10)\ _\infty$	1.2128e-05	5.9179e-02
$\ u4^{1024} - v(\cdot, 1/10)\ _\infty$	3.0970e-06	1.0569e+00
$\ u5^{4098} - v(\cdot, 1/10)\ _\infty$	7.7834e-07	5.0538e+01
$\ u6^{16384} - v(\cdot, 1/10)\ _\infty$	1.9484e-07	1.6319e+03

There's no marked change in the runtimes, obviously. The change in boundary conditions doesn't affect the code — it just affects the accuracy. Doing a least-squares fit as above, I find $\alpha = 2.0$ (as expected) and $c = 1.1e-02$. This implies that to get an error of $1e-08$ I would have to take $h = 9.5e-05$. Turning to the runtime, I find $\beta = 4.7$ and $d = 5.1e-09$. And so to reach an error of $10e-08$ It would take $8.3e+05$ seconds. Which is about ten days.

What could we do to make things go faster? As you'll see in your homework, if we'd done things via Crank-Nicolson then the truncation error would be $\mathcal{O}(k^2 + h^2)$. And rather than taking k smaller by factors of 4 each time, we'd take them smaller by factors of 2, just like h . This will make things run much faster.

A word of experience: if your time-stepping is $\mathcal{O}(k)$ then a bug in the implementation of the time-stepping will likely leave it $\mathcal{O}(k)$. Similarly, using accuracy to go bug hunting won't have an effect if the spatial discretization is $\mathcal{O}(h)$. So not only is getting higher accuracy a good idea for accuracy's sake but it provides a useful tool for bug-hunting.

Another word of experience: sometimes you'll have a code for which convergence has not been proven. But if you do runs and show errors that are decreasing and ratios that are going to a constant then it's consistent with a numerical scheme that's convergent. The next thing you need to convince yourself of is that what it's converging to is actually a solution of the problem you're interested in.

More words of experience: In the above, I set $D = 1$. It's a bad idea, in general, to set your parameters to 1. This is because if there's a bug, like you should've coded in D^2 but you coded in D^3 then this type of error will be missed if $D = 1$. Also, when you're testing a code for convergence, you don't need to run the code for a long time. I chose to run to the time

$t = 1/10$ with the time-steps I chose because the fifth solution computed in under 10 minutes. I got the same quality of convergence information that I would have gotten had I computed to time $t = 1$.