5 Discretise 1D space

The 2014 plan: Taylor's theorem, consistency, varying coefficients, finding equilibria, eigenvalues, symmetric discretisations reflect conservation,

Contents

We aim to be able to computationally solve most PDEs knowing something about the typical structures found in separation of variables and Sturm–Liouville theory. We start with the heat PDE, $u_t = u_{xx}$, and then more complicated PDEs.

5.1 Taylor's theorem underpins the method of lines

Theorem 5.1 (Taylor's theorem, (Kreyszig 2011, p.691)). *For functions* $f(x)$ *differentiable* $(P+1)$ *times in some domain,*

$$f(x+h) = f(x)+hf'(x)+\frac{1}{2}h^2f''(x)+\cdots+\frac{1}{P!}h^Pf^{(P)}(x)+R_P(x,h),$$

where the ***remainder*** $R_P(x,h) := \frac{1}{(P+1)!}h^{P+1}f^{(P+1)}(\xi)$ *for some* $\xi(x,h)$ *such that* $x \leqslant \xi \leqslant h$.

Example 5.1 $f''(x) \approx [f(x-h) - 2f(x) + f(x+h)]/h^2$ as seen in other courses.

Solution: Taylor says, for $P = 3$,

$$f(x \pm h) = f(x) \pm hf'(x) + \frac{1}{2}h^2f''(x) \pm \frac{1}{6}h^3f'''(\xi_\pm).$$

Then

$$\begin{aligned} & f(x-h) - 2f(x) + f(x+h) \\ = \; & +f(x) - hf'(x) + \tfrac{1}{2}h^2 f''(x) - \tfrac{1}{6}h^3 f'''(\xi_-) \\ & -2f(x) \\ & +f(x) + hf'(x) + \tfrac{1}{2}h^2 f''(x) + \tfrac{1}{6}h^3 f'''(\xi_+) \\ = \; & h^2 f''(x) + \tfrac{1}{6}h^3[f'''(\xi_+) - f'''(\xi_-)] \end{aligned}$$

Divide by h^2 to get

$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} - \tfrac{1}{6}h[f'''(\xi_+) - f'''(\xi_-)]$$

and so the approximation is OK when parameter h is small (more specifically when hf''' is small). ♣

Example 5.2 Approximate solutions $u(x,t)$ to the heat PDE, $u_t = u_{xx}$, on $0 < x < 1$ with BCs $u(0,t) = a$ and $u(1,t) = b$.

Solution: For simplicity consider $x = 1/2$ and $h = 1/2$. Then previous example applies to give $u_{xx}(\frac{1}{2},t) \approx \frac{u(0,t)-2u(\frac{1}{2},t)+u(1,t)}{(1/2)^2} = 4(a - 2u(\frac{1}{2},t) + b)$. Defining $u_2(t) := u(\frac{1}{2},t)$ the heat PDE says $du_2/dt = u_t(\frac{1}{2},t) \approx 4(a - 2u(\frac{1}{2},t) + b) = 4(a+b) - 8u_2$. That is, $\dot{u}_2 + 8u_2 \approx 4(a+b)$. From first year, this ODE has general solution $u_2(t) = \frac{a+b}{2} + ce^{-8t}$. That is, it predicts the midpoint value quickly approaches the mean of the two ends. But is $h = 1/2$ small enough for the approximation to be acceptable? and what about other spatial positions? ♣

Example 5.3 Use finite differences to compute approximate solutions $u(x,t)$ to the heat PDE, $u_t = u_{xx}$, on $0 < x < 1$ with BCs $u(0,t) = 0$ and $u(1,t) = \sin 8t$.

Solution: Draw diagram: choose N ($N = 3$ is the previous example), set $h := 1/(N-1)$, $x_j := (j-1)/(N-1)$ and $u_j(t) := u(x_j,t)$ for $j = 1, 2, \ldots, N$. Taylor example applies to give $u_{xx}(x_j,t) \approx \frac{u(x_j-h,t)-2u(x_j,t)+u(x_j+h,t)}{h^2} =$

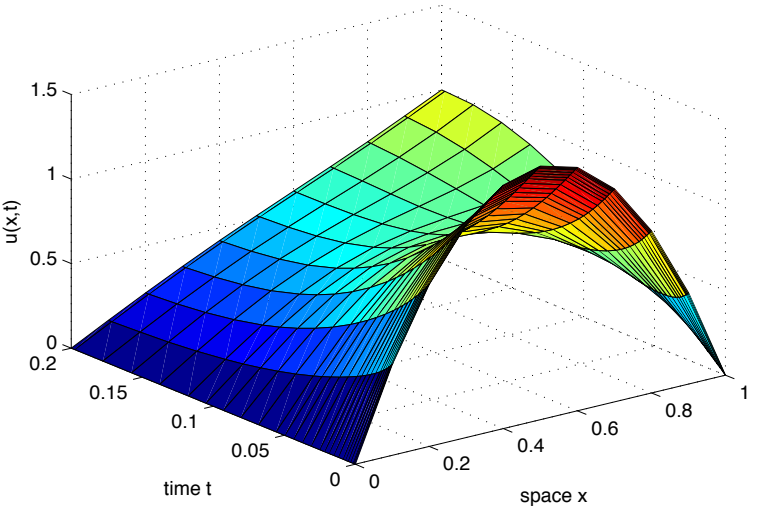

Figure 5.1: Algorithm 5.1 computes this solution of the heat PDE $u_t = u_{xx}$.

$\frac{u(x_{j-1},t)-2u(x_j,t)+u(x_{j+1},t)}{h^2} = \frac{u_{j-1}-2u_j+u_{j+1}}{h^2}$. Then the heat PDE is approximately the set of ODEs

$$\begin{aligned} \frac{du_j}{dt} &= u_t(x_j,t) = u_{xx}(x_j,t) \\ &\approx \frac{u_{j-1}-2u_j+u_{j+1}}{h^2}, \end{aligned}$$

for only $j = 2,\dots,N-1$, as otherwise involves values outside the domain. The extreme ODEs are

$$\begin{aligned} \frac{du_2}{dt} &\approx \frac{u_1-2u_2+u_3}{h^2} = \frac{a-2u_2+u_3}{h^2}, \\ \frac{du_{N-1}}{dt} &\approx \frac{u_{N-2}-2u_{N-1}+u_N}{h^2} = \frac{u_{N-2}-2u_{N-1}+b}{h^2}. \end{aligned}$$

This is a set of $N-2$ ODEs, all coupled together. Here we could solve analytically, but the point of this approach is to solve computationally to find, for example, Figure 5.3. ♣

For the moment we assume integrating in time is a solved task (*Modelling with ODEs*). So Algorithm 5.1 invokes `ode15s()` in Matlab (or in Octave invoke `lsode()` but interchange `u` and `t`, or in Scilab invoke `ode()`). Let's explain Algorithm 5.1.

1. Main function header so we can define other functions in the same file: repeat this function name in the filename `heat.m`.
2. Always include a description: `%` preceeds comment.
3. Always include author and date: usefully, these initial comment lines will be displayed upon `help heat`.
4. Define `global` those variables to be shared between the main code and the time derivative function `dudt()`.
5. Set the number of spatial points: prefer descriptive names.
6. Use `linspace()` to set the grid of points in column vector `x`.
7. The points in the interior correspond to index $j = 2, 3, \ldots, N-1$.
8. Set initial shape of the field $u0 = u_j(0) = 6x_j(1-x_j)$
10. `ode15s()` integrates the ODEs in `dudt()`, here over a time $0 \leqslant t \leqslant 0.2$, starting at initial condition `u0`—of which we only want the interior values.
12. Wrap the computed interior solution `u` with the boundary values for each of the computed times in `t`—it is one of the glitches in Matlab that `u` is a column vector in the time derivative function, but forms rows in the result of `ode15s`.
13. Draw a surface plot of the computed solution: plots a surface patch between each time and space point.
14. Label the axes of the plot.
16. Define the time derivative function `dudt`.
17. Empower access by this function to the `global` variables.
18. Compute $h = dx$ from the information already encoded: do not code potentially contradictory values.
19. Wrap the interior (column) vector `u` with the two boundary values.

20. Compute time derivative of the interior values from the ODEs.

22. Function `uleft()` computes value at the left end of domain.

23. At the left the value is to be zero (for each time).

25. Function `uright()` computes value at the right end of domain.

26. At the right the value is to be $\sin 8t$ (for each time).

Algorithm 5.1: method of lines to solve the heat PDE. This framework adapts to many similar PDE problems. Note: every document must include title, author and date.

```
1  function heat
2  % Compute solutions of heat PDE with finite differences
3  % AJR, 13/8/2014
4  global j x
5  nPoints=11
6  x=linspace(0,1,nPoints)'
7  j=2:nPoints-1
8  u0=6*x.*(1-x)
9
10 [t,u]=ode15s(@dudt,[0 0.2],u0(j))
11
12 u=[uleft(t) u uright(t)]
13 surf(x,t,u)
14 xlabel('space x'),ylabel('time t'),zlabel('u(x,t)')
15 %-----------------------------------------------------
16 function ut=dudt(t,u)
17 global j x
18 dx=x(2)-x(1);
19 u=[uleft(t);u;uright(t)];
20 ut=(u(j-1)-2*u(j)+u(j+1))/dx^2;
21 %-----------------------------------------------------
22 function ua=uleft(t)
23 ua=zeros(size(t));
24 %-----------------------------------------------------
25 function ub=uright(t)
26 ub=sin(8*t);
```

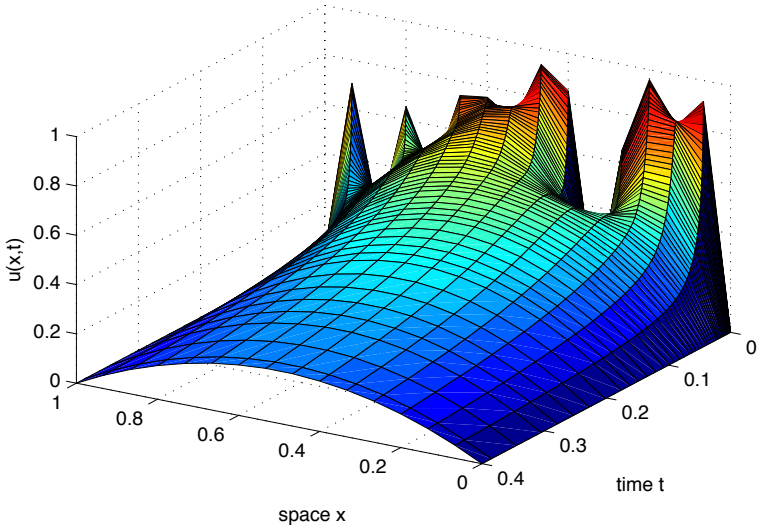

Figure 5.2: Algorithm 5.2 computes this solution of the nonlinear PDE $u_t = uu_{xx}$ *showing the emergence of a slowly decaying parabola.*

Algorithm 5.1 exhibits an important principle: code once, use several times.

- `dx` is not `1/(nPoints-1)` because the `1` is the length of the domain which is already coded in the `x=linspace()`, so we must use these `x` to determine `dx`.
- The left and right boundary values are needed in two places, so code each once, and use twice.

Example 5.4 Numerical methods are most valuable for nonlinear problems such as solving $u_t = uu_{xx}$ for $0 < x < 1$ and boundary values $u(0,t) = u(1,t) = 0$. Recall from 'homework' that separation of variables finds the special solution $u(x,t) = x(1-x)/(c+2t)$: Algorithm 5.2 suggests that this is an emergent solution from all initial conditions. Modify Algorithm 5.1 to get Algorithm 5.2:

- change the initial condition to random as one way to explore the space of initial conditions: `rand()`

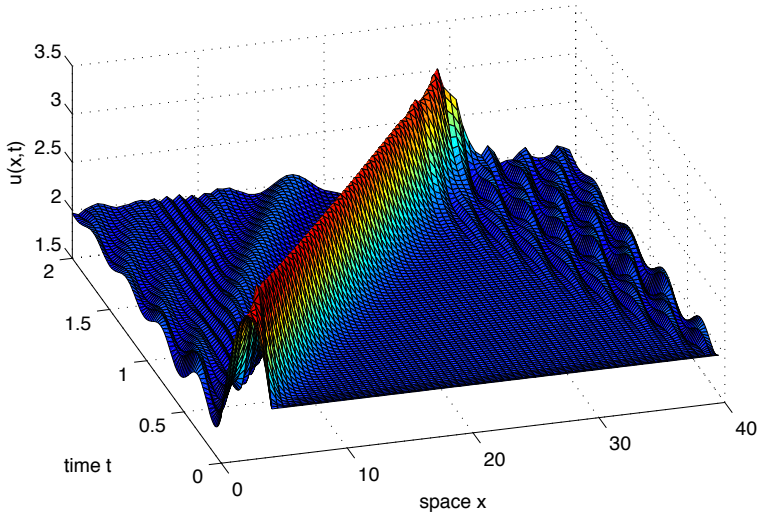

Figure 5.3: Algorithm 5.3 computes this solution of the nonlinear Korteweg–de Vries PDE $u_t+6uu_x+u_{xxx}=0$. *Observe emergence of a travelling solitary wave.*

generates uniformly distributed numbers on $[0,1)$.

- integrate a little longer to see the emergent algebraic decay of the known special solution;
- change the finite difference code to that for the nonlinear PDE;
- change boundary value code to zeros.

Optional: play with the code to solve variations on the theme: add forcing such as $1-x$, or multiply the diffusion coefficient by x, and so on.

Ongoing research in the School explores such emergence: how does such relatively simple structures emerge and evolve from complexity? ♣

Example 5.5 (Korteweg–de Vries PDE). The nonlinear Korteweg–de Vries PDE $u_t+6uu_x+u_{xxx}=0$ partially models waves in shallow water and nonlinear optics.

Algorithm 5.2: method of lines to solve the nonlinear PDE $u_t = uu_{xx}$. Observe emergence of simple parabola.

```
1  function heatnon
2  % Compute solutions of nonlinear PDE via finite differences
3  % AJR, 13/8/2014
4  global j x
5  nPoints=16
6  x=linspace(0,1,nPoints)'
7  j=2:nPoints-1
8  u0=rand(size(x))
9
10 [t,u]=ode15s(@dudt,[0 0.4],u0(j))
11
12 u=[uleft(t) u uright(t)]
13 surf(x,t,u)
14 xlabel('space x'),ylabel('time t'),zlabel('u(x,t)')
15 %--------------------------------------------------
16 function ut=dudt(t,u)
17 global j x
18 dx=x(2)-x(1);
19 u=[uleft(t);u;uright(t)];
20 ut=u(j).*(u(j-1)-2*u(j)+u(j+1))/dx^2;
21 %--------------------------------------------------
22 function ua=uleft(t)
23 ua=zeros(size(t));
24 %--------------------------------------------------
25 function ub=uright(t)
26 ub=zeros(size(t));
```

Let's solve the PDE on $0 < x < 40$ with boundary conditions that the solution is 40-periodic. Start with an initial 'lump' of water for $0 < x < 4$ on depth otherwise $u(x, 0) = 2$. Algorithm 5.3 shows an emergent travelling solitary wave. Modify Algorithm 5.1 to get Algorithm 5.3:

7. as domain is periodic we discard the end point as it is identical to the start point;
8. the initial condition has a parabolic lump situated for $x < 4$ which is coded inline since `(x<4)` evaluates to one for $x < 4$ and otherwise is zero;
12. there is no need to wrap boundary values around the solution as the solution is periodic;
18. implement periodicity by wrapping around `u` values from the other end;
19. then the interior points correspond to a shifted `j`;
20–21. code the discrete version of the PDE, using `...` to tell Matlab that the statement continues onto the next line;

- discard the boundary value functions as they are replaced by periodicity.

Optional: show a longer time simulation and that the solitary wave wraps around the domain; use a larger initial lump and get two or more solitary waves emerging; use a smaller grid spacing and show fine scale 'noise' begins to dominate the numerical solution (keep the fine scale acceptable with $\Delta x \geqslant 0.5$).

But how do we justify the discretisation of the Korteweg–de Vries PDE coded in Algorithm 5.3? ♣

Algorithm 5.3: method of lines to solve the nonlinear Korteweg–de Vries PDE $u_t + 6uu_x + u_{xxx} = 0$.

```
1  function kdv
2  % Compute solutions of Korteweg-de Vries PDE
3  % with perdiodic boundary conditions.
4  % AJR, 14/8/2014
5  global x
6  nPoints=80
7  x=linspace(0,40,nPoints+1); x=x(1:nPoints)';
8  u0=2+x.*(1-x/4).*(x<4);
9
10 [t,u]=ode15s(@dudt,[0 2],u0);
11
12 surf(x,t,u)
13 xlabel('space x'),ylabel('time t'),zlabel('u(x,t)')
14 %-------------------------------------------------------
15 function ut=dudt(t,u)
16 global x
17 dx=x(2)-x(1);
18 u=[u(end-1:end);u;u(1:2)];
19 j=(1:length(x))+2;
20 ut=-3*(-u(j-1).^2+u(j+1).^2)/(2*dx) ...
21    -(-u(j-2)+2*u(j-1)-2*u(j+1)+u(j+2))/(2*dx^3) ;
```

5.1.1 Ensure a discretisation is consistent with the PDE

Computer algebra checks via some of the following steps in Matlab.

1. Find the Taylor series of any given function such as the arcsinh, namely $\sinh^{-1} x = x - \frac{1}{6}x^3 + \frac{3}{40}x^5 + \mathcal{O}(x^6)$,

```
syms x
taylor(asinh(x),x)
```

2. But we want to consider an arbitrary function $u(x)$ and its nearby values $u(x \pm h), u(x \pm 2h), \ldots$: for example, compute the Taylor series $u(x+h) = u(x) + hu'(x) + \frac{1}{2}h^2u''(x) + \cdots + \mathcal{O}(h^6)$ (albeit printed in reverse order) as `D(u)` denotes a derivative:

```
syms u(x) h
taylor(u(x+h),h)
```

3. Now check that $[u(x-h)-2u(x)+u(x-h)]/h^2$ estimates the second derivative to error approximately $h^2u^{iv}/12$ by

```
taylor((u(x-h)-2*u(x)+u(x+h))/h^2,h)
```

4. Similarly, as coded in Algorithm 5.3, we find that $3(u(x+h)^2 - u(x-h)^2)/(2h) = 6uu' + h^2(uu''' + 3u'u'') + \mathcal{O}(h^4)$ after asking for simplified results:

```
taylor(3*(-u(x-h)^2+u(x+h)^2)/(2*h),h)
simplify(ans)
```

5. Lastly, as coded in Algorithm 5.3, we find that $(u(x+2h) - 2u(x+h) + 2u(x-h) - u(x-2h))/(2h^3) = u''' + \frac{1}{4}h^2u''''' + \mathcal{O}(h^4)$

```
taylor((u(x+2*h)-2*u(x+h)+2*u(x-h)-u(x-2*h))/(2*h^3),h
```

Putting the last two results together, the time derivative function in Algorithm 5.3 codes the PDE

$$u_t = -6uu_x - h^2(uu_{xxx} + 3u_xu_{xx}) - u_{xxx} - \tfrac{1}{4}h^2u_{xxxxx} + \mathcal{O}(h^4),$$

that is,

$$u_t+6uu_x+u_{xxx} = -h^2(uu_{xxx}+3u_xu_{xx}+\tfrac{1}{4}u_{xxxxx})+\mathcal{O}(h^4),$$

This is called the **equivalent pde** of the discretisation. A discretisation is called **consistent** with a given PDE if the equivalent PDE $\to$ given PDE as the grid spacing $h \to 0$ for all x in the domain (as it does here).

Rule of thumb: prefer $\mathcal{O}(h^2)$ consistency (or better), avoid $\mathcal{O}(h)$ consistency as being too inaccurate.

Theorem 5.2 (Lax's equivalence theorem: consistency + stability = convergence)**.** *If a discretisation is consistent, and the solutions of the discretisation are stable as* $h \to 0$*, then the solutions of the discretisation approach the true solutions as* $h \to 0$*.*

Example 5.6 Consider the previous three examples. The discretisations for all three problems so far are consistent to the respective PDEs. The solutions to the linear and nonlinear heat PDEs appear stable, but those of the KdV PDE are not, as $h \to 0$. Thus we are assured the computed solutions are good for the heat PDEs, but the computations for the KdV are not necessarily so. Warning: wave PDEs are usually delicate! ♣

(Research in the School seeks to systematically generate highly consistent and stable discretisations to ensure routine quick convergence.)

5.1.2 Operator algebra generates discretisations

Challenge: how do we find a consistent discretisation for a given PDE? Answer: some operator algebra helps.

Definition 5.3. *Define the following operators:*

- *differentiation* ∂ *so that* $\partial u(x) = u'(x)$*;*
- *shift* $\mathcal{E}$ *by one grid spacing* h *so that* $\mathcal{E}u(x) = u(x+h)$*;*

- *centred difference* $\delta := \mathcal{E}^{1/2} - \mathcal{E}^{-1/2}$; *and*
- *centred mean* $\mu := (\mathcal{E}^{1/2} + \mathcal{E}^{-1/2})/2$.

For example, $\partial^2 u = u''(x)$, $\partial(u^2) = 2uu'$, $\mathcal{E}^2 u(x) = \mathcal{E}u(x+h) = u(x+2h)$, by induction $\mathcal{E}^p u(x) = u(x+ph)$ so that naturally $\mathcal{E}^{-1}u(x) = u(x-h)$ and $\mathcal{E}^{\pm 1/2}u(x) = u(x \pm h/2)$.

Theorem 5.4. *These operators are related:*

	$\mathcal{E}$	δ	$h\partial$
$\mathcal{E} =$	$\mathcal{E}$	$1 + \frac{1}{2}\delta^2 + \delta\sqrt{1+\frac{1}{4}\delta^2}$	$e^{h\partial}$
$\delta =$	$\mathcal{E}^{\frac{1}{2}} - \mathcal{E}^{-\frac{1}{2}}$	δ	$2\sinh\frac{1}{2}h\partial$
$h\partial =$	$\log\mathcal{E}$	$2\sinh^{-1}\frac{1}{2}\delta$	$h\partial$
$\mu =$	$\frac{1}{2}(\mathcal{E}^{\frac{1}{2}} + \mathcal{E}^{-\frac{1}{2}})$	$\sqrt{1+\frac{1}{4}\delta^2}$	$\cosh\frac{1}{2}h\partial$

Proof. • Start at the top right: for all $u(x)$ Taylor's theorem gives that

$$\begin{aligned} \mathcal{E}u(x) &= u(x+h) \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} h^n u^{(n)}(x) \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} h^n \partial^n u(x) \\ &= e^{h\partial} u(x). \end{aligned}$$

Hence $\mathcal{E} = e^{h\partial}$. Taking logarithm of both sides implies $\log\mathcal{E} = h\partial$ in the third row.

- Then in the second row, starting from definition

$$\begin{aligned} \delta &:= \mathcal{E}^{1/2} - \mathcal{E}^{-1/2} \\ &= e^{h\partial/2} - e^{-h\partial/2} \\ &= 2\frac{e^{h\partial/2} - e^{-h\partial/2}}{2} \\ &= 2\sinh(h\partial/2). \end{aligned}$$

Rearranging this then gives the third row entry $h\partial = 2\sinh^{-1}(\delta/2)$.

- Then consider

$$\begin{aligned} 1+\tfrac{1}{4}\delta^2 &= 1+\tfrac{1}{4}(\mathcal{E}^{1/2}-\mathcal{E}^{-1/2})^2 \\ &= 1+\tfrac{1}{4}(\mathcal{E}-2+\mathcal{E}) \\ &= \tfrac{1}{4}\mathcal{E}+\tfrac{1}{2}+\tfrac{1}{4}\mathcal{E} \\ &= (\tfrac{1}{2}\mathcal{E}^{1/2}+\tfrac{1}{2}\mathcal{E}^{-1/2})^2 \\ &= \mu^2, \end{aligned}$$

which gives the middle entry of the bottom row.

□

Example 5.7 Discretise the second derivative using

```
1 syms delta
2 taylor(4*asinh(delta/2)^2,delta)
```

Solution: The second derivative

$$\begin{aligned} \partial^2 &= \frac{1}{h^2}(h\partial)^2 \\ &= \frac{1}{h^2}[2\sinh^{-1}(\delta/2)]^2 \\ &= \frac{1}{h^2}[\delta^2-\tfrac{1}{12}\delta^4+\mathcal{O}(\delta^6)] \\ &\qquad \text{rewrite using shifts} \\ &= \frac{1}{h^2}(\mathcal{E}^{1/2}-\mathcal{E}^{-1/2})^2+\frac{1}{h^2}\mathcal{O}(\delta^4) \\ &= \frac{1}{h^2}(\mathcal{E}-2+\mathcal{E}^{-1})+\frac{1}{h^2}\mathcal{O}(h^4\partial^4) \end{aligned}$$

Hence $u'' = \frac{1}{h^2}(u(x+h)-2u(x)+u(x-h))+\mathcal{O}(h^2u'''')$. We could also expand and use δ^4 in the approximation of u'' and then the error would be $\mathcal{O}(h^4u^{(6)})$. ♣

Example 5.8 Discretise the first derivative using

```
1 syms delta
2 taylor(2*asinh(delta/2),delta)
3 taylor(2*asinh(delta/2)/sqrt(1+delta^2/4),delta)
```

and the mean operator.

Solution: The following does not work for us:

$$\begin{aligned}\partial &= \frac{1}{h}(h\partial)\\ &= \frac{1}{h}2\sinh^{-1}(\delta/2)\\ &= \frac{1}{h}[\delta - \tfrac{1}{24}\delta^3 + \mathcal{O}(\delta^5)]\\ &= \frac{1}{h}(\mathcal{E}^{1/2} - \mathcal{E}^{-1/2}) + \frac{1}{h}\mathcal{O}(\delta^3)\end{aligned}$$

as then $u'(x) \approx \frac{1}{h}[u(x+h/2) - u(x-h/2)]$ which, while correct and excellent in some circumstances, is not in terms of $u(x_j)$. Fix the problem with the mean operator:

$$\begin{aligned}\partial &= \frac{\mu}{h}[h\partial/\mu]\\ &= \frac{\mu}{h}\left[2\sinh^{-1}(\delta/2)/\sqrt{1+\delta^2/4}\right]\\ &= \frac{\mu}{h}[\delta - \tfrac{1}{6}\delta^3 + \mathcal{O}(\delta^5)]\\ &= \frac{1}{h}[\mu\delta - \tfrac{1}{6}\mu\delta^3 + \mathcal{O}(\delta^5)]\\ &= \frac{1}{2h}(\mathcal{E}^{1/2} + \mathcal{E}^{-1/2})(\mathcal{E}^{1/2} - \mathcal{E}^{-1/2}) + \frac{1}{h}\mathcal{O}(\delta^3)\\ &= \frac{1}{2h}(\mathcal{E} - \mathcal{E}^{-1}) + \frac{1}{h}\mathcal{O}(h^3\partial^3)\end{aligned}$$

as then $u'(x) \approx \frac{1}{2h}[u(x+h) - u(x-h)] + \mathcal{O}(h^2u''')$. We coded this for $(u^2)_x$. We could retain the term $\mu\delta^3$ in the approximation of u' and then the error would be $\mathcal{O}(h^4u^{(5)})$. ♣

Similarly for other derivatives in a PDE.

Rule of thumb: where possible, prefer to discretise in 'conservative' form. Rewrite the KdV as $u_t + \partial_x(3u^2 + u_{xx}) = 0$ which is of the conservative form $u_t + q_x = 0$. Consequently we discretised $6uu_x = 3(u^2)_x$ as $3[u(x+h)^2 - u(x-h)^2]/(2h)$, not $6u(x)[u(x+h) - u(x-h)]/(2h)$ because the latter usually behaves worse—try it.

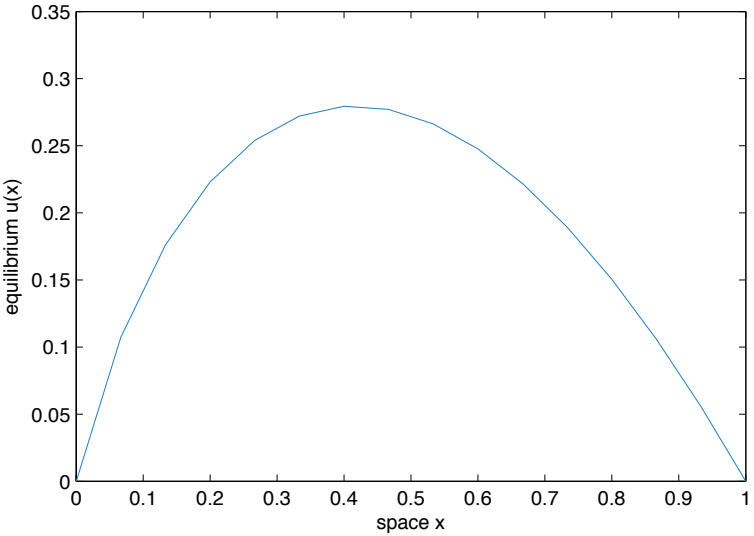

Figure 5.4: Algorithm 5.4 computes this equilibrium of the forced nonlinear PDE $u_t = uu_{xx} + 1 - x$.

5.2 Find equilibria

In applications we often want to find 'steady states', 'fixed points' or 'equilibria'. A not so quick and far too dirty approach is simply to computationally integrate for long times and hope an equilibrium emerges. Resist this temptation.

Instead encode a wrapper around your time derivative function to find a u such that $u_t = 0$. Several advantages: checks the function; and is in the spirit of code once, use many times; and is also able to find unstable equilibria.

Example 5.9 Find the equilibrium in Figure 5.4 of the forced nonlinear PDE $u_t = uu_{xx}+1-x$ for $0 < x < 1$ and boundary values $u(0,t) = u(1,t) = 0$. Algorithm 5.4 reuses and modifies the code of Algorirhm 5.2.

- The function code of Algorithm 5.4 is the exactly same (apart from adding the forcing term `1-x(j)`); the driver program is mostly the same.

- Modify title and date.
- Specify a smooth initial guess `u0` as we expect a smooth answer.
- Define a one-line function using `@` which just evaluates u_t at 'time 0' for any given u.
- Ask `fsolve` to find a solution of $u_t = 0$ starting with guess `u0`.
- Upon executing, `fsolve` reports success with the message

  ```
  Equation solved.

  fsolve completed because the vector of function
  values is near zero as measured by the default
  value of the function tolerance, and the problem
  appears regular as measured by the gradient.

  <stopping criteria details>
  ```

  Other messages indicate failure.
- The result is a column vector so wrap boundary values columnwise.
- Plot a 2D graph; and if needed print it to a file with the `print` command (using eps format as Matlab does best with eps).

Optional: solve variations on the theme: increase the number of points; or multiply the diffusion coefficient by x; starting from random guess often find unphysical negative equilibria; and so on. ♣

5.3 Numerical linearisation characterises solution dynamics

Recall that we often algebraically linearise equations to be able to find complete but approximate solutions analytically. Analogously, we use numerical linearisation to completely

Algorithm 5.4: use `fsolve` to find equilibria of the nonlinear PDE $u_t = uu_{xx} + 1 - x$.

```
1  function equilheatnon
2  % Compute equilibrium of nonlinear PDE
3  % via finite differences. AJR, 15/8/2014
4  global j x
5  nPoints=16
6  x=linspace(0,1,nPoints)'
7  j=2:nPoints-1
8  u0=ones(size(x))
9
10 fun=@(u) dudt(0,u);
11 u=fsolve(fun,u0(j))
12
13 u=[uleft(0);u;uright(0)];
14 plot(x,u)
15 xlabel('space␣x'),ylabel('equilibrium␣u(x)')
16 print -depsc2 equilheatnon
17 %--------------------------------------------------
18 function ut=dudt(t,u)
19 global j x
20 dx=x(2)-x(1);
21 u=[uleft(t);u;uright(t)];
22 ut=u(j).*(u(j-1)-2*u(j)+u(j+1))/dx^2+1-x(j);
23 %--------------------------------------------------
24 function ua=uleft(t)
25 ua=zeros(size(t));
26 %--------------------------------------------------
27 function ub=uright(t)
28 ub=zeros(size(t));
```

characterise the numerical code in the neighbourhood of an equilibrium. If the numerical characteristics do not match our analytic expectations, then we suspect errors or discretisation issues to resolve.

Form an implicit eigenproblem Given a PDE for $u(x,t)$, say. Suppose we have coded a discretisation $d\vec{u}/dt = \vec{f}(\vec{u})$, and found an equilibrium $\vec{u}^*$, which satisfies $\vec{f}(\vec{u}^*) = \vec{0}$. We seek the dynamics of small perturbations away from equilibrium: seek $\vec{u}(t) = \vec{u}^* + \epsilon\vec{v}e^{\lambda t}$ for small ϵ. The discretisation then reduces to an eigenproblem $\lambda\vec{v} = A\vec{v}$:

$$\begin{aligned}
& \frac{d\vec{u}}{dt} = \vec{f}(\vec{u}) \\
\implies \quad & \vec{0} + \lambda\epsilon\vec{v}e^{\lambda t} = \vec{f}(\vec{u}^* + \epsilon\vec{v}e^{\lambda t}) \\
\implies \quad & \lambda\epsilon\vec{v}e^{\lambda t} = \vec{f}(\vec{u}^*) + \vec{f}'(\vec{u}^*)\epsilon\vec{v}e^{\lambda t} + \mathcal{O}(\epsilon^2) \\
\implies \quad & \lambda\epsilon\vec{v}e^{\lambda t} \approx A\epsilon\vec{v}e^{\lambda t} \quad \text{where } A = \left.\frac{\partial\vec{f}}{\partial\vec{u}}\right|_{\vec{u}^*} \\
\implies \quad & \lambda\vec{v} = A\vec{v}
\end{aligned}$$

It appears we need the matrix A of partial derivatives of the coded function—but this is too tedious. But instead of finding the matrix A, we invoke the procedure `eigs()` that uses just the product $A\vec{v}$: this product we get direct from the function $\vec{f}$ by evaluating $\vec{f}(\vec{u}^* + \epsilon\vec{v})/\epsilon$ for some small ϵ, say 10^{-6}. It would be $[\vec{f}(\vec{u}^* + \epsilon\vec{v}) - \vec{f}(\vec{u}^*)]/\epsilon$ but we know $\vec{f}(\vec{u}^*) = \vec{0}$.

Example 5.10 Compute the eigenvalues and eigenvectors of the coded discretisation of the heat PDE:

$$\frac{du_j}{dt} = u_{xx}(x_j,t) \approx \frac{u_{j-1} - 2u_j + u_{j+1}}{h^2},$$

for $0 < x < 1$ with homogeneous boundary conditions $u(0,t) = u(1,t) = 0$. Here the equilibrium is the zero solution $u(x,t) = 0$. Algorithm 5.5 invokes `eigs()` in Matlab.

Solution: Explain Algorithm 5.5.

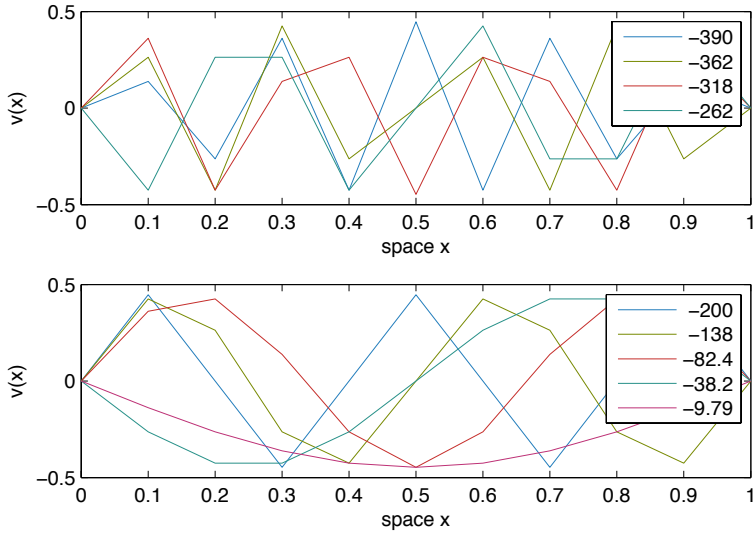

Figure 5.5: Algorithm 5.5 computes these eigenvalues and eigenvectors of the coded discretisation of the heat PDE $u_t = u_{xx}$*: the bottom ones a reasonably accurate; the top ones are not.*

- Reuse: most of the code is the same as Algorithm 5.1. Importantly, the time derivative function is identical, and the boundary conditions just changed to homogeneous, so the results here correspond precisely to what happens elsewhere.

9. Set `u0` to the equilibrium.

11. Set `small` to the size of the perturbation.

12. Define a function that estimates '$A\vec{v}$' from the coded time derivative function using $\vec{f}(\vec{u}^* + \epsilon\vec{v})/\epsilon$.

13. The Matlab function `[v,d]=eigs(fun,n,k)` then computes `k` eigenvalues and corresponding `k` eigenvectors of a matrix A given only that `fun` computes the product $\texttt{fun}(\vec{v}) = A\vec{v}$ and that $\vec{v} \in \mathbb{R}^{\texttt{n}}$. The eigenvectors are stored in the `k` columns of `v`, and the eigenvalues are stored in the diagonal of `d`. Here ask for all eigenvalues and eigenvectors.

15–16. Compare the discretisation eigenvalues with the known analytic ones and see the smaller eigenvalues are reasonably accurate: the discrepancy is from the discretisation, not from `eigs()`.

17–24. Plot the eigenvectors in two groups using `subplot()`, with a legend listing the corresponding eigenvalue.

The eigenvectors show the correct sinusoidal like oscillations.

♣

Algorithm 5.5: use numerical linearisation to characterise the coded discretiation of the heat PDE. This framework adapts to many similar PDE problems.

```
function heateig
% Compute spectrum of heat PDE with finite differences
% knowing zero is an equilibrium.
% AJR, 13/8/2014
global j x
nPoints=11
x=linspace(0,1,nPoints)'
j=2:nPoints-1
u0=zeros(size(x))

small=1e-6
fun=@(v) dudt(0,u0(j)+small*v)/small;
[v,d]=eigs(fun,nPoints-2,nPoints-2)

d=diag(d)'
analytic=-(nPoints-2:-1:1).^2*pi^2
v=[uleft(d);v;uright(d)];
subplot(2,1,1), plot(x,v(:,1:4))
legend(num2str(d(1:4)',3))
xlabel('space␣x'),ylabel('v(x)')
subplot(2,1,2), plot(x,v(:,end-4:end))
legend(num2str(d(end-4:end)',3))
xlabel('space␣x'),ylabel('v(x)')
print -depsc2 heateig
%-----------------------------------------------------
function ut=dudt(t,u)
global j x
dx=x(2)-x(1);
u=[uleft(t);u;uright(t)];
ut=(u(j-1)-2*u(j)+u(j+1))/dx^2;
%-----------------------------------------------------
function ua=uleft(t)
ua=zeros(size(t));
%-----------------------------------------------------
function ub=uright(t)
ub=zeros(size(t));
```