Chapter 13 - Boundary Value Problems and shooting

MATH3018/6141, Semester 1

Boundary Value Problems

For IVPs need both the system of differential equations

$$\boldsymbol{y}'(x) = \boldsymbol{f}(x, \boldsymbol{y}(x))$$

and the initial data $\boldsymbol{y}_0 = \boldsymbol{y}(0)$ to specify unique solution.

Instead specify solution in $x \in [0, b]$ by giving some data at $x = 0$ and some at $x = b$. Given exactly n independent conditions we normally have (although are not guaranteed) a unique solution. This is a *Boundary Value Problem* (BVP).

A single ODE ($n = 1$) is always an IVP. Simplest BVPs occur for systems of size $n = 2$. Typically write this as a single equation,

$$y'' = f(x, y, y'), \quad y(a) = A, \; y(b) = B, \quad x \in [a, b].$$

Can also have problems with boundary conditions in which only y' appears or, more generally, ones in which y and y' are combined.

Shooting Method

Simple idea:

1. convert BVP to IVP by *guessing* additional initial conditions $\boldsymbol{y}_a$;
2. solve IVP and check (dis)agreement with conditions for end point, $\boldsymbol{y}_b$;
3. modify initial guess until solved.

More formally, look at simple problem

$$y'' = f(x, y, y'), \quad y(a) = A, \; y(b) = B, \quad x \in [a, b].$$

Convert to IVP

$$y'' = f(x, y, y'), \quad y(a) = A, \; y'(a) = z$$

where z is initially unknown. Solution y depends on x, but also on z.

Aim: Find z such that $y(b) = B$.

Shooting and root finding

Write solution to IVP

$$y'' = f(x, y, y'), \quad y(a) = A, \; y'(a) = z$$

as $y(x, z)$ to encode dependence on z. Condition to enforce is

$$y(b, z) = B.$$

Define function

$$\phi(z) = y(b, z) - B.$$

The root $\phi(z) = 0$ gives required value of z.

Shooting involves finding root of a nonlinear function; evaluating function at any point involves solving an IVP!

Example

Consider problem

$$y'' + y' + 1 = 0, \quad y(0) = 0, \; y(1) = 1, \quad x \in [0, 1].$$

First reformulate as the IVP

$$y'' + y' + 1 = 0, \quad y(0) = 0, \; y'(0) = z.$$

Then reformulate IVP in first order form,

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 0 \\ y_2' &= -1 - y_2, & y_2(0) &= z. \end{aligned}$$

Want to solve this IVP, modifying z until

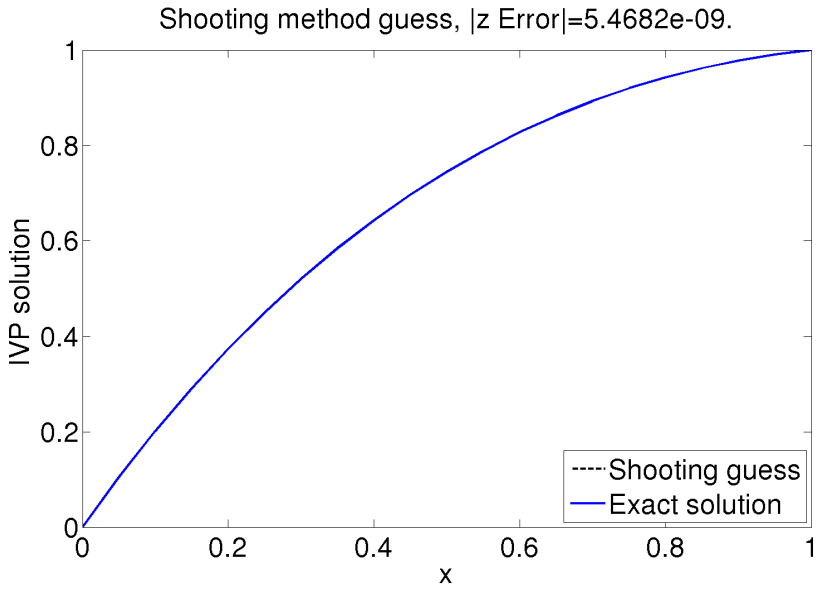
$$\phi(z) = y(1) - 1 = y_1(1) - 1 = 0.$$

Example (continued)

Use library RKF45 to solve IVP, and secant method to find root, to solve

$$\begin{aligned} y'' + y' + 1 &= 0, \\ y(0) &= 0, \\ y(1) &= 1, \\ x &\in [0, 1]. \end{aligned}$$

Secant method requires two guesses for z; use $0, 10$ which bracket the root.



Example (continued)

The problem

$$y'' + y' + 1 = 0, \quad y(0) = 0, \; y(1) = 1, \quad x \in [0, 1]$$

can be solved "analytically" using shooting. The new IVP

$$y'' + y' + 1 = 0, \quad y(0) = 0, \; y'(0) = z$$

can be solved directly to find

$$y(x, z) = (1 + z)(1 - e^{-x}) - x.$$

Then find z such that solution satisfies boundary condition $y(1, z) = 1$; equivalently, find the root of

$$\phi(z) = y(1, z) - 1.$$

This gives the exact solution

$$y(x) = \frac{2e}{e - 1}(1 - e^{-x}) - x.$$

Root finding

For shooting method need to

1. solve IVPs (previous section)
2. find roots of nonlinear equations.

For root finding, construct a sequence that converges to root; often use *secant* method

$$z_{n+1} = z_n - \phi(z_n)\frac{z_n - z_{n-1}}{\phi(z_n) - \phi(z_{n-1})}$$

or *Newton's* method

$$z_{n+1} = z_n - \frac{\phi(z_n)}{\phi'(z_n)}.$$

Secant method needs two guesses and one function evaluation; Newton needs one guess but two function evaluations, one of ϕ'.

Secant usually preferable: simpler, normally just as efficient. Sometimes the better convergence of Newton's method useful.

Newton's method for root finding

Newton's method

$$z_{n+1} = z_n - \frac{\phi(z_n)}{\phi'(z_n)}$$

requires the derivative of the function

$$\phi(z) = y(b, z) - B.$$

Could evaluate numerically using

$$\phi'(z_n) \simeq \frac{\phi(z_n + h) - \phi(z_n)}{h}, \quad h \ll 1.$$

Essentially the secant method, only less robust.

Alternative: Write $u(x, z) = \partial_z y$ and differentiate the *entire IVP*

$$y'' = f(x, y, y'), \quad y(a) = A, \; y'(a) = z$$

with respect to z. This gives following *Variational problem* for $u(x, z)$:

$$u'' = (\partial_y f)u + (\partial_{y'} f)u', \quad u(a) = 0, \; u'(a) = 1.$$

Solve this variational problem together with problem for $y(x, z)$ to find $\phi'(z) = u(b, z)$ which can then be used in Newton's Method.

Problems with shooting

For many problems shooting is effective: try it first. However, can fail in some cases:

1. Fails to satisfy all the boundary conditions. Usually a minor point, but can be crucial in special cases.
2. Root finding requires initial guess(es). With a poor guess method may not converge.
3. BVP may have multiple solutions: shooting may converge to "wrong" one.
4. IVP generated may be unstable, even for a well-behaved BVP.
5. Considerably harder to implement for higher order problems as this involves finding roots to a nonlinear system with more than one independent variable.

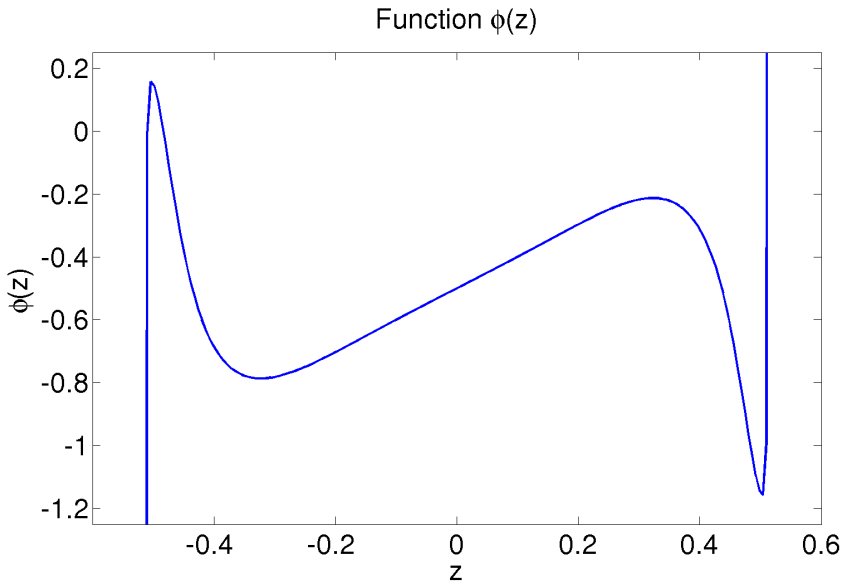
Multiple solutions

The example BVP

$$y'' + y - 2y\tanh(y^2) = 0,$$
$$y(0) = 0,$$
$$y(8) = 1/2$$

has more than one solution; resulting ϕ for root-find has strange behaviour.

Zooming, see the roots are very close together; corresponding solutions have different signs!

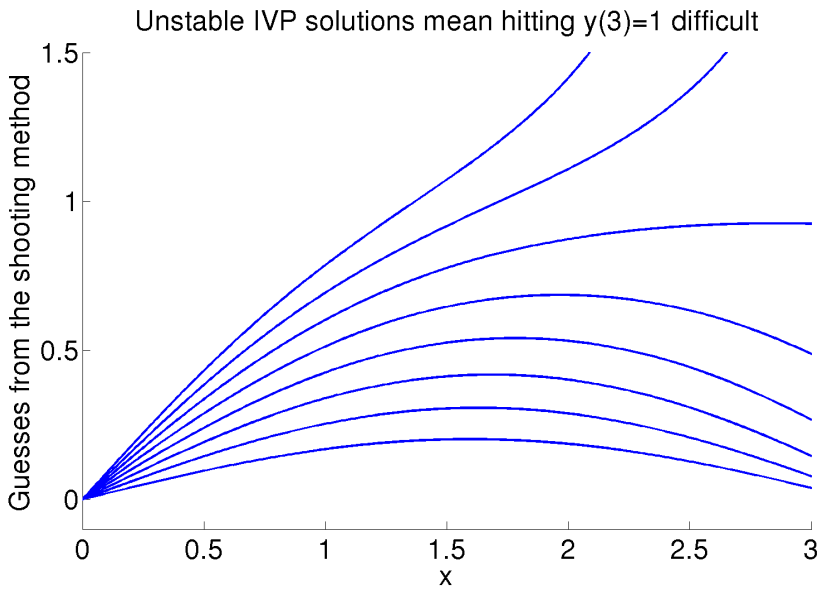


Unstable IVP

The example BVP

$$y'' + y - y^3 = 0,$$
$$y(0) = 0,$$
$$y(3) = 1$$

has well-behaved single solution. The IVP solutions have very different behaviour as z is varied. For many values of z the IVP method diverges before finding root. Shooting therefore does not work well on this problem.



Summary

- Boundary value problems (BVPs) are ODEs where not all the conditions required to specify the problem on the interval $x \in [a, b]$ are given at a single point.
- Shooting methods convert the BVP to an IVP with some free initial data; the free data is modified to fit the boundary conditions at the "end" of the interval. This requires nonlinear root finding.
- The secant method is the standard nonlinear root finding method used; if the additional convergence of Newton's method is required the variational equation is typically needed to evaluate the derivative of the nonlinear function.
- Shooting methods, when they work, are fast and accurate. There are some problems for which the BVP is well-posed but the IVP used in the shooting method is not (*e.g.* the IVP has infinite solutions for some values of z).

Chapter 14 - Finite difference/Relaxation methods for Boundary Value Problems

MATH3018/6141, Semester 1

Boundary Value Problems

Considering simple boundary value problem

$$y'' = f(x, y, y'), \quad y(a) = A, \; y(b) = B, \quad x \in [a, b].$$

Boundary conditions are only examples (other boundary conditions can occur).

Standard approach: first try *shooting* method. Convert BVP to IVP with some initial data free. Free data then modified, using root finding, to satisfy boundary conditions.

Shooting is straightforward and efficient – when it works. Otherwise use *relaxation* methods based on *finite differences*.

The linear problem

Consider the linear problem

$$y'' + p(x)y' + q(x)y = f(x), \quad y(a) = A, \; y(b) = B, \quad x \in [a, b].$$

Introduce a *grid*, evenly spaced over the interval,

$$x_i = a + ih, \quad i = 0, 1, \ldots, n+1, \quad h = \frac{b-a}{n+1}.$$

Contains n *interior* points and 2 boundary points. The value of y at the boundary points given by boundary conditions,

$$y_0 = y(a) = A,$$
$$y_{n+1} = y(b) = B.$$

Value of y at interior points unknown; these give approximate solution. Require approximation to converge to true solution as $h \to 0$.

Finite differences

Given grid $\{x_j\}$, approximate derivatives using standard finite difference formulas. Typically use centred differencing formulas

$$y'(x_i) = \frac{y_{i+1} - y_{i-1}}{2h} + \mathcal{O}(h^2),$$
$$y''(x_i) = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + \mathcal{O}(h^2),$$

which are second order accurate.

Substituting into BVP equation

$$y'' + p(x)y' + q(x)y = f(x), \quad y(a) = A, \; y(b) = B, \quad x \in [a, b]$$

for *interior* points gives finite difference formula ($q_i = q(x_i)$ etc.)

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p_i \left(\frac{y_{i+1} - y_{i-1}}{h} \right) + q_i y_i = f_i$$

Constructing the linear system

Thus obtain a system of (n) linear algebraic equations for $y_1, y_2, \cdots, y_n$

$$y_{i-1}\left(1-\tfrac{h}{2}p_i\right)+y_i\left(h^2q_i-2\right)+y_{i+1}\left(1+\tfrac{h}{2}p_i\right)=h^2f_i, \quad 1\le i\le n.$$

Recall that we already know $y_0 = A$ and $y_{n+1} = B$ from the boundary conditions.

Here the formula for the interior points obtained by multiplying equation on previous slide by h^2. This is a linear system of form

$$T\boldsymbol{y}=\boldsymbol{F}$$

for $n \times n$ matrix T.

Take $n = 3$ for example. In the interior

$$\begin{pmatrix} -2+h^2q_1 & 1+\frac{h}{2}p_1 & 0 \\ 1-\frac{h}{2}p_2 & -2+h^2q_2 & 1+\frac{h}{2}p_2 \\ 0 & 1-\frac{h}{2}p_3 & -2+h^2q_3 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}=\begin{pmatrix} h^2f_1 \\ h^2f_2 \\ h^2f_3 \end{pmatrix}-\begin{pmatrix} (1-\frac{h}{2}p_0)y_0 \\ 0 \\ (1+\frac{h}{2}p_3)y_4 \end{pmatrix}.$$

Constructing the linear system

Thus obtain a system of (n) linear algebraic equations for $y_1, y_2, \cdots, y_n$

$$y_{i-1}\left(1-\tfrac{h}{2}p_i\right)+y_i\left(h^2q_i-2\right)+y_{i+1}\left(1+\tfrac{h}{2}p_i\right)=h^2f_i, \quad 1\le i\le n.$$

Recall that we already know $y_0 = A$ and $y_{n+1} = B$ from the boundary conditions.

Here the formula for the interior points obtained by multiplying equation on previous slide by h^2. This is a linear system of form

$$T\boldsymbol{y}=\boldsymbol{F}$$

for $n \times n$ matrix T.

The full system for $n = 3$ is thus

$$\begin{pmatrix} -2+h^2q_1 & 1+\frac{h}{2}p_1 & 0 \\ 1-\frac{h}{2}p_2 & -2+h^2q_2 & 1+\frac{h}{2}p_2 \\ 0 & 1-\frac{h}{2}p_3 & -2+h^2q_3 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}=\begin{pmatrix} h^2f_1-A(1-\frac{h}{2}p_1) \\ h^2f_2 \\ h^2f_3-B(1+\frac{h}{2}p_3) \end{pmatrix}.$$

Points about the linear system

Important point: provided central differencing is used, the matrix is *tridiagonal*. Very efficient algorithms exist for solving such *tridiagonal* linear systems (*e.g.* LU decomposition).

Also note that here the matrix T is not modified by the boundary conditions. Always true for Dirichlet type conditions which fix $y(a)$. Not true for Neumann type conditions which fix $y'(a)$, for example.

Example

For the problem

$$y'' + y' + 1 = 0, \quad y(0) = 0,\ y(1) = 1, \quad x \in [0, 1]$$

we have $p(x) = 1$, $q(x) = 0$, $f(x) = -1$. Look at grid with 3 points ($n = 1, h = 1/2$). Gives the equations

$$\begin{aligned} y_0 &= 0, \\ y_0\left(1 - \tfrac{h}{2} \cdot 1\right) + y_1\left(h^2 \cdot 0 - 2\right) + y_2\left(1 + \tfrac{h}{2} \cdot 1\right) &= h^2 \cdot (-1), \\ y_2 &= 1. \end{aligned}$$

Boundary points y_0, y_2 given by boundary conditions and are exact. Central point y_1 has value

$$y_1 = \tfrac{3}{4}; \quad y(x = 1/2) = 0.74492.$$

Good accuracy for such a coarse grid, as $y(x)$ close to linear.

Example: 2

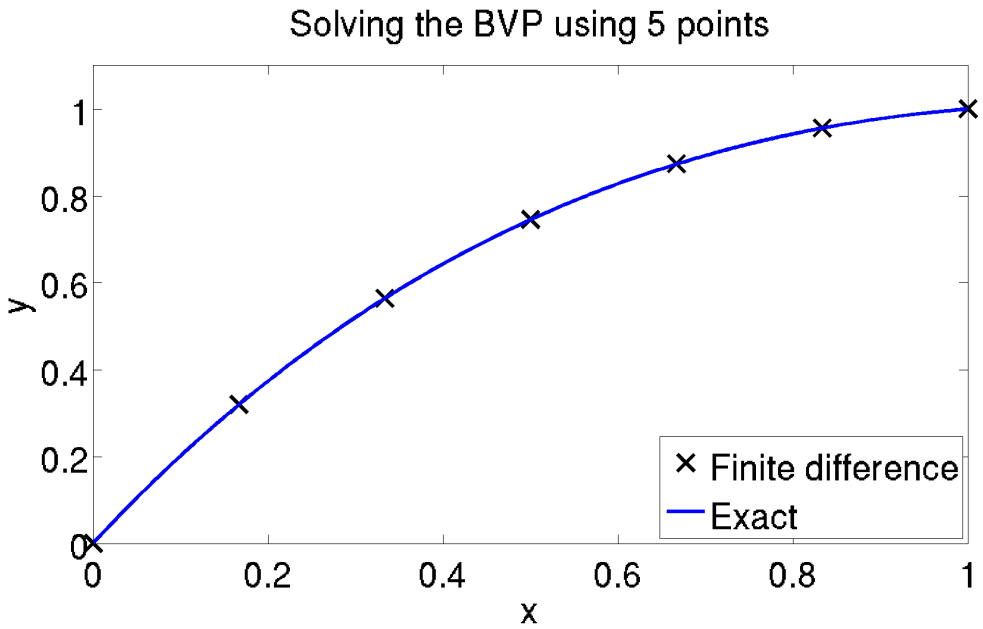
The problem

$$y'' + y' + 1 = 0,$$
$$y(0) = 0,$$
$$y(1) = 1,$$
$$x \in [0, 1]$$

solved with finite differences is impressively accurate.

Increase n the result converges well.

The convergence is second order in h.



Example: 2

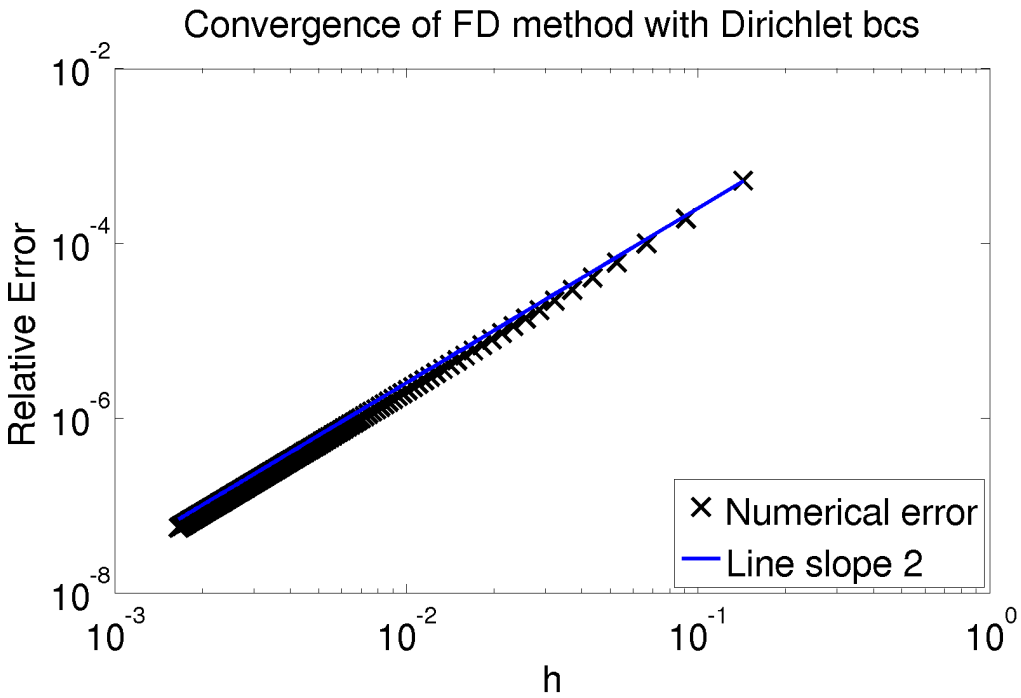
The problem

$$\begin{aligned} y'' + y' + 1 &= 0, \\ y(0) &= 0, \\ y(1) &= 1, \\ x &\in [0, 1] \end{aligned}$$

solved with finite differences is impressively accurate.

Increase n the result converges well.

The convergence is second order in h.



Boundary conditions

When constructing the linear system

$$T\boldsymbol{y} = \boldsymbol{F}$$

using Dirichlet type boundary conditions the boundary conditions do not alter matrix T they only modify vector $\boldsymbol{F}$.

If instead have Neumann type boundary conditions, e.g. $y'(b) = \alpha$, the situation changes. First need finite difference approximation of *boundary condition itself*. For example could use first order differencing:

$$\frac{y_{n+1} - y_n}{h} = \alpha.$$

Then rearrange to find condition on boundary point y_{n+1}:

$$y_{n+1} = y_n + h\alpha.$$

Boundary conditions

When constructing the linear system

$$T\boldsymbol{y} = \boldsymbol{F}$$

using Dirichlet type boundary conditions the boundary conditions do not alter matrix T they only modify vector $\boldsymbol{F}$.

$$y'(b) = \alpha \rightarrow y_{n+1} = y_n + h\alpha.$$

Note that

1. The value of y_{n+1} is now *not specified* but that we now have an extra finite difference equation $y_{n+1} = y_n + h\alpha$ in order to determine it.
2. Use condition on y_{n+1} to construct the linear system, giving new terms including y_n; hence matrix T is modified as well as $\boldsymbol{F}$. In particular system size is now $n + 1$ rather than n.

Example

For the problem

$$y'' + y' + 1 = 0, \quad y(0) = 0,\ y'(1) = \tfrac{3-e}{e-1}, \quad x \in [0,1]$$

have same solution as before. Look at grid with 3 points ($n = 1, h = 1/2$). Gives equations (two match Dirichlet case)

$$\begin{aligned} y_0 &= 0, \\ y_0\left(1 - \tfrac{h}{2}\cdot 1\right) + y_1\left(h^2 \cdot 0 - 2\right) + y_2\left(1 + \tfrac{h}{2}\cdot 1\right) &= h^2 \cdot (-1), \\ y_2 &= y_1 + h\tfrac{3-e}{e-1}. \end{aligned}$$

Substituting in expressions for y_0, y_2, find central point y_1 has value

$$\begin{aligned} & y_1 = \tfrac{1}{3} + \tfrac{5\alpha}{6} = 0.470; && y_e(x = 1/2) = 0.74492. \\ \Rightarrow \quad & y_2 = 0.470 + h\alpha = 0.552; && y_e(x = 1) = 1. \end{aligned}$$

This result is much worse than the Dirichlet case.

Example: 2

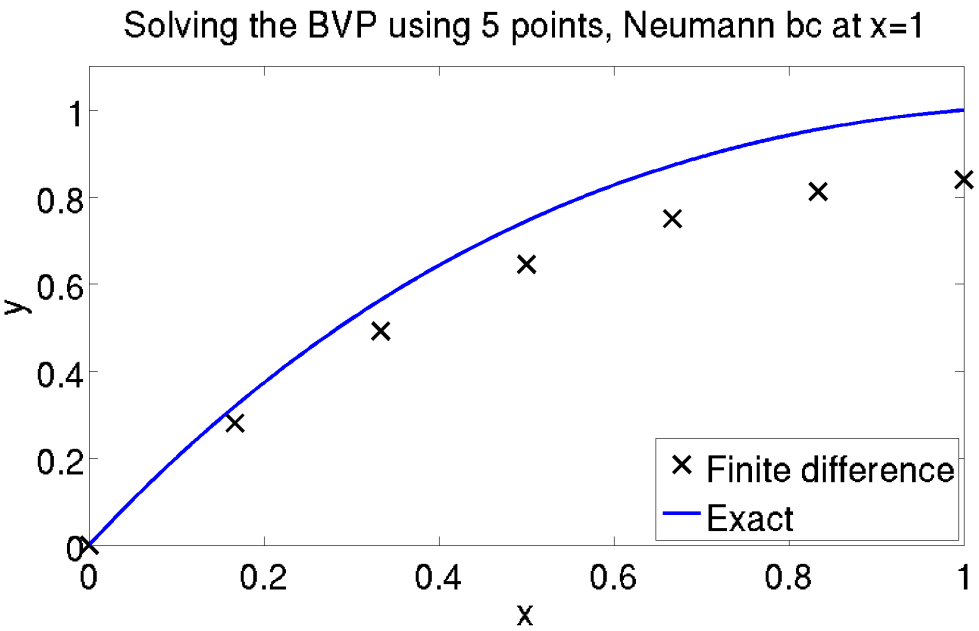
The approximate solution to

$$\begin{aligned} y'' + y' + 1 &= 0, \\ y(0) &= 0, \\ y'(1) &= \tfrac{3-e}{e-1}, \\ x &\in [0, 1] \end{aligned}$$

is not as good as Dirichlet case.

Increasing n the result converges eventually.

Convergence only first order in h because difference formula for derivative $y'(1)$ only first order.



Example: 2

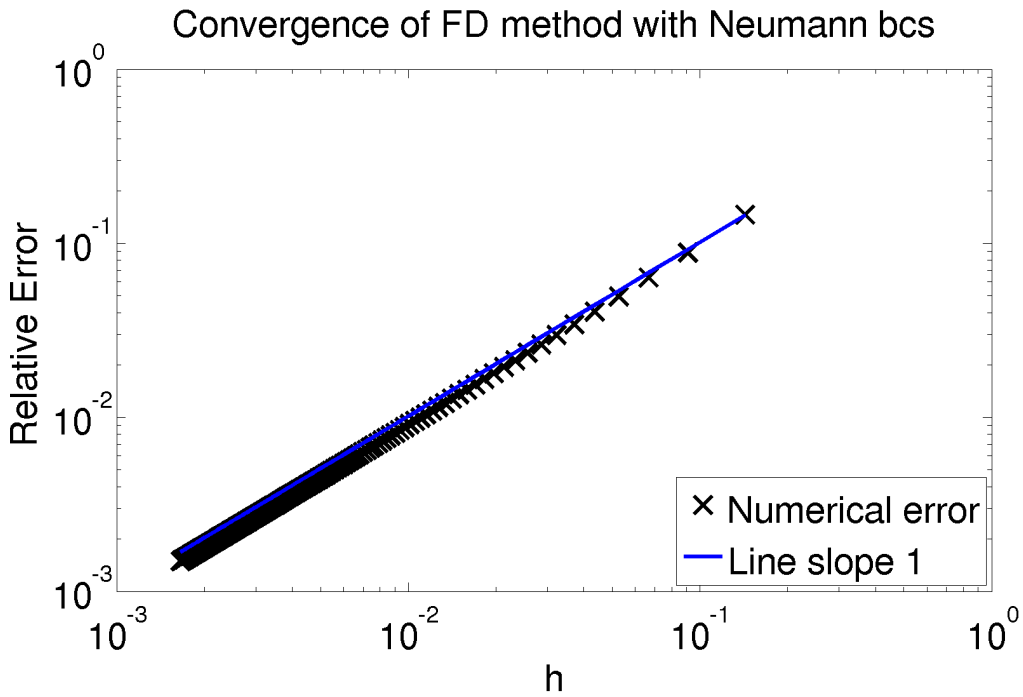
The approximate solution to

$$\begin{aligned} y'' + y' + 1 &= 0, \\ y(0) &= 0, \\ y'(1) &= \tfrac{3-e}{e-1}, \\ x &\in [0, 1] \end{aligned}$$

is not as good as Dirichlet case.

Increasing n the result converges eventually.

Convergence only first order in h because difference formula for derivative $y'(1)$ only first order.



Second order differencing for the Neumann condition

- Introduce extra "ghost" point $x_{n+2} = b + h$ outside the domain.
- On the edge of domain at $x = x_{n+1}$ the equation is satisifed and hence the $n+1$'th difference equation is

$$y_n\left(1 - \tfrac{h}{2}p_{n+1}\right) + y_{n+1}\left(h^2 q_{n+1} - 2\right) + y_{n+2}\left(1 + \tfrac{h}{2}p_{n+1}\right) = h^2 f_{n+1}$$

- But can now approximate the Neumann condition $y'(b) = \alpha$ by the central difference

$$\frac{y_{n+2} - y_n}{2h} = \alpha$$

- The error in boundary condition approximation is now second order (*i.e.* $O(h^2)$).
- But we now have as system of $n+2$ linear equations for the $n+2$ unknowns $y_1, y_2, \cdots, y_{n+2}$.

Example: 3

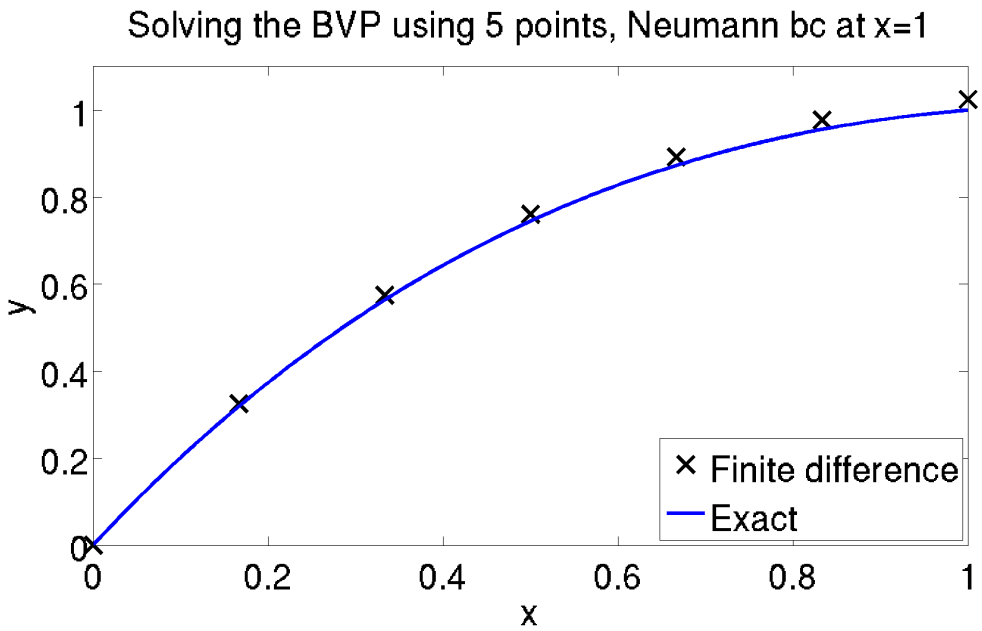
Take same problem

$$y'' + y' + 1 = 0,$$
$$y(0) = 0,$$
$$y'(1) = \frac{3-e}{e-1},$$
$$x \in [0, 1]$$

using second order approximation to derivative in boundary condition gives better results.

Increasing n the result converges faster.

Convergence is second order in h.



Example: 3

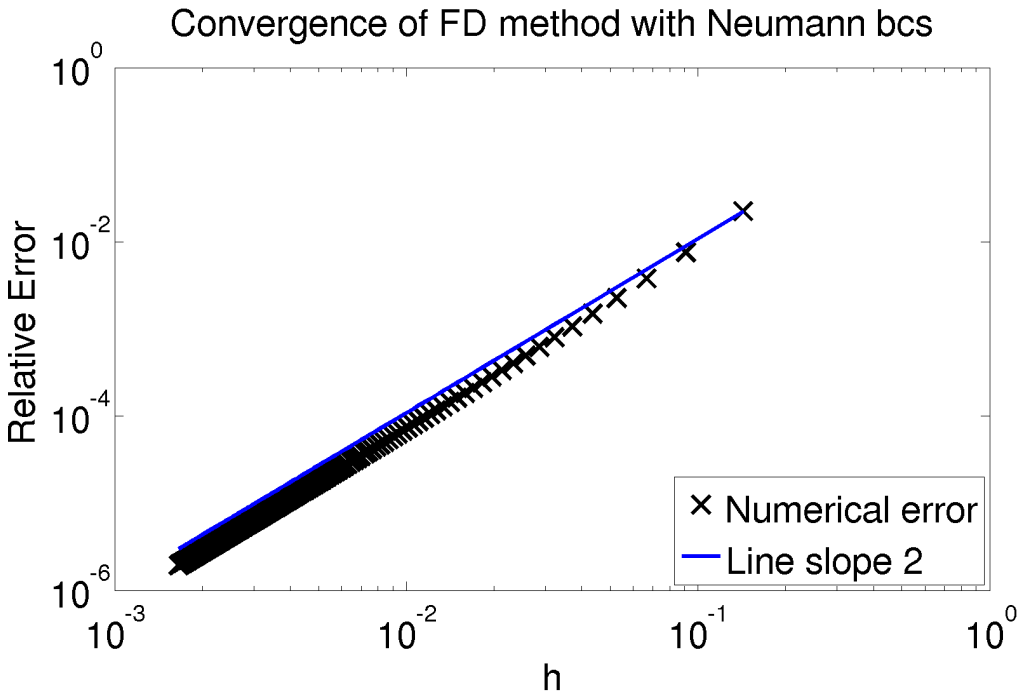
Take same problem

$$y'' + y' + 1 = 0,$$
$$y(0) = 0,$$
$$y'(1) = \frac{3-e}{e-1},$$
$$x \in [0, 1]$$

using second order approximation to derivative in boundary condition gives better results.

Increasing n the result converges faster.

Convergence is second order in h.



Error analysis

Central differencing methods normally converge at second order. Examples above suggest second order convergence for the BVP.

Define error vector $\boldsymbol{e}$ so that i'th entry $(\boldsymbol{e})_i = y(x_i) - y_i$. Then calculate $T\boldsymbol{e}$. The i'th entry of this vector, for the interior entries, is

$$(T\boldsymbol{e})_i = (y(x_{i-1}) - y_{i-1})\left(1 - \tfrac{h}{2}p_i\right) - (y(x_i) - y_i)\left(2 - h^2 q_i\right) + (y(x_{i+1}) - y_{i+1})\left(1 + \tfrac{h}{2}p_i\right).$$

But

$$y_{i-1}\left(1 - \tfrac{h}{2}p_i\right) - y_i\left(2 - h^2 q_i\right) + y_{i+1}\left(1 + \tfrac{h}{2}p_i\right) = h^2 f_i.$$

And Taylor expanding exact solution $y(x)$ gives

$$y(x_{i-1})\left(1 - \tfrac{h}{2}p_i\right) - y(x_i)\left(2 - h^2 q_i\right) + y(x_{i+1})\left(1 + \tfrac{h}{2}p_i\right) = h^2 f_i + O(h^4).$$

Hence

$$(T\boldsymbol{e})_i = O(h^4) \quad \Longrightarrow \quad T\boldsymbol{e} = h^4 \boldsymbol{G}$$

for some a vector $\boldsymbol{G}$ that is independent of h.

Error analysis: 2

The error satisfies a linear system

$$T\boldsymbol{e} = h^4 \boldsymbol{G}$$

where T is the same tridiagonal matrix that defines the method and $\boldsymbol{G}$ is a vector that is independent of h.

Hence bound the error using

$$\begin{aligned} \|\boldsymbol{e}\| &= h^4 \|T^{-1}\boldsymbol{G}\| \\ &\leq h^4 \|T^{-1}\| \|\boldsymbol{G}\| \end{aligned}$$

Estimate for global error given by

$$\|\boldsymbol{e}\|_1 = \sum_{i=1}^{n} |e_i|$$

Try to estimate size of $\|T^{-1}\|_1$ and $\|G\|_1$ for $n \times n$ matrix T^{-1} and n-vector $\boldsymbol{G}$ since

$$\|\boldsymbol{e}\|_1 \leq h^4 \|T^{-1}\|_1 \|\boldsymbol{G}\|_1$$

Error analysis: 3

T^{-1} is a full $n \times n$ matrix with order one entries therefore since

$$\|T^{-1}\|_1 = \max(\|\boldsymbol{c}_1\|_1, \|\boldsymbol{c}_2\|_1, \cdots, \|\boldsymbol{c}_n\|_1)$$

and each of the column vectors $\boldsymbol{c}_j$ are full n-vectors we expect

$$\|T^{-1}\|_1 = O(n) = O(1/h).$$

Similarly we expect that for the n-vector $\boldsymbol{G}$

$$\|\boldsymbol{G}\|_1 = O(n) = O(1/h).$$

It follows from

$$\|\boldsymbol{e}\|_1 \leq h^4 \|T^{-1}\|_1 \|\boldsymbol{G}\|_1$$

that the global error

$$\|\boldsymbol{e}\|_1 \leq Kh^2$$

for some constant K. I.E. the global error is quadratic in h.

Nonlinear problems

In linear case

$$y'' + p(x)y' + q(x)y = f(x), \quad y(a) = A, \; y(b) = B, \quad x \in [a, b]$$

approximating solution on a grid leads to a linear system of form $T\boldsymbol{y} = \boldsymbol{F}$. In nonlinear case

$$y'' - f(x, y, y') = 0, \quad y(a) = A, \; y(b) = B, \quad x \in [a, b]$$

f depends on unknowns so do not obtain a linear system for $\boldsymbol{y}$.

Strategy: Use finite differences to write as a system of nonlinear equations

$$\boldsymbol{r}(\boldsymbol{y}) = \boldsymbol{0}$$

Here $\boldsymbol{y}$ is the vector of values of the solution on finite difference grid.

Solve this system of nonlinear equations using Newton's method.

The process of computing a sequence of guesses, each of which requires the solution of a linear system, is called *relaxation*.

Newton relaxation

Use finite differences to approximate nonlinear problem on grid $x = a + ih$ (where $i = 1, \cdots, n$) as system of nonlinear equations $\boldsymbol{r}(\boldsymbol{y}) = \boldsymbol{0}$.

The components of the equation vector $\boldsymbol{r}$ are given by

$$r_i = y_{i-1} + y_{i+1} - 2y_i - h^2 f\left(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h}\right).$$

The r_i are also sometimes called the *residuals*

Nonlinear root finding problem – find $\boldsymbol{y}$ so that $\boldsymbol{r}(\boldsymbol{y}) = \boldsymbol{0}$.

Guess a solution $\boldsymbol{y}^{(k)}$ and calculate $\boldsymbol{r}(\boldsymbol{y}^{(k)})$.

If $\boldsymbol{r}(\boldsymbol{y}^{(k)}) = \boldsymbol{0}$ then $\boldsymbol{y}^{(k)}$ is a solution.

Newton relaxation

If $\boldsymbol{r}(\boldsymbol{y}^{(k)}) \neq \boldsymbol{0}$ then want to calculate the correction vector $\boldsymbol{c}$ defined by

$$\boldsymbol{y} = \boldsymbol{y}^{(k)} + \boldsymbol{c}$$

To do this write $\boldsymbol{r}(\boldsymbol{y}) = \boldsymbol{r}(\boldsymbol{y}^{(k)} + \boldsymbol{c})$.
Recall that $\boldsymbol{r}(\boldsymbol{y}) = \boldsymbol{0}$ and Taylor expand this about $\boldsymbol{y}^{(k)}$:

$$\boldsymbol{0} = \boldsymbol{r}(\boldsymbol{y}^{(k)} + \boldsymbol{c}) = \boldsymbol{r}(\boldsymbol{y}^{(k)}) + J(\boldsymbol{y}^{(k)}) \cdot \boldsymbol{c} + \cdots$$

J is the Jacobian matrix with components $J_{ij} = \partial r_i / \partial y_j$.
Thus if $\boldsymbol{y}^{(k)}$ lies close to the solution $\boldsymbol{y}$ then

$$\boldsymbol{r}(\boldsymbol{y}^{(k)}) + J(\boldsymbol{y}^{(k)}) \cdot \boldsymbol{c} \approx \boldsymbol{0}.$$

This is a *linear* system for $\boldsymbol{c}$ and suggests that the next guess $\boldsymbol{y}^{(k+1)}$ should be defined by

$$\boldsymbol{y}^{(k+1)} = \boldsymbol{y}^{(k)} + \boldsymbol{c}, \quad \boldsymbol{c} = -J^{-1}(\boldsymbol{y}^{(k)}) \cdot \boldsymbol{r}(\boldsymbol{y}^{(k)})$$

The algorithm

Construct initial guess $\boldsymbol{y}^{(0)}$ (conventionally zero). Then iteratively

1. construct the residual

$$r_i^{(k)} = y_{i-1}^{(k)} + y_{i+1}^{(k)} - 2y_i^{(k)} - h^2 f\left(x_i, y_i^{(k)}, \frac{y_{i+1}^{(k)} - y_{i-1}^{(k)}}{2h}\right),$$

2. construct the Jacobian matrix $J(\boldsymbol{y}^{(k)})$ with components

$$J_{ij}^{(k)} = \frac{\partial r_i}{\partial y_j}(\boldsymbol{y}^{(k)}),$$

3. construct the correction $\boldsymbol{c}$ by solving

$$J^{(k)}\boldsymbol{c}^{(k)} = -\boldsymbol{r}^{(k)},$$

4. construct the new guess by

$$\boldsymbol{y}^{(k+1)} = \boldsymbol{y}^{(k)} + \boldsymbol{c}^{(k)}.$$

Repeat until the correction is sufficiently small.

The Jacobian

Choosing central differencing the Jacobian

$$J^{(k)} = \frac{\partial r_i^{(k)}}{\partial y_j^{(k)}}$$

is also tridiagonal, as

$$\frac{\partial r_i}{\partial y_j} = \begin{cases} 0 & j < i-1, \\ 1 + \frac{h}{2}\frac{\partial f}{\partial y'}\left(x_i, y_i^{(k)}, \frac{1}{2h}(y_{i+1}^{(k)} - y_{i-1}^{(k)})\right) & j = i-1, \\ -2 - h^2\frac{\partial f}{\partial y}\left(x_i, y_i^{(k)}, \frac{1}{2h}(y_{i+1}^{(k)} - y_{i-1}^{(k)})\right) & j = i, \\ 1 - \frac{h}{2}\frac{\partial f}{\partial y'}\left(x_i, y_i^{(k)}, \frac{1}{2h}(y_{i+1}^{(k)} - y_{i-1}^{(k)})\right) & j = i+1, \\ 0 & j > i+1. \end{cases}$$

Solution of each individual system is still very efficient. Take advantage of structure. Use LU-decomposition or appropriate variants of Gauss-Seidel or SOR iterative schemes.

Example

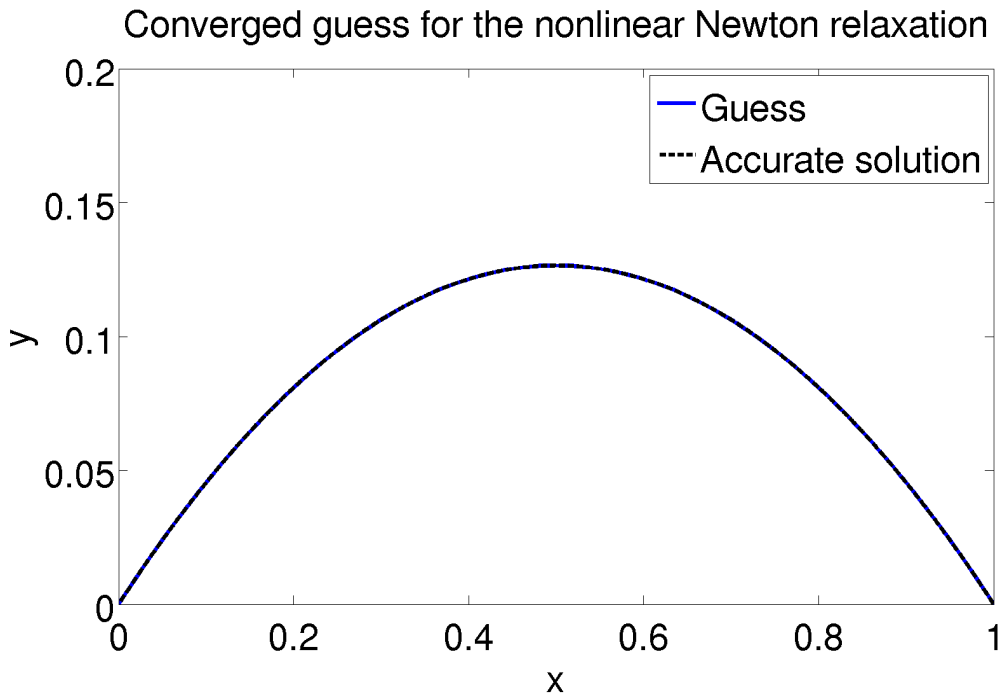
In the problem

$$y'' = -\frac{1}{1+y^2},$$
$$y(0) = 0,$$
$$y(1) = 0,$$
$$x \in [0,1]$$

use Newton relaxation. Initial guess trivial.

First iteration already reasonable.

After 4 iterations solution has settled down.



Summary

- First shoot, then relax.
- Finite difference methods are typically less accurate and efficient, and for nonlinear problems are more complex to implement. However, they are much more likely to work in complex cases.
- For linear problems finite difference methods convert the ODE to a linear system $T\boldsymbol{y} = \boldsymbol{F}$.
  - The matrix will be tri-diagonal (when using centred differencing).
  - The boundary conditions are normally directly encoded in the known vector $\boldsymbol{b}$ (but more complex boundary conditions may require modification of A as well).
- For nonlinear problems we construct a sequence of guesses $\boldsymbol{y}^{(n)}$ that converge to the solution; the previous guess is used to solve a linear system.

Chapter 15 - Collocation methods for Boundary Value Problems

MATH3018/6141, Semester 1

Boundary Value Problems

Considering simple boundary value problem

$$y'' = f(x, y, y'), \quad y(a) = A, \; y(b) = B, \quad x \in [a, b].$$

Boundary conditions are only examples here.

Have considered the shooting method (accurate, efficient, may not work) and the finite difference (relaxation) method (not particularly accurate or efficient, nearly always works). Both methods require a *grid* with n points. Accuracy of method depends on $h \propto n^{-1}$.

Can instead use a function basis, independently of a grid.

Function basis expansion

Aim to solve problem

$$\mathcal{L}y = f,$$

where $\mathcal{L}$ is a differential operator.

Function basis methods *assume* the solution $y(x)$ can be written

$$y(x) = \sum_j c_j u_j(x);$$

constants c_j are *basis coefficients*, $u_j(x)$ are *basis functions*.

Choose "simple" basis functions u_j. Get *approximate* solution by truncating series:

$$y(x) = \sum_j^n c_j u_j(x).$$

Additional conditions give linear system for unknowns (c_j), defining solution everywhere.

Collocation methods

Simplest function basis method uses a grid. *Collocation* method: fix c_j by insisting that BVP satisfied exactly at fixed set of points.

1. Assume approximate solution has form (n fixed)

$$\sum_{j}^{n} c_j u_j(x).$$

2. Boundary conditions $\implies$ two coefficients (e.g. $c_{0,1}$).
3. Evaluate $\mathcal{L}y$ at collocation points $\{x_j\}$ $\implies$ linear system.
4. Solve the linear system to give the basis coefficients c_j.

Example

We consider the problem (with exact solution $\exp(x)$)

$$y'' - y = 0, \quad y(0) = 1, \; y(1) = e.$$

Choose basis $u_j = x^j \implies y = \sum c_j x^j$. Boundary conditions:

$$c_0 = 1, \quad c_1 = e - c_0 - \sum_{j=2}^{n} c_j.$$

Differential operator $\mathcal{L}y = y'' - y$ implies:

$$\sum_{j=2}^{n} c_j \left(j(j-1)x^{j-2} \right) - \sum_{k=0}^{n} c_k x^k = 0 \quad \text{At collocation points } x = x_l.$$

Rewrite as

$$\sum_{j=2}^{n} c_j \left(j(j-1)x_l^{j-2} - x_l^j \right) - c_0 - c_1 x_l = 0$$

Example: 2

Substitute

$$c_1 = e - c_0 - \sum_{k=2}^{n} c_k \quad \text{into} \quad \sum_{j=2}^{n} c_j \left(j(j-1)x_l^{j-2} - x_l^j \right) - c_0 - c_1 x_l = 0$$

to get

$$\sum_{j=2}^{n} \left(j(j-1)x_l^{j-2} - x_l^j + x_l \right) c_j = c_0 + (e - c_0)x_l = 1 + (e-1)x_l$$

In order to determine the $n-1$ variables $c_2, c_3, \cdots, c_n$ need $n-1$ collocation points $x_1, x_2, \cdots x_{n-1}$. Then get linear system of form $A\boldsymbol{c} = \boldsymbol{b}$ ($\{x_l\}$ collocation points) where A is $n-1 \times n-1$ matrix

$$A_{lj} = \left(j(j-1)x_l^{j-2} \right) - x_l^j + x_l, \quad j = 2, 3, \cdots n \quad l = 1, 2, \cdots n-1$$

Vector $\boldsymbol{b}$ from boundary conditions.

Example: 3

When using only three polynomial basis functions we have

$$c_0 = 1, \quad c_1 = e - 1 - c_2.$$

We then evaluate the system at the collocation point $x_1 = 1/2$ to get

$$\begin{aligned} A_{12}c_2 &= 1 + (e-1)\frac{1}{2} = \frac{1}{2}(1+e), \\ A_{12} &= 2 \times 1 \times \left(\frac{1}{2}\right)^0 - \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right) = \frac{9}{4}, \\ c_2 &= \tfrac{2}{9}(1+e). \end{aligned}$$

This, combined with boundary conditions, gives approximate solution

$$y = 1 + \tfrac{1}{9}\left((7e-11)x + 2(1+e)x^2\right).$$

Example: 3

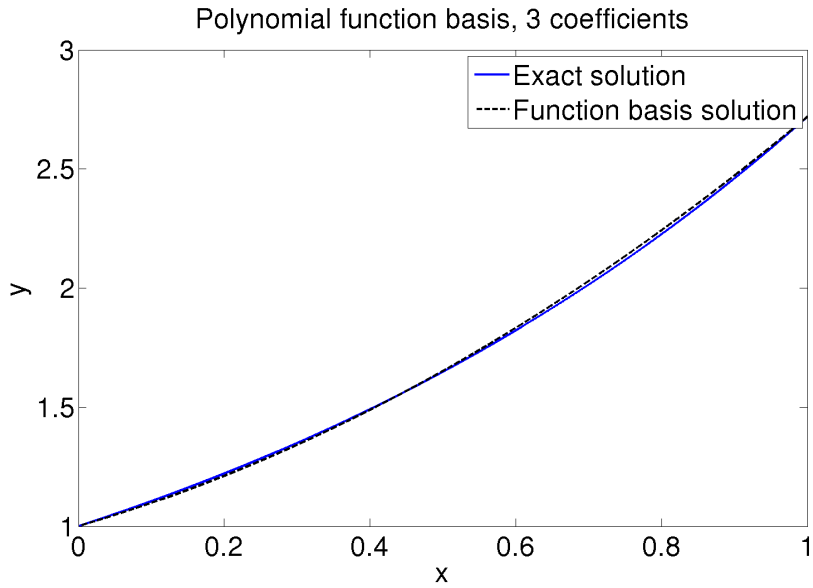
With only a few points the method is very accurate.

Use Chebyshev collocation points

$$x_k = \frac{1}{2}\left(1 + \cos\left(\frac{(k-1)\pi}{n-1}\right)\right)$$

to check convergence with n.

Convergence is *exponentially* fast.



Example: 3

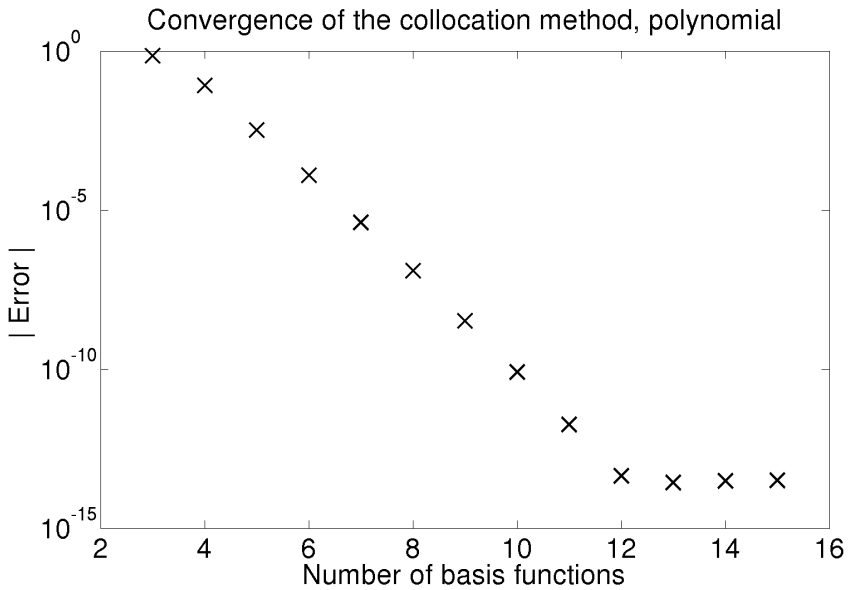
With only a few points the method is very accurate.

Use Chebyshev collocation points

$$x_k = \frac{1}{2}\left(1 + \cos\left(\frac{(k-1)\pi}{n-1}\right)\right)$$

to check convergence with n.

Convergence is *exponentially* fast.



Summary

- Collocation methods are popular; given the right choice of basis and collocation points the convergence can be faster than any polynomial, giving floating point accuracy with a few dozen basis functions at most.

Chapter 16 - Partial Differential Equations and Elliptic Equations

MATH3018/6141, Semester 1

Partial differential equations

Partial differential equations (PDEs) are differential equations for functions with more than one independent variable, say $u(x, y)$ or $y(t, x)$.

Have more complex behaviour than ODEs and describe a wider variety of physics.

Numerical methods for partial differential equations is an area of active research. There are many interesting problems that current methods are incapable of solving with current computing resources.

Will cover some basic, well-understood methods for PDEs.

2nd Order Linear PDEs

Consider only 2nd order linear problems of the form

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial y} = f(x, y),$$

where $u = u(x, y)$ and A, B, C, D, E are constants. Sometimes use the "evolutionary" notation $y = y(t, x)$, where derivatives will be with respect to t and x.

As with ODEs, unique solution are only found when the correct number and type of *boundary conditions* are given.

The type of boundary conditions required depends on the type of the problem being solved.

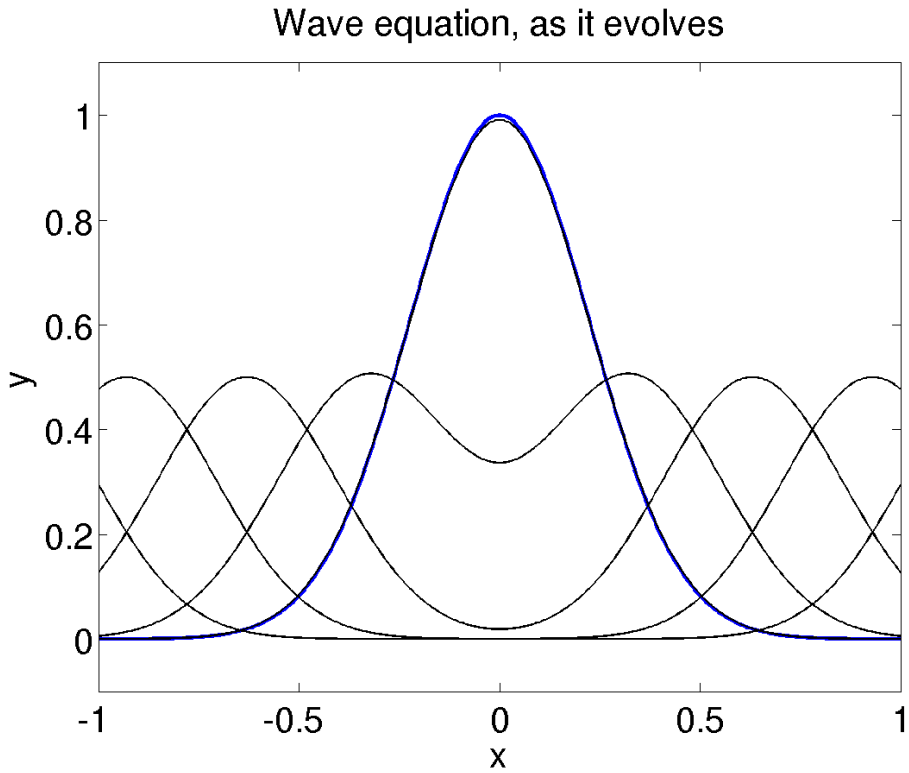
For evolution problems ($y = y(t, x)$), boundary conditions specified at the initial time $t = 0$ are called *initial conditions*.

Hyperbolic equations

Hyperbolic equations: associated with wave propagation, central to hydro- and electro- dynamics. The prototype is the wave equation

$$\partial_{tt} y - c^2 \partial_{xx} y = 0.$$

Information is propagated at *finite speed c*.

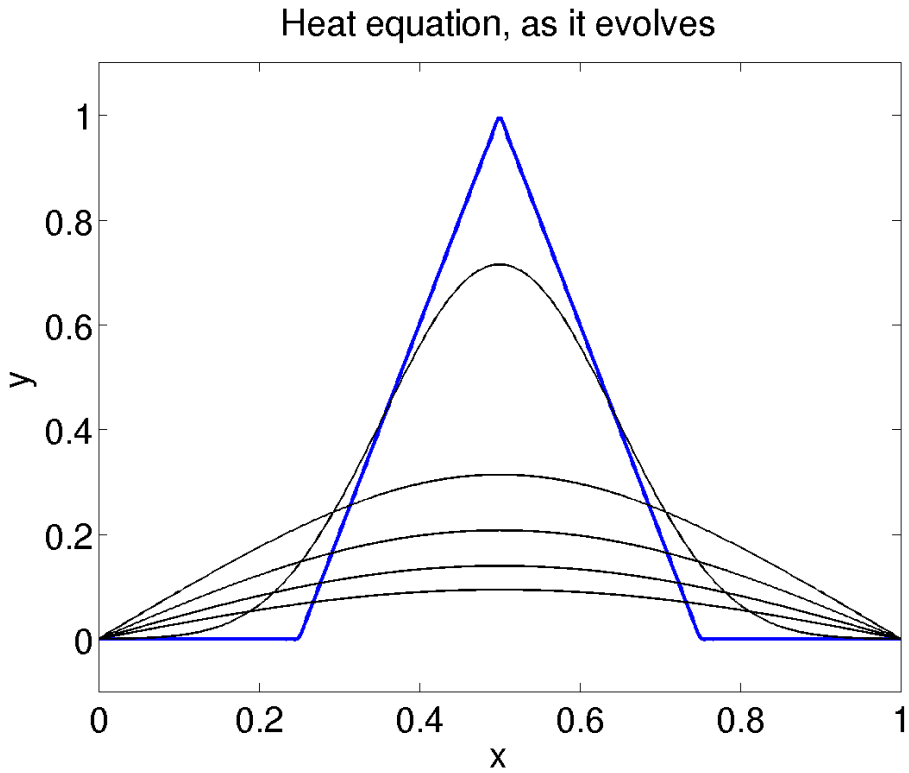


Parabolic equations

Parabolic equations associated with diffusion problems. The prototype is the heat equation

$$\partial_t y - k\partial_{xx} y = 0.$$

Information is propagated at "*infinite speed*" and smooths the solution.

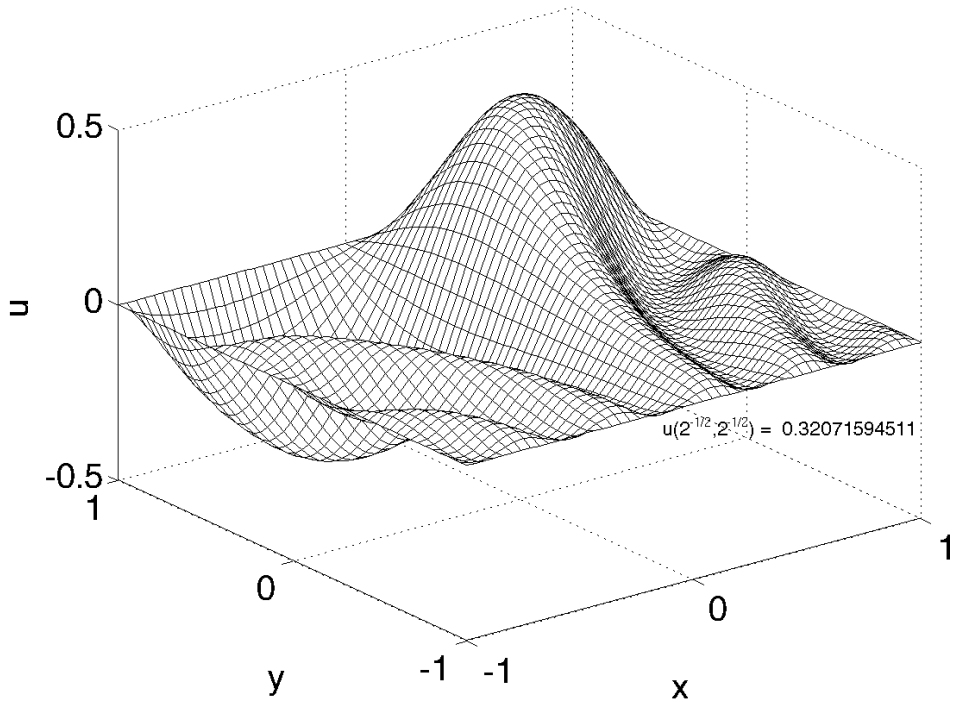


Elliptic equations

Elliptic equations associated with static or stationary problems. The prototype is Laplace's equation

$$\partial_{xx} u + \partial_{yy} u = 0.$$

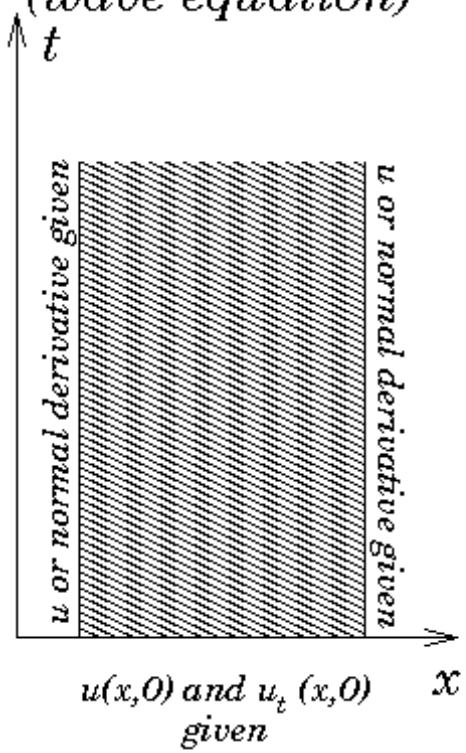
Generalized boundary value problems (in a sense).



Boundary conditions

Hyperbolic

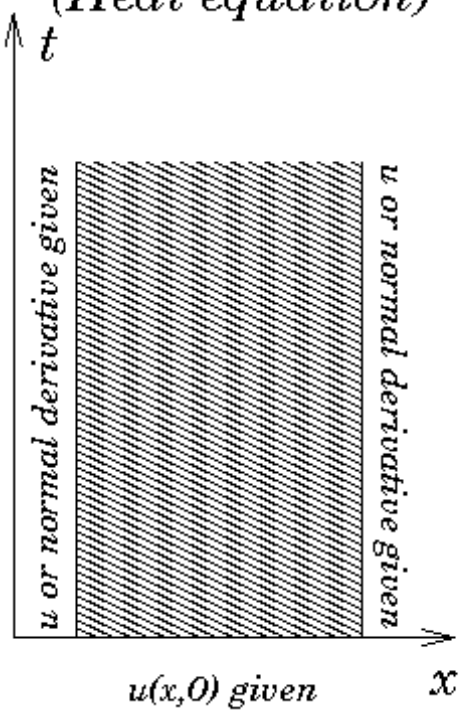
(wave equation)



$$u_{tt} - c^2 u_{xx} = 0$$

Parabolic

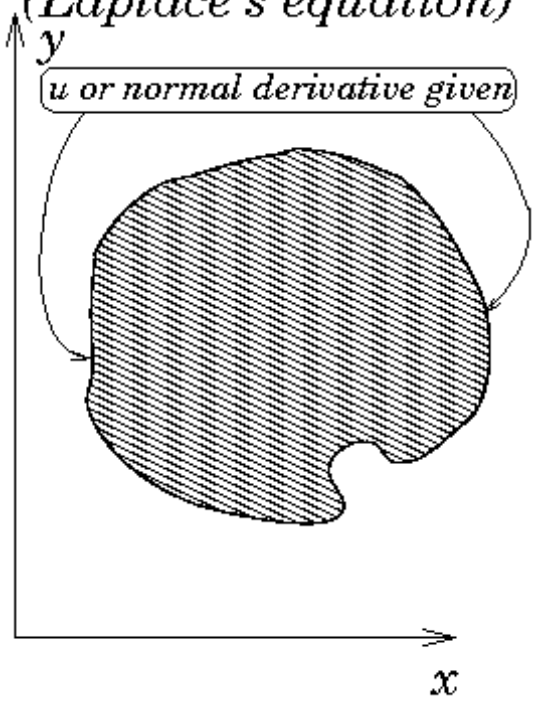
(Heat equation)



$$u_t - k\, u_{xx} = 0$$

Elliptic

(Laplace's equation)



$$u_{xx} + u_{yy} = 0$$

Finite difference methods for elliptic equations

Take Poisson's equation with Dirichlet boundary conditions

$$\begin{cases} u_{xx} + u_{yy} = f(x,y), & \text{in } \Omega, \\ u\big|_{\partial\Omega} = \phi(x,y), & \text{on } \partial\Omega \end{cases}.$$

Simple approach: replace all derivatives with finite difference equivalents.

Directly generalizes BVPs. Algorithm:

1. Introduce grid covering Ω.
2. Convert PDE to linear system for unknown interior points.
3. Solve the linear system.

As in BVP case, boundary data is known through ϕ.

The grid

Assume that Ω is a rectangle:

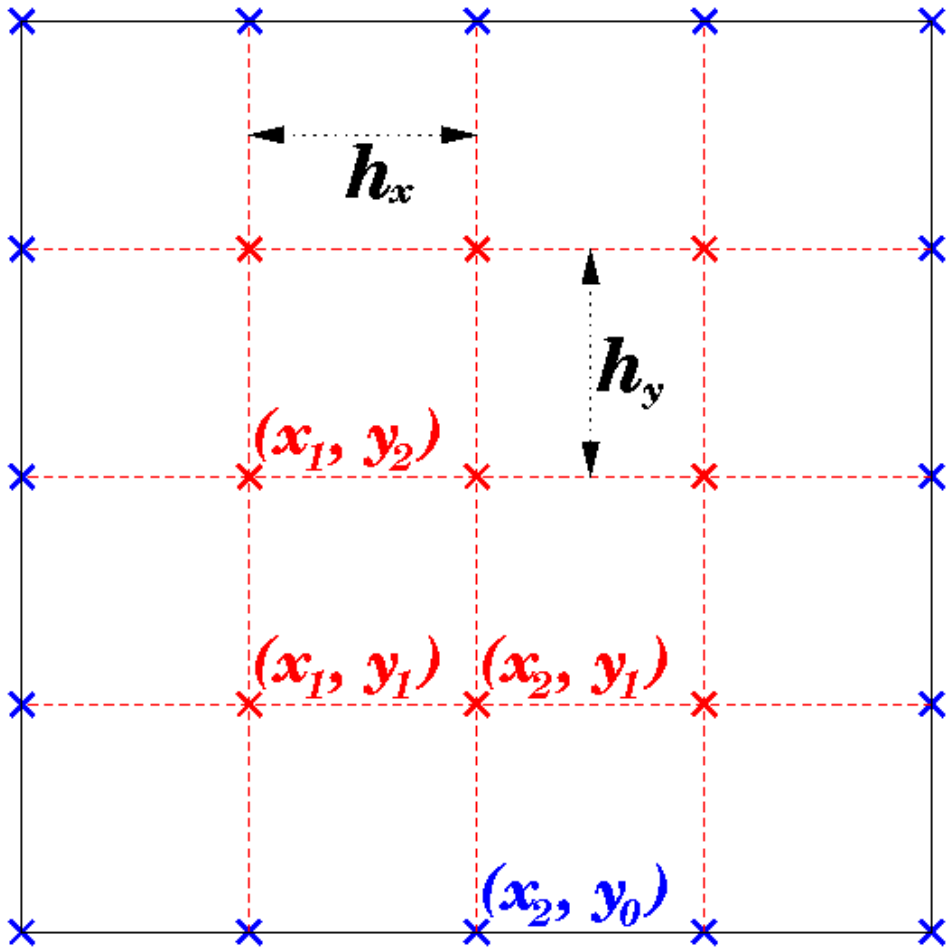
$$x \in [0, a], \quad y \in [0, b].$$

Use $n+2$ points in x, $m+2$ in y:

$$h_x = \frac{a}{n+1},$$
$$h_y = \frac{b}{m+1}.$$

Location of (i, j) grid point is

$$\begin{pmatrix} x_i \\ y_j \end{pmatrix} = \begin{pmatrix} ih_x \\ jh_y \end{pmatrix}.$$



Finite difference formula

Apply central differencing to PDE

$$u_{xx} + u_{yy} = f$$

for the *interior* points $0 < i < n+1$, $0 < j < m+1$. Result:

$$u_{xx} = \frac{u_{i-1,j} + u_{i+1,j} - 2u_{i,j}}{h_x^2} + \mathcal{O}(h_x^2),$$
$$u_{yy} = \frac{u_{i,j-1} + u_{i,j+1} - 2u_{i,j}}{h_y^2} + \mathcal{O}(h_y^2).$$

Rearranging and defining $\alpha = (h_x/h_y)^2$ gives

$$\left[u_{i-1,j} + u_{i+1,j} - 2u_{i,j}\right] + \alpha\left[u_{i,j-1} + u_{i,j+1} - 2u_{i,j}\right] = h_x^2 f_{i,j}$$

System of $n \times m$ linear equations for $n \times m$ unknowns, as boundary data ($u_{0,j}$, $u_{n+1,j}$ etc.) is known.

Ordering

Want to form a linear system

$$A\boldsymbol{u} = \boldsymbol{F}.$$

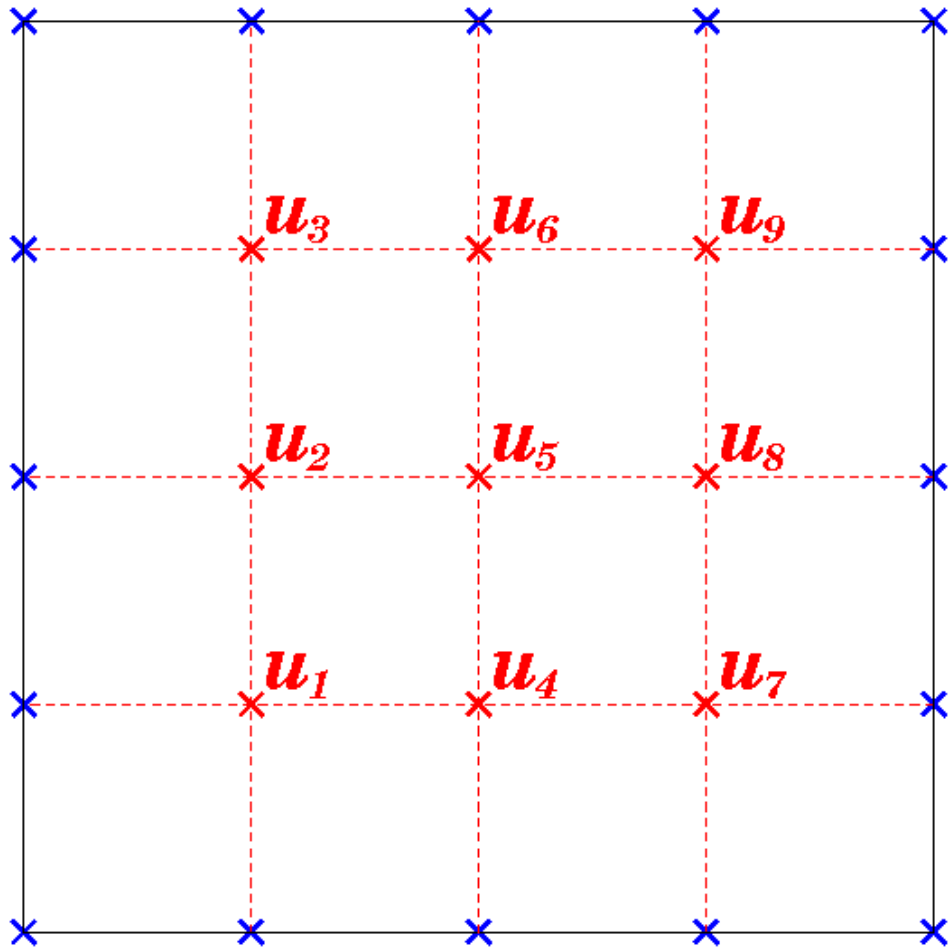
But $u_{i,j}$ form a matrix, so re-order entries as vector of unknowns $\boldsymbol{u}$.

Order by rows (as in figure):

$$u_k = u_{i,j}, \quad k = i + n(j-1).$$

OR order by *columns* :

$$u_k = u_{i,j}, \quad k = j + m(i-1).$$



The matrix

Apply natural ordering by columns to equation

$$\left[u_{i-1,j} + u_{i+1,j} - 2u_{i,j}\right] + \alpha\left[u_{i,j-1} + u_{i,j+1} - 2u_{i,j}\right] = h_x^2 f_{i,j}.$$

For 2×3 example in the notes have

$$\boldsymbol{u} = [u_{1,1}, u_{1,2}, u_{1,3}, u_{2,1}, u_{2,2}, u_{2,3}]^T$$

and the matrix A given by

$$\begin{pmatrix}
-2(1+\alpha) & \alpha & 0 & 1 & 0 & 0 \\
\alpha & -2(1+\alpha) & \alpha & 0 & 1 & 0 \\
0 & \alpha & -2(1+\alpha) & 0 & 0 & 1 \\
1 & 0 & 0 & -2(1+\alpha) & \alpha & 0 \\
0 & 1 & 0 & \alpha & -2(1+\alpha) & \alpha \\
0 & 0 & 1 & 0 & \alpha & -2(1+\alpha)
\end{pmatrix}.$$

The matrix is *sparse* but not tridiagonal.

Boundary terms and the known vector

In constructing A, ignored known elements such as $u_{0,j}$, (will be moved to RHS). Construct $\boldsymbol{F}$ for 2×3 example with equation

$$\left[u_{i-1,j} + u_{i+1,j} - 2u_{i,j}\right] + \alpha\left[u_{i,j-1} + u_{i,j+1} - 2u_{i,j}\right] = h_x^2 f_{i,j}$$

and ordering

$$\boldsymbol{u} = \left[u_{1,1}, u_{1,2}, u_{1,3}, u_{2,1}, u_{2,2}, u_{2,3}\right]^T$$

to find

$$\boldsymbol{F} = \begin{pmatrix} h_x^2 f_{1,1} - u_{0,1} - \alpha u_{1,0} \\ h_x^2 f_{1,2} - u_{0,2} \\ h_x^2 f_{1,3} - u_{0,2} - \alpha u_{1,4} \\ h_x^2 f_{2,1} - u_{3,1} - \alpha u_{2,0} \\ h_x^2 f_{2,2} - u_{3,2} \\ h_x^2 f_{2,3} - u_{3,3} - \alpha u_{2,4} \end{pmatrix}$$

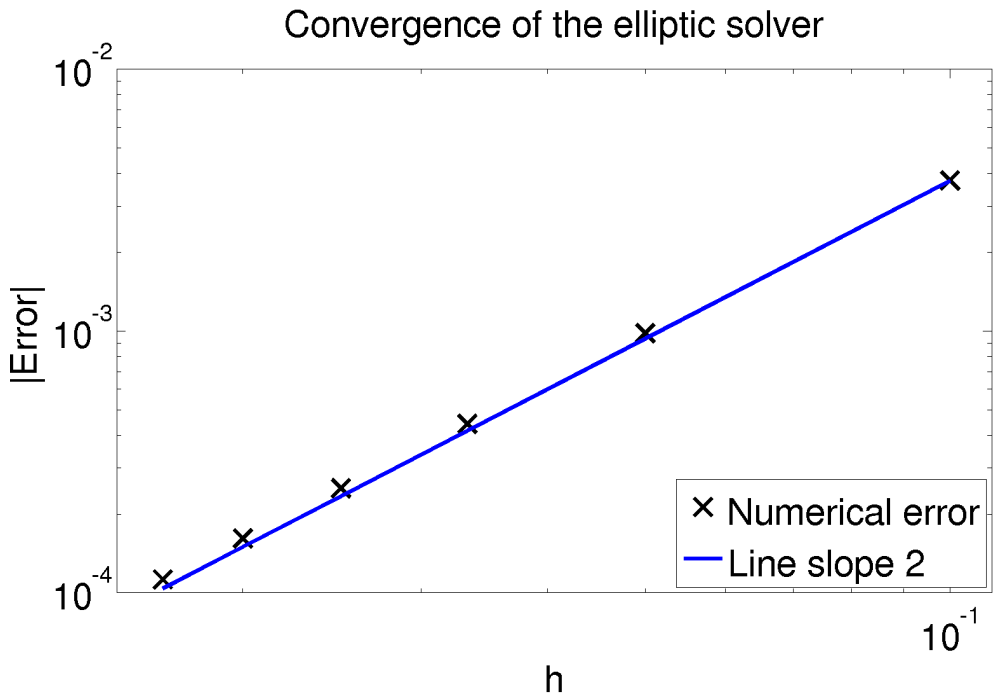
Example

Solve

$$u_{xx}+u_{yy} = -2\pi^2 \sin(\pi x)\sin(\pi y)$$

on $(x, y) \in [0, 1]^2$ with trivial boundary conditions.

Increasing resolution improves the accuracy steadily and we can see second order convergence.



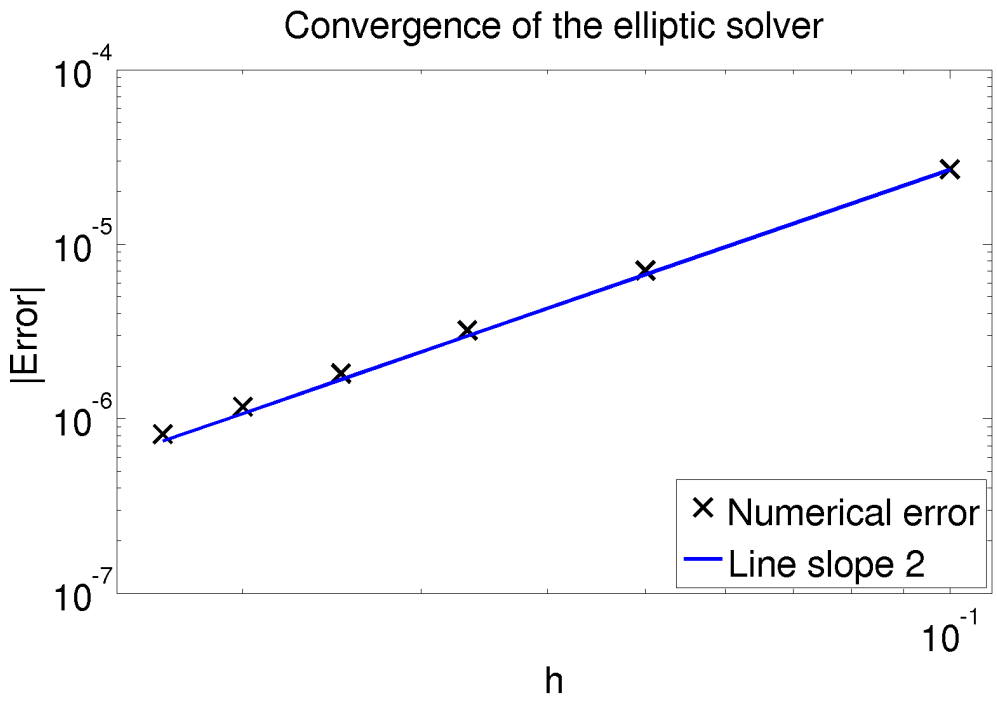
Example: 2

The equation

$$
\begin{aligned}
u_{xx} + u_{yy} &= xe^y, \quad (x, y) \in [0, 1]^2, \\
u &= 0 \qquad \text{on } x = 0, \\
u &= e^y \qquad \text{on } x = 1, \\
u &= x \qquad \text{on } y = 0, \\
u &= xe \qquad \text{on } y = 1,
\end{aligned}
$$

is also straightforwardly solved.

Increasing resolution improves the accuracy steadily and we can still see second order convergence.



Iterative methods for the linear system

Have constructed a second order convergent method. But *direct* matrix solver takes $\mathcal{O}(N^3)$ operations. Thus if we solve on an $n \times n$ grid we are looking at $\mathcal{O}(n^6)$ operations! Extending to three dimensions on an $n \times n \times n$ grid we will require $\mathcal{O}(n^9)$ operations.

Instead note that the finite difference equations

$$\left[u_{i-1,j} + u_{i+1,j} - 2u_{i,j}\right] + \alpha \left[u_{i,j-1} + u_{i,j+1} - 2u_{i,j}\right] = h_x^2 f_{i,j}$$

in good form for an *iterative* method. These often work well for sparse matrices, as here. In particular matrix diagonally (although not striclty) dominant so might expect Gauss-Seidel to work.

Gauss-Seidel or variants on the SOR methods are frequently used for solving elliptic equations in this fashion.

In practice never assemble the matrix A or the vector $\boldsymbol{F}$ for linear equation $A\boldsymbol{u} = \boldsymbol{F}$ but apply Gauss-Seidel/SOR directly to the difference scheme.

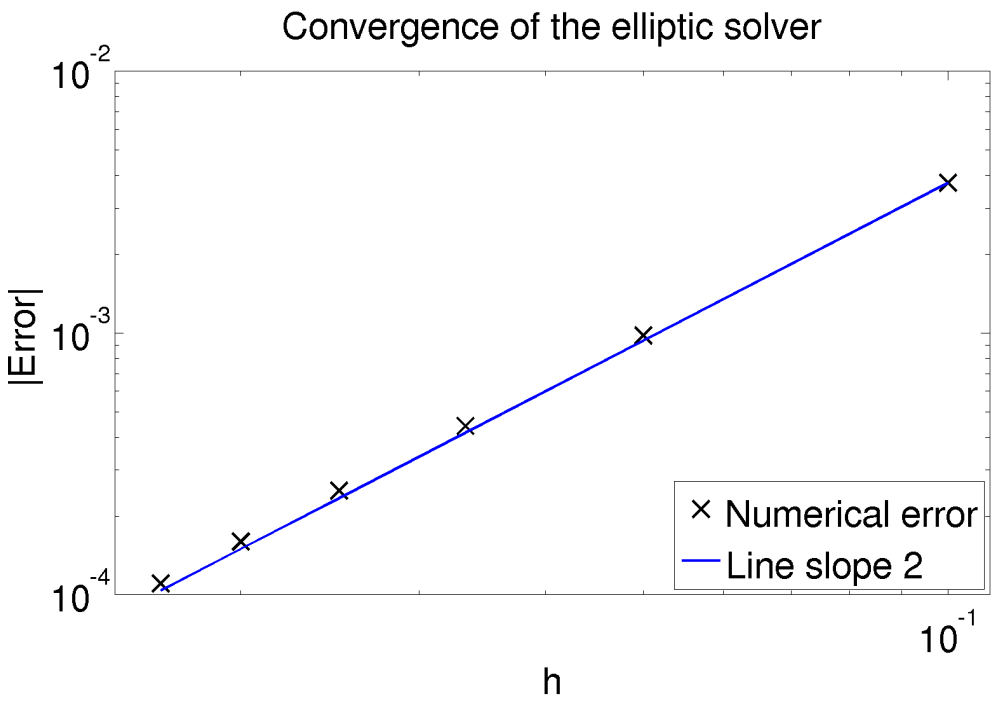
Example

The equation

$$u_{xx}+u_{yy} = -2\pi^2 \sin(\pi x)\sin(\pi y)$$

on $(x, y) \in [0, 1]^2$ with trivial boundary conditions is straightforwardly solved.

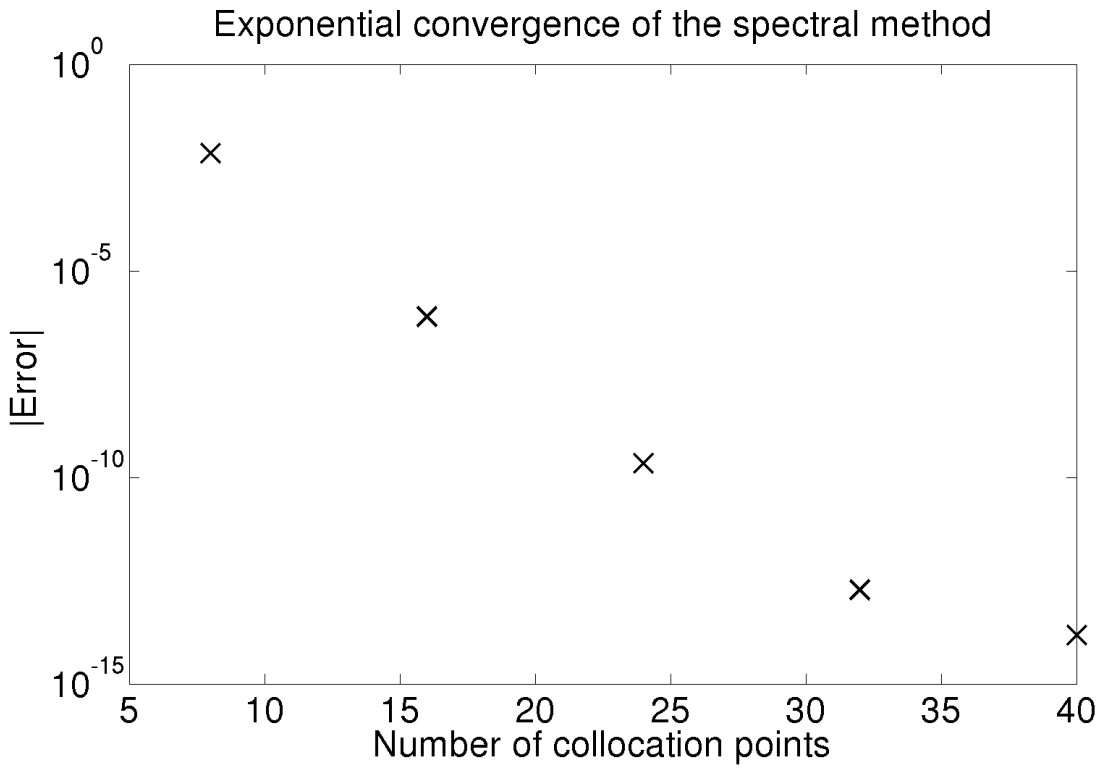
Increasing resolution improves the accuracy steadily and we can see second order convergence.



Collocation methods

Collocation methods extend to elliptic equations. Work well on simple domains with smooth boundary conditions.

Convergence is exceptionally fast; when these methods work, finite differences cannot compete.



Summary

- Partial differential equations have (at least) two independent variables, either denoted (x, y) for boundary value (elliptic) problems or (t, x) for evolutionary (hyperbolic, parabolic) problems.
- Boundary conditions are required to specify a unique solution. In evolutionary problems the conditions at $t = 0$ are called initial conditions.
- Elliptic equations are in some sense generalizations of boundary value problems.
- Using an evenly spaced grid and central differencing on linear elliptic PDE leads to a linear set of finite difference equations.
- By ordering the nodes can rewrite as a linear matrix equation of the form $A\boldsymbol{x} = \boldsymbol{b}$.
- The matrix A is large and sparse and so the equation $A\boldsymbol{x} = \boldsymbol{b}$ is best solved with iterative methods.
- Collocation methods, when they work, are much more efficient.

Chapter 17 - Basic Methods for Parabolic PDEs

MATH3018/6141, Semester 1

Partial differential equations

Partial differential equations (PDEs) are differential equations involving dependent variables that are functions of more than one independent variable, *e.g.* $u(x, y)$ or $y(t, x)$.

Have more complex behaviour than ODEs and describe a wider variety of physics.

Only look at linear problems.

Also only consider finite difference methods: simple to analyse but not always best.

Here consider simple methods for linear second order parabolic equations.

Finite Difference Methods

Want to solve the prototype 2nd order linear parabolic PDE

$$\partial_t y - k\partial_{xx} y = 0.$$

Adopt following strategy:

1. Introduce grid (equally spaced for simplicity) covering domain. Bounded in space ($x \in [0, 1]$ in following), semi-infinite in time ($t \geq 0$ in following).
2. Replace PDE with finite differences. Only works for interior points (in space).
3. Rearrange interior difference equations to get either
   - *Explicit* method: Can evaluate solution at the next time step (at any space point) explicitly from the solution at previous time step and the boundary data. OR
   - *Implicit* method: Must solve a linear system to evaluate the solution at the next timestep.

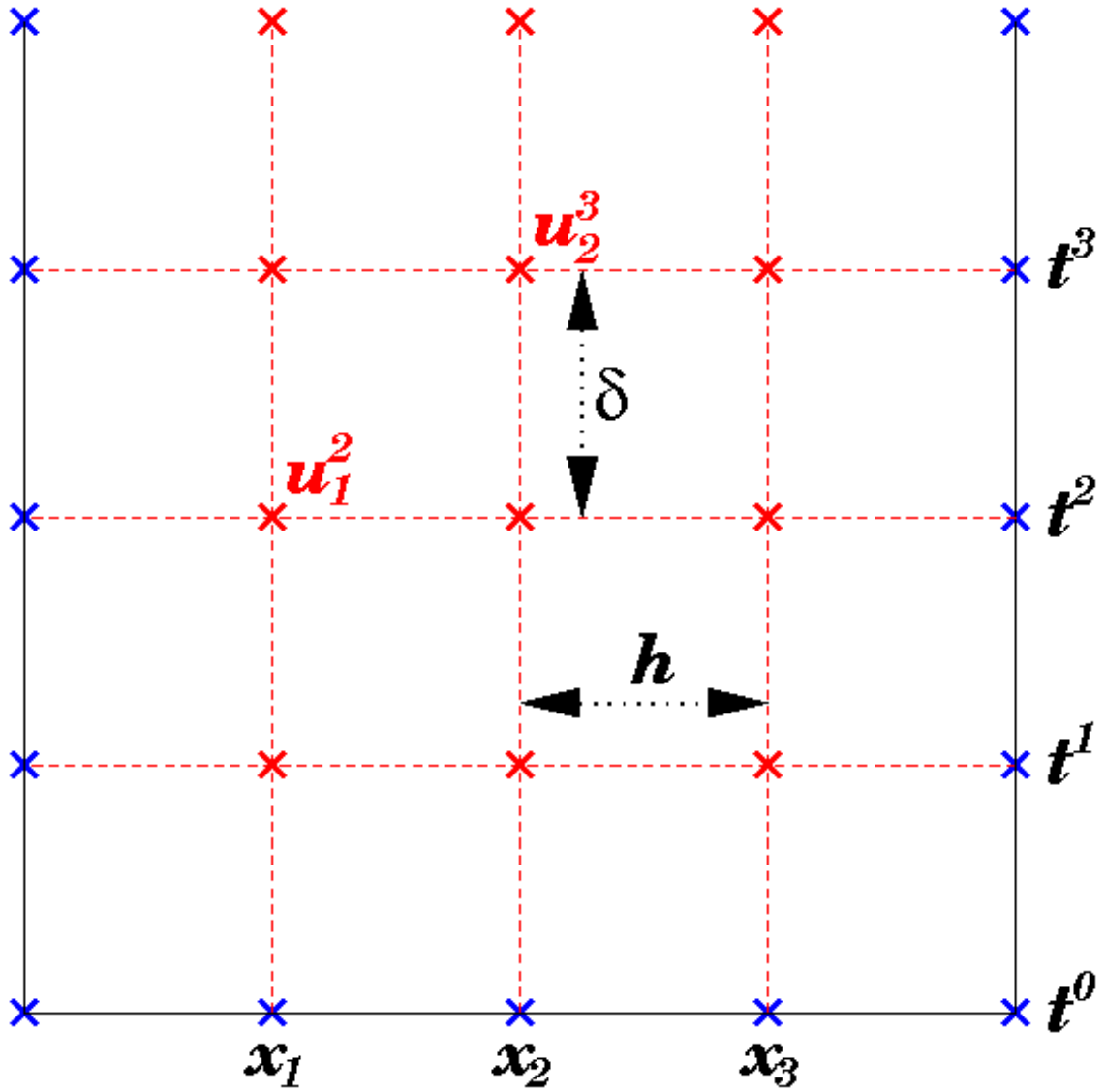
The Grid

Introduce grid. Initial conditions fix known data at $t = t^0$. Use initial data and

boundary conditions to calculate solution at next time step $t = t^1$. Repeat

process calculating solution at t^2 from boundary conditions and solution calculated in previous step at $t = t^1$ March forward in

time by repeating the process as many times as neccessary.



The Grid

Note: write discrete grid points

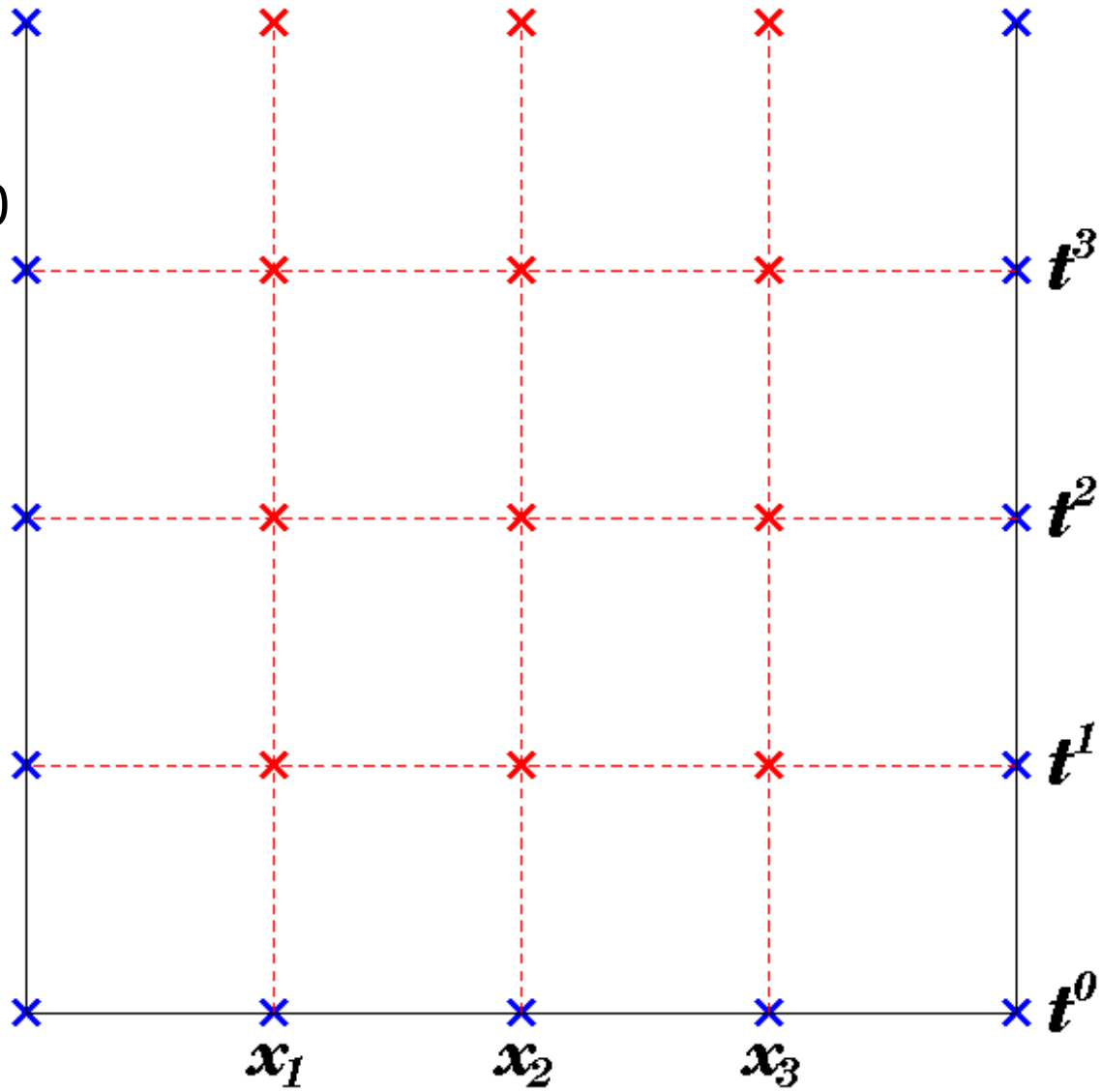
$$(x_i, t^n), \quad 0 \leq i \leq N+1, \; n \geq 0$$

where i, n are integer grid indices. t^2 means "the second time step", not "t squared".

From spatial boundaries get grid spacing

$$h = 1/(N+1).$$

However timestep, δ, can be set freely.



The Grid

Note: write discrete grid points

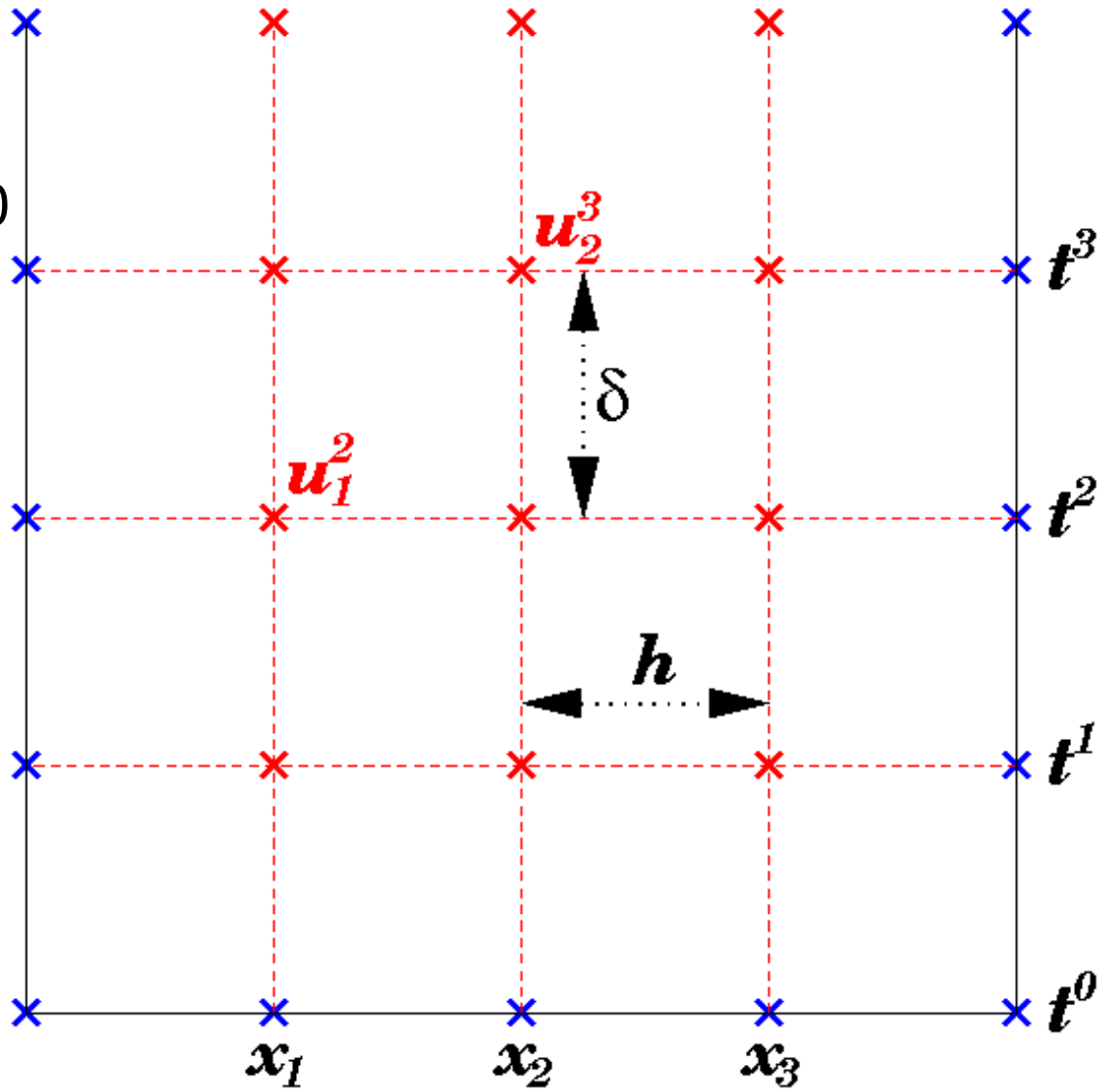
$$(x_i, t^n), \quad 0 \leq i \leq N+1, \; n \geq 0$$

where i, n are integer grid indices. t^2 means "the second time step", not "t squared".

From spatial boundaries get grid spacing

$$h = 1/(N+1).$$

However timestep, δ, can be set freely.



FTCS

Look at heat equation with $k = 1$,

$$\partial_t y - \partial_{xx} y = 0.$$

Given initial conditions

$$y(x, 0) = g(x)$$

and boundary conditions

$$y(0, t) = 0, \quad y(1, t) = 0.$$

Introduce grid (x_i, t^n).

Initial conditions imply that at $t = t^0 = 0$ have $y_i^0 = g(x_i)$.

Convert heat equation to difference equation. Standard approach is to use central differencing:

$$\partial_{xx} y|_{x=x_i} = \frac{y_{i+1} + y_{i-1} - 2y_i}{h^2} + \mathcal{O}(h^2).$$

FTCS

$$\partial_t y - \partial_{xx} y = 0. \quad y(x,0) = g(x),\ y(0,t) = 0,\ y(1,t) = 0.$$

$$\partial_{xx} y|_{x=x_i} = \frac{y_{i+1} + y_{i-1} - 2y_i}{h^2} + \mathcal{O}(h^2).$$

Central differencing in *time* difficult [only one time slice (t^0) known]; instead use *forward* differencing

$$\partial_t y|_{t=t^n} = \frac{y^{n+1} - y^n}{\delta} + \mathcal{O}(1).$$

Substituting in the difference formulas and rearranging gives

$$y_i^{n+1} = y_i^n + \frac{\delta}{h^2}\left(y_{i+1}^n + y_{i-1}^n - 2y_i^n\right);$$

$$s = \delta/h^2 \implies y_i^{n+1} = (1-2s)y_i^n + s\left(y_{i+1}^n + y_{i-1}^n\right).$$

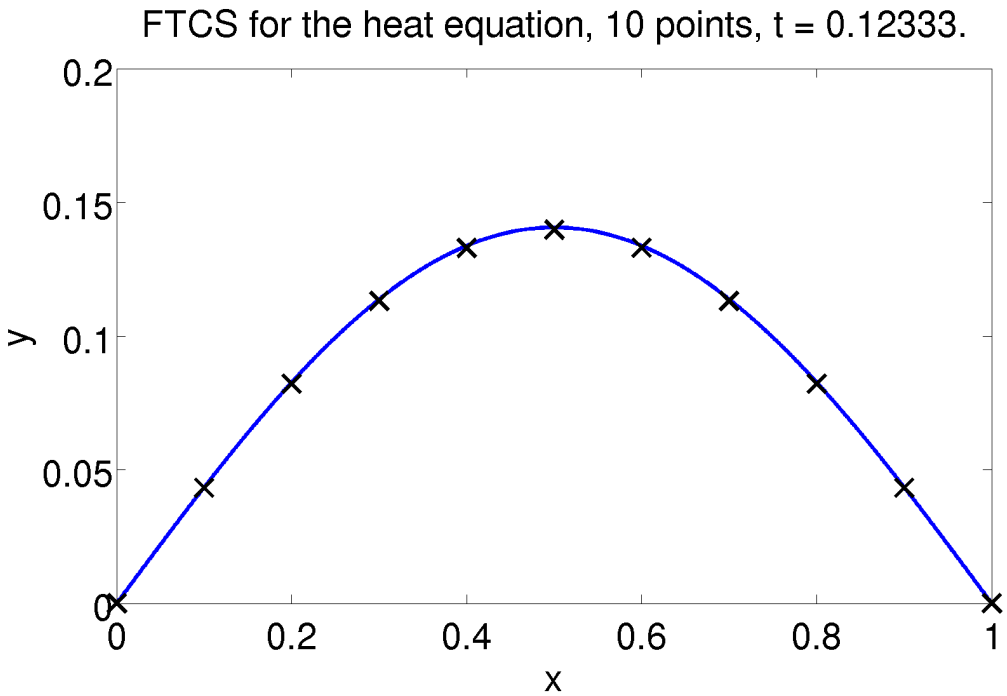
Explicit algorithm called FTCS (Forward Time, Centred Space).

Example

Apply FTCS to heat equation. Initial data is "tent" function. Choose timestep such that $s = 1/3$.

Result evolves in roughly the right fashion. Increasing resolution improves accuracy as we evolve.

Convergence check: second order convergence.

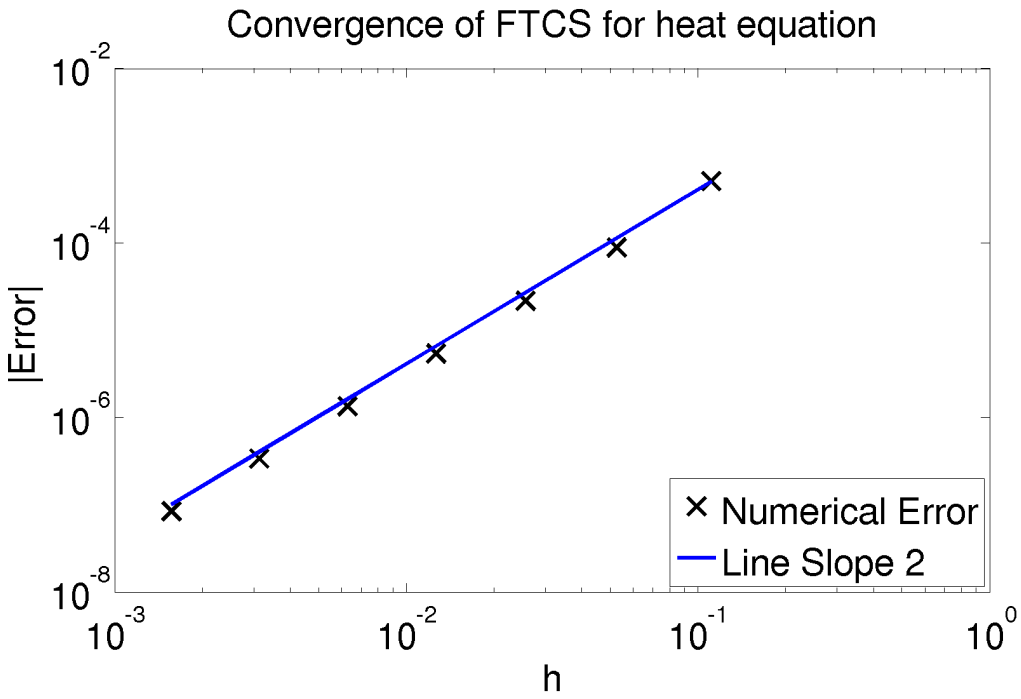


Example

Apply FTCS to heat equation. Initial data is "tent" function. Choose timestep such that $s = 1/3$.

Result evolves in roughly the right fashion. Increasing resolution improves accuracy as we evolve.

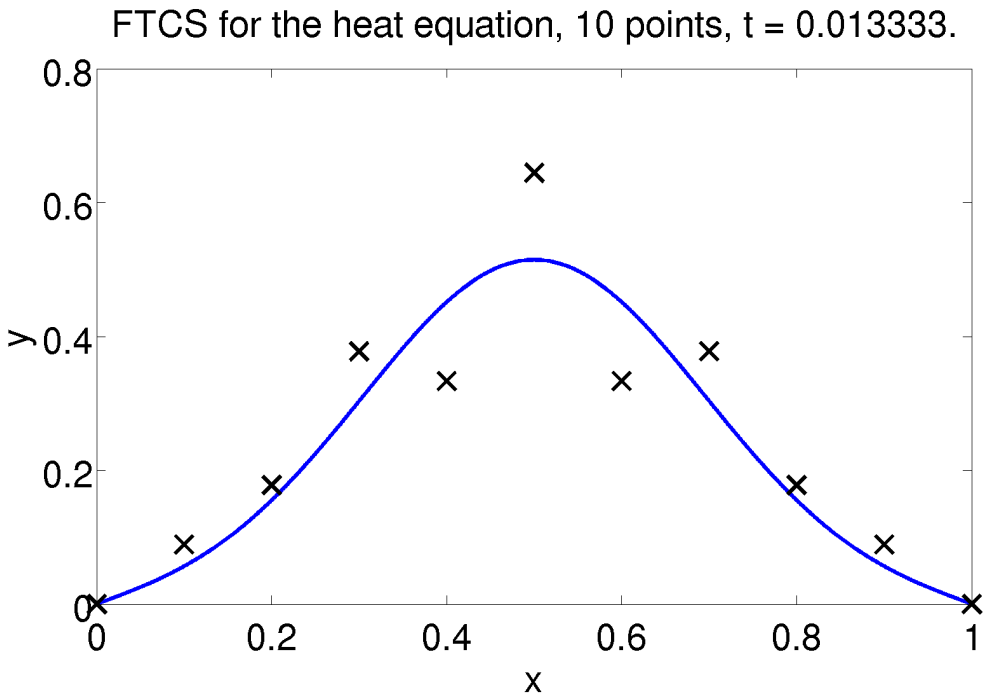
Convergence check: second order convergence.



Example: 2

Fixing $s = \delta/h^2 = 1/3$ is a severe restriction on time step. Double $n \implies h$ reduced by half $\implies \delta$ reduced by a quarter.

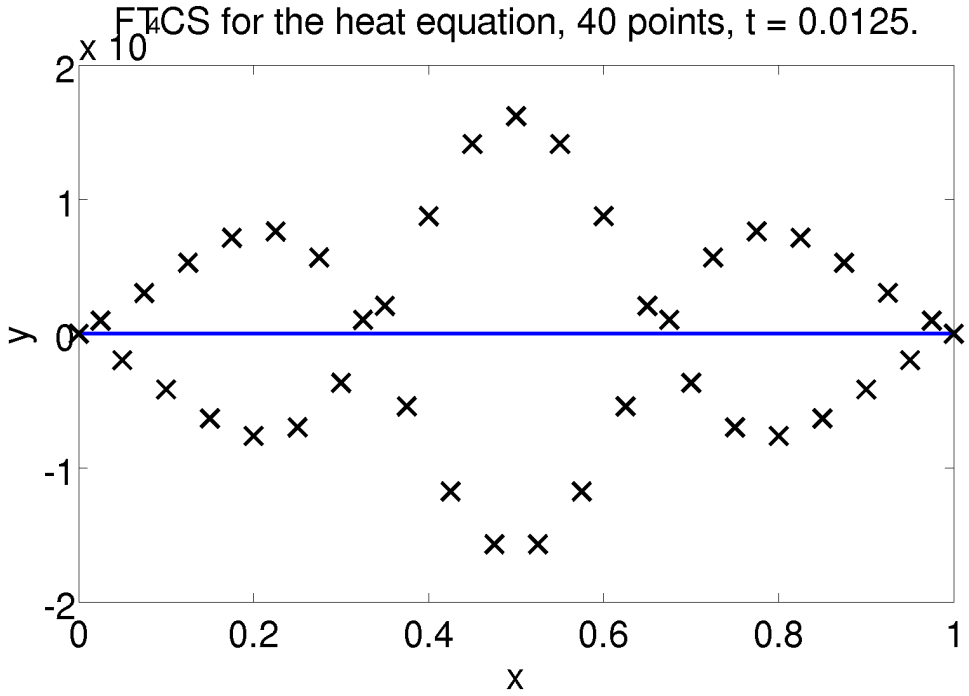
Instead increase timestep setting $s = 2/3$. Less restrictive, but evolution not as good. In fact, it seems to be rubbish.



Example: 2

Increasing resolution helped before; now just make things worse.

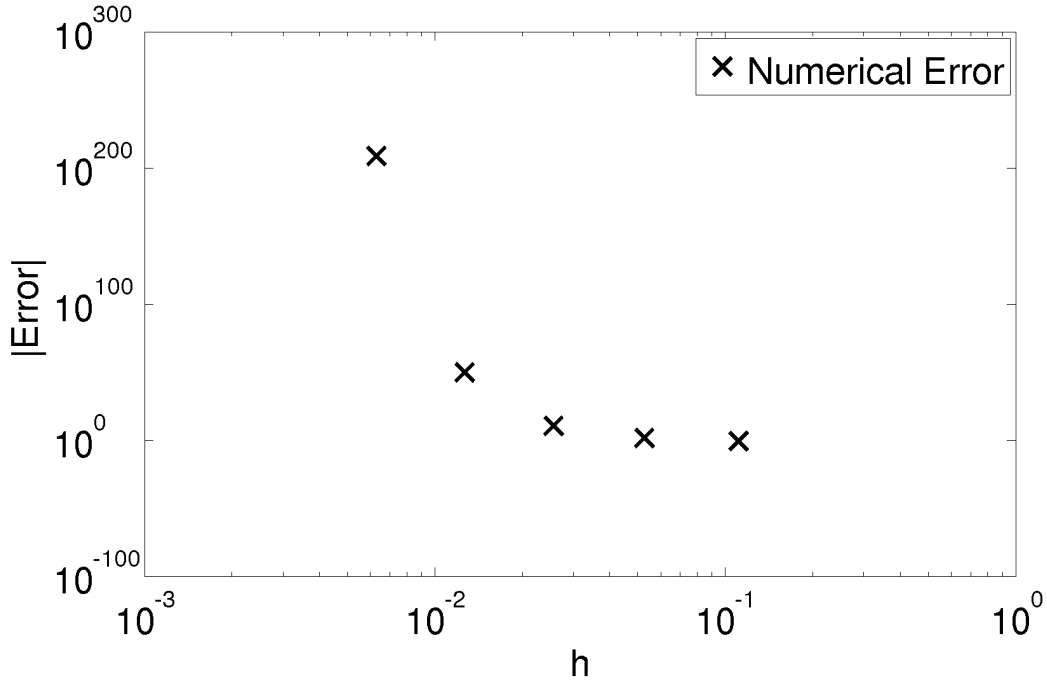
No convergence at all!



Example: 2

Increasing resolution helped before; now just make things worse.

No convergence at all!



BTCS

FTCS has problems where $s > 1/2$: a limitation. Instead modify finite differencing to improve behaviour.

Still have the problem

$$\partial_t y - \partial_{xx} y = 0. \quad y(x,0) = g(x),\ y(0,t) = 0,\ y(1,t) = 0$$

and central differencing in space

$$\partial_{xx} y|_{x=x_i} = \frac{y_{i+1} + y_{i-1} - 2y_i}{h^2} + \mathcal{O}(h^2).$$

Try *backward* differencing in time,

$$\partial_t y|_{t=t^{n+1}} = \frac{y^{n+1} - y^n}{\delta} + \mathcal{O}(1).$$

BTCS

$$\partial_t y - \partial_{xx} y = 0. \quad y(x,0) = g(x),\ y(0,t) = 0,\ y(1,t) = 0.$$

$$\partial_{xx} y|_{x=x_i} = \frac{y_{i+1} + y_{i-1} - 2y_i}{h^2} + \mathcal{O}(h^2).$$

$$\partial_t y|_{t=t^{n+1}} = \frac{y^{n+1} - y^n}{\delta} + \mathcal{O}(1).$$

Substituting in the difference formulas and rearranging gives

$$y_i^{n+1} = y_i^n + \frac{\delta}{h^2}\left(y_{i+1}^{n+1} + y_{i-1}^{n+1} - 2y_i^{n+1}\right);$$

$$s = \delta/h^2 \implies (1+2s)y_i^{n+1} = y_i^n + s\left(y_{i+1}^{n+1} + y_{i-1}^{n+1}\right).$$

This *implicit* algorithm is called BTCS (Backward Time, Centred Space). It requires solving a linear system.

BTCS - linear system

BTCS can be written

$$(1+2s)y_i^{n+1} - s\left(y_{i+1}^{n+1} + y_{i-1}^{n+1}\right) = y_i^n.$$

See directly that we have the linear system

$$A\boldsymbol{y}^{n+1} = \boldsymbol{y}^n + \boldsymbol{F}$$

where the matrix is, for example

$$\begin{pmatrix} 1+2s & -s & 0 & 0 & \dots \\ -s & 1+2s & -s & 0 & \dots \\ 0 & -s & 1+2s & -s & \dots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

and the known vector is (y_0, y_{N+1} from boundary conditions)

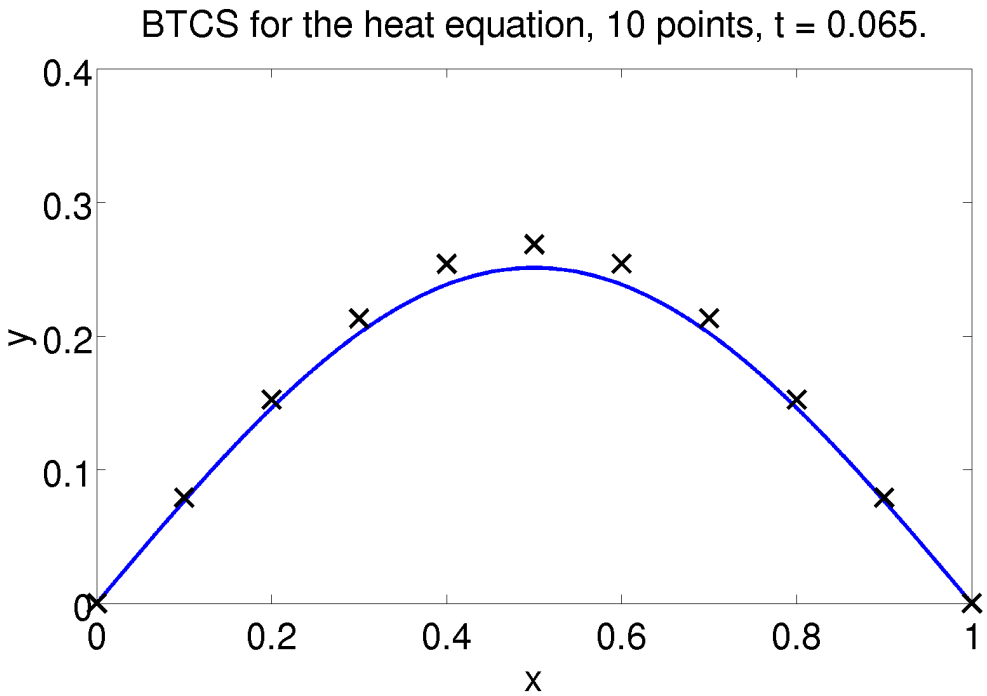
$$\boldsymbol{F} = \begin{pmatrix} y_0^n + sy_0^{n+1} & y_1^n & \dots & y_2^n & y_{N+1}^n + sy_{N+1}^{n+1} \end{pmatrix}^T.$$

Example

Apply BTCS to heat equation. Initial data is "tent" function. Choose timestep such that $s = 1/2$.

Result evolves in roughly the right fashion. Increasing resolution improves accuracy as we evolve.

Convergence check: second order convergence.

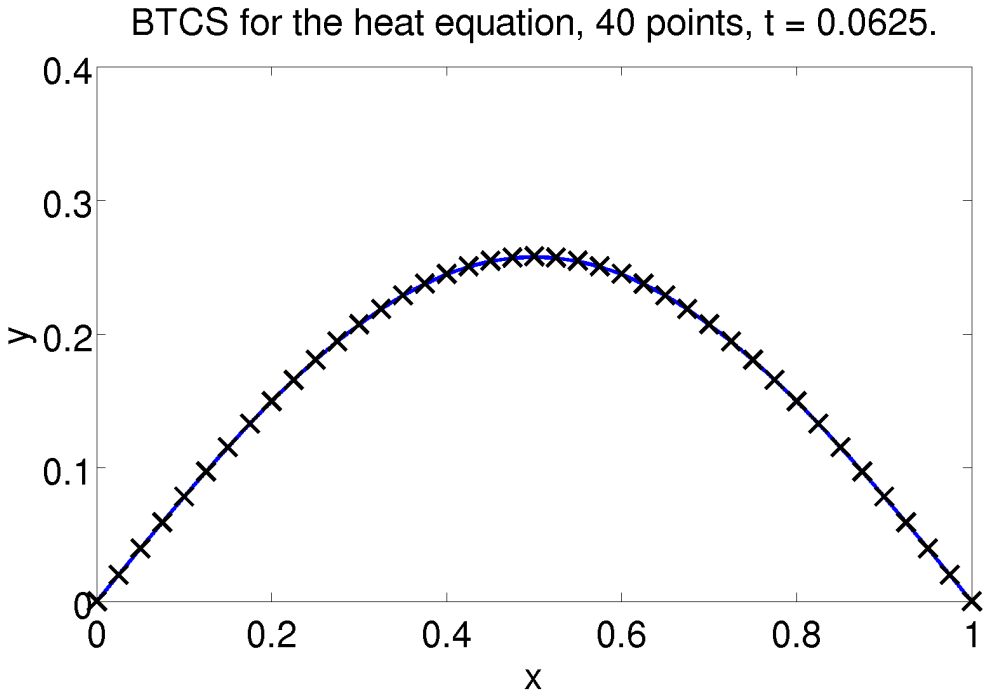


Example

Apply BTCS to heat equation. Initial data is "tent" function. Choose timestep such that $s = 1/2$.

Result evolves in roughly the right fashion. Increasing resolution improves accuracy as we evolve.

Convergence check: second order convergence.

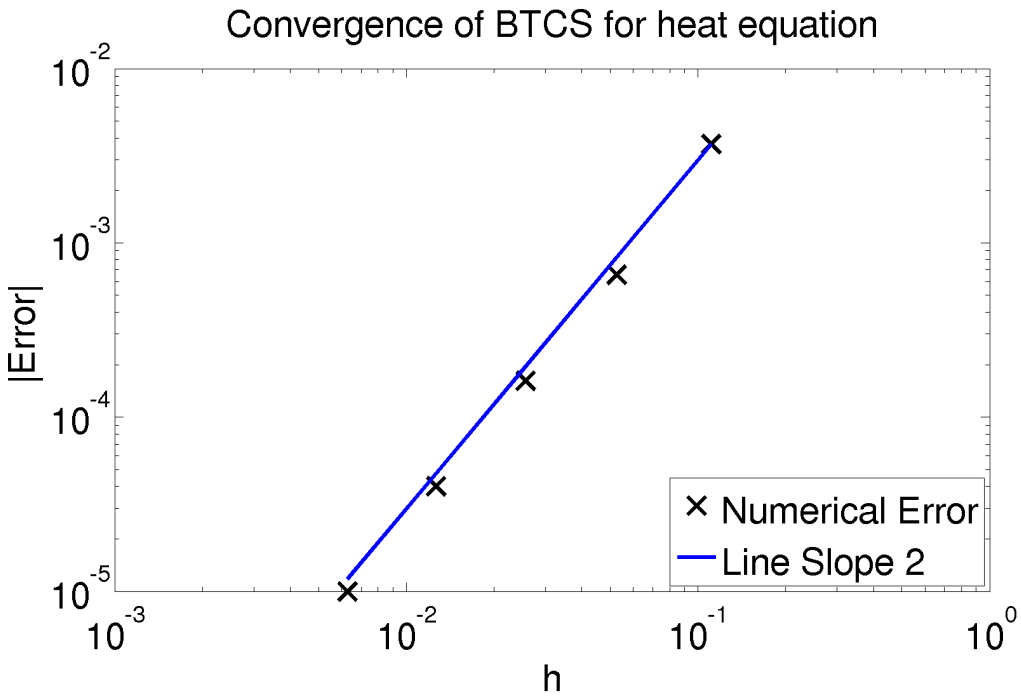


Example

Apply BTCS to heat equation. Initial data is "tent" function. Choose timestep such that $s = 1/2$.

Result evolves in roughly the right fashion. Increasing resolution improves accuracy as we evolve.

Convergence check: second order convergence.



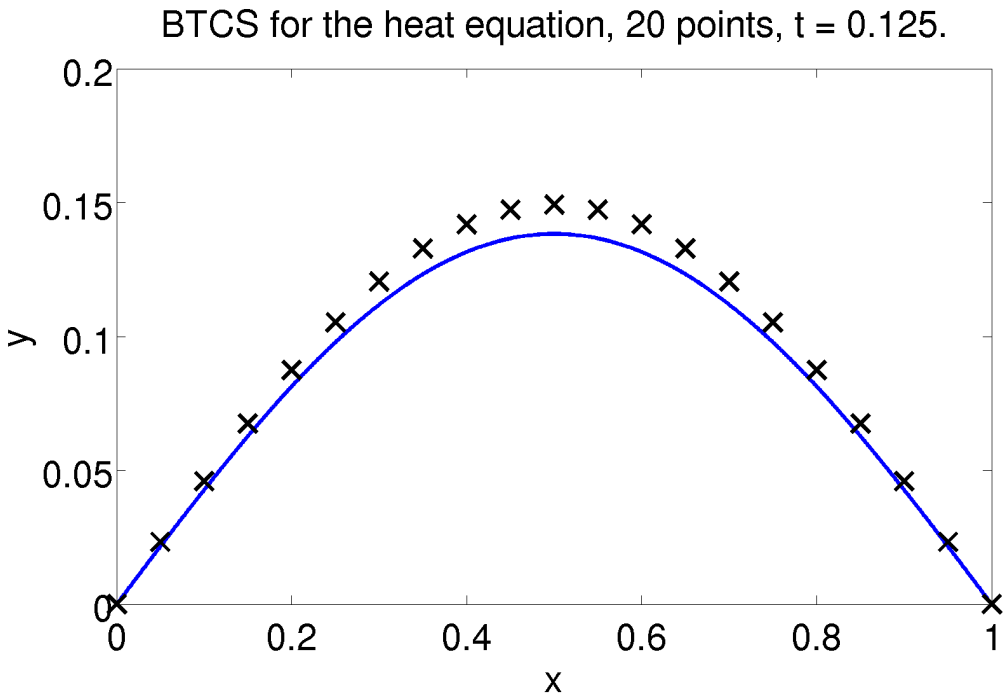
Example: 2

Fixing $s = \delta/h^2 = 1/2$ is a severe restriction on the time step. Double $n \implies h$ reduced by half $\implies \delta$ reduced by a quarter.

Instead increase s to 5: less restrictive and evolution appears reasonable.

Increasing resolution helps here.

Still have second order convergence.



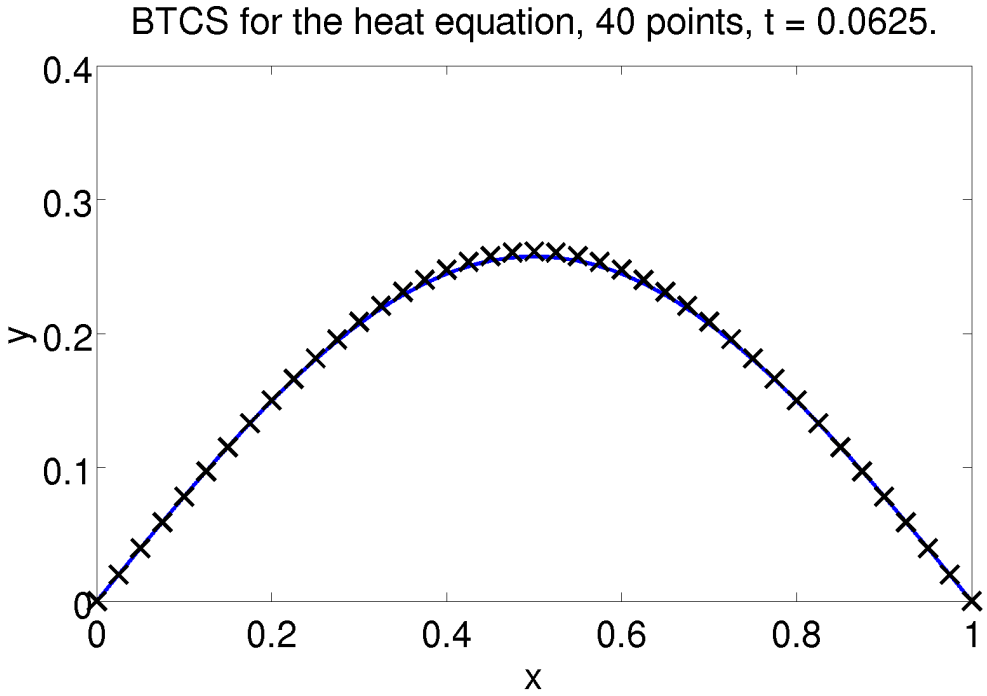
Example: 2

Fixing $s = \delta/h^2 = 1/2$ is a severe restriction on the time step. Double $n \implies h$ reduced by half $\implies \delta$ reduced by a quarter.

Instead increase s to 5: less restrictive and evolution appears reasonable.

Increasing resolution helps here.

Still have second order convergence.



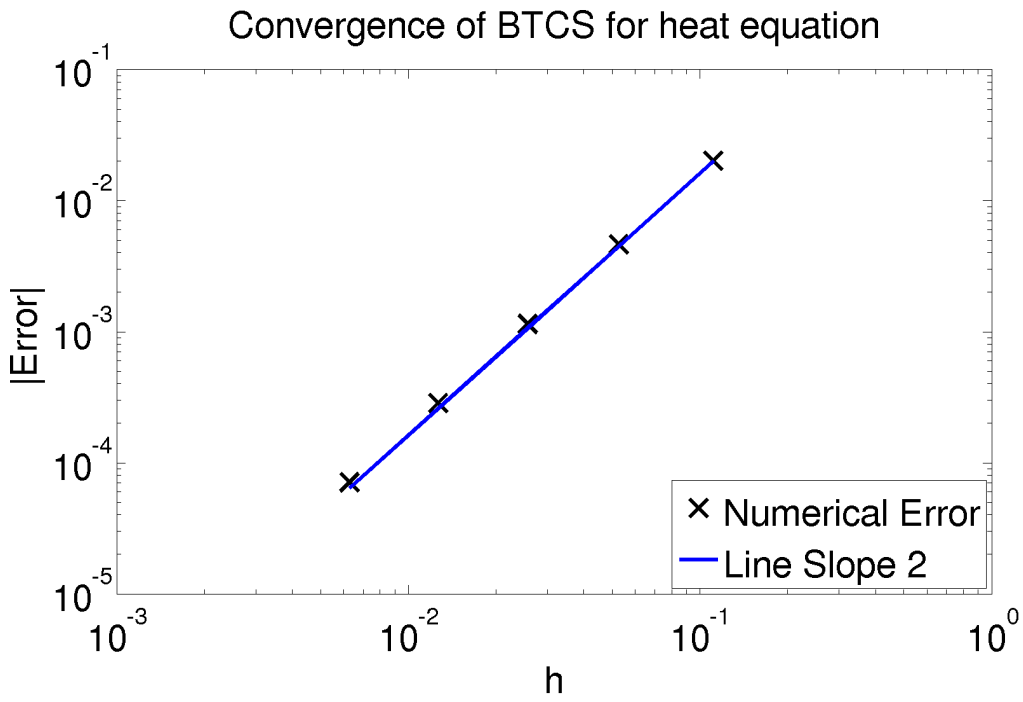
Example: 2

Fixing $s = \delta/h^2 = 1/2$ is a severe restriction on the time step. Double $n \implies h$ reduced by half $\implies \delta$ reduced by a quarter.

Instead increase s to 5: less restrictive and evolution appears reasonable.

Increasing resolution helps here.

Still have second order convergence.



Summary

- Finite difference methods result from the standard algorithm:
  1. Introduce a grid
  2. Replacing derivatives with finite differences
  3. Rearrange to get either
     - ★ an explicit update formula, or
     - ★ an implicit equation which can be converted to a linear system.
- For the heat equation the simple FTCS method is stable if $s < 1/2$, whereas the implicit BTCS method is good for all s.
- For the advection equation FTCS is unstable for all c, whilst FTBS is stable if $c < 1$.
- Von Neumann stability analysis in the next lecture will indicate the reasons for the results above.

Chapter 18- Von Neumann stability

MATH3018/6141, Semester 1

Partial differential equations

Partial differential equations (PDEs) are differential equations involving dependent variables that are functions of more than one independent variable, *e.g.* $u(x, y)$ or $y(t, x)$.

Have more complex behaviour than ODEs and describe a wider variety of physics.

Only look at linear problems.

Also only consider finite difference methods: simple to analyse but not always best.

Previously looked at simple finite difference algorithms applied to heat and advection equations.

At present have no explanation for which algortihm worked and which did not; for that we need stability theory.

The results

Algorithm	Heat equation
FTCS	Stable $s < 1/2$
BTCS	Stable $\forall s$
FTBS	??

Here $s = k\delta/h^2$ for the heat equation

$$\partial_t y - k\partial_{xx} y = 0.$$

Consistency

Finite difference equation is *consistent* if it tends to the PDE as the grid sizes go to zero *independently*.

Equivalent to requiring the local error tot vanish as $h, \delta \to 0$ in any way.

All the methods defined previously are clearly consistent.

Stability

A finite difference algortithm is *stable* if errors in the solution decay in time. If the errors grow in time it is *unstable*.

Have seen that BTCS is stable for the heat equation, whilst FTCS is stable only when $s < 1/2$.

For heat equation FTCS is *conditionally* stable, whilst BTCS is *unconditionally* stable.

For unstable algorithms spurious solutions grow from small errors and swamp the genuine solution.

This idea is closely related to the concept of well-posedness for PDEs.

Convergence

Key concept is *convergence*.

A finite difference algortihm is convergent if in the limit of zero grid spacing it converges to the exact solution of the PDE.

Exceptionally difficult to directly prove in general. Instead rely on

Lax's Theorem (1954): Given a properly posed linear initial-value problem and a finite difference approximation to it that is consistent, stability is necessary and sufficient for convergence.

Can be extended (with some caveats) to nonlinear problems with non-trivial boundary conditions, even when discontinuities form.

Determining stability

Von Neumann stability method for linear algorithms and equations:

1. Make ansatz that there are solutions of the finite difference equation of the form

$$y_\ell^{(k)} = e^{i\alpha\ell h} q^k, \qquad (i = \sqrt{-1}, \alpha \in \mathbb{R}, q \in \mathbb{C}).$$

2. Substitute this solution ansatz into the finite difference equation.
3. Determine q as a function of α, the grid spacings h, δ, and known numbers (such as the diffusion coefficient k).
4. Determine under what conditions we have stability, i.e. when

$$|q| < 1 \quad \forall \alpha.$$

Can apply method blindly to produce stability result; but how does it come about?

Von Neumann stability

The finite difference equations for a linear PDE have the form

$$\mathcal{L}_1 \boldsymbol{y}^{(k+1)} - \mathcal{L}_2 \boldsymbol{y}^{(k)} = \boldsymbol{f}.$$

where $\mathcal{L}_1$ and $\mathcal{L}_2$ are linear difference operators and

$$\boldsymbol{y}^{(k)} = (y_1^{(k)}, y_2^{(k)}, \cdots, y_{n-1}^{(k)})^T$$

is the solution vector at the k'th time step.
Suppose $\boldsymbol{y}^{(k)}$ represents the real solution and $\boldsymbol{z}^{(k)}$ represents the solution containing the error.
But $\boldsymbol{z}^{(k)}$ will satisfy the same difference equations

$$\mathcal{L}_1 \boldsymbol{z}^{(k+1)} - \mathcal{L}_2 \boldsymbol{z}^{(k)} = \boldsymbol{f}.$$

Write the error as $\boldsymbol{\mathcal{E}}^{(k)} = \boldsymbol{z}^{(k)} - \boldsymbol{y}^{(k)}$. By linearity this satisfies

$$\mathcal{L}_1 \boldsymbol{\mathcal{E}}^{(k+1)} - \mathcal{L}_2 \boldsymbol{\mathcal{E}}^{(k)} = \boldsymbol{0}.$$

Does this error $\boldsymbol{\mathcal{E}}^{(k)}$ grow?

Linear stability

Difference equation for error

$$\mathcal{L}_1 \boldsymbol{\mathcal{E}}^{(k+1)} - \mathcal{L}_2 \boldsymbol{\mathcal{E}}^{(k)} = \mathbf{0}.$$

Look for a solution

$$\boldsymbol{\mathcal{E}}^{(k)} = (e^{i\alpha h} q^k, e^{i\alpha 2h}, \cdots, e^{i\alpha \ell h} q^k, \cdots, e^{i\alpha (n-1)h} q^k)^T$$

i.e. on the ℓ'th grid point look for

$$\mathcal{E}_\ell^{(k)} = e^{i\alpha \ell h} q^k$$

Subsitute ansatz into difference equations and determine q as a function of the wavenumber α.

$$\text{If} \quad |q| < 1 \quad \forall \alpha \quad \text{difference equations are stable}$$

and the error decays. Otherwise they are unstable.

FTCS for the heat equation

Our finite difference algorithm is

$$y_\ell^{(k+1)} = (1-2s)y_\ell^{(k)} + s\left(y_{\ell+1}^{(k)} + y_{\ell-1}^{(k)}\right).$$

From the preceding analysis the same difference is satisfied by the error $\mathcal{E}_\ell^{(k)}$. Look for

$$\mathcal{E}_\ell^{(k)} = e^{i\alpha\ell h}q^k.$$

$$\implies \quad e^{i\alpha\ell h}q^{k+1} = (1-2s)e^{i\alpha\ell h}q^k + s\left(e^{i\alpha(\ell+1)h}q^k + e^{i\alpha(\ell-1)h}q^k\right).$$

Removing the common factor of $e^{\alpha j\ell h}q^k$ gives

$$q = (1-2s) + s\left(e^{i\alpha h} + e^{-i\alpha h}\right).$$

Relates q to wavenumber α and grid spacing h and parameter s.

FTCS for the heat equation: 2

$$y_\ell^{(k+1)} = (1-2s)y_\ell^{(k)} + s\left(y_{\ell+1}^{(k)} + y_{\ell-1}^{(k)}\right).$$

$$q = (1-2s) + s\left(e^{i\alpha h} + e^{-i\alpha h}\right).$$

Now bound $|q|$ for *any* α (may depend on grid spacing).
More obvious what is happening if we write

$$q = 1 - 4s\sin^2\left(\frac{\alpha h}{2}\right).$$

See that, for arbitrary α, sin term bounded between 0 and 1. Therefore the limiting values for q are

$$1 - 4s \leq q \leq 1 \qquad\qquad (0 \leq \sin^2(\dots) \leq 1)$$

Note $q < -1$ only for large waveneumber α, *i.e.* short perturbations.
Stability, and thus convergence, happens only for

$$|q| < 1 \Leftrightarrow s < \tfrac{1}{2} \Leftrightarrow \frac{\delta}{h^2} < \tfrac{1}{2}.$$

BTCS for the heat equation

We look at BTCS in the form

$$(1+2s)y_\ell^{(k+1)} = y_\ell^{(k)} + s\left(y_{\ell+1}^{(k+1)} + y_{\ell-1}^{(k+1)}\right).$$

Same equations satisfied by the error $\mathcal{E}_\ell^{(k)}$.
Use ansatz $\mathcal{E}_\ell^{(k)} = \exp(\alpha j \ell h) q^k$ and remove common factors:

$$(1+2s)q = 1 + sq\left(e^{i\alpha h} + e^{-i\alpha h}\right).$$

Rewrite this as

$$\begin{aligned} q &= 1 + 2s\left(e^{i\alpha h} + e^{-i\alpha h} - 2\right)q \\ &= 1 - 4sq\sin^2\left(\tfrac{\alpha h}{2}\right) \\ \implies \qquad q &= \left(1 + 4s\sin^2\left(\tfrac{\alpha h}{2}\right)\right)^{-1}, \end{aligned}$$

which is *always* less than 1. Hence BTCS is unconditionally stable.

Summary

- Von Neumann stability analysis is the key tool for understanding the results seen “experimentally” in the last lecture.
- By looking at the growth or decay of individual frequency modes we can check the stability of the method; Lax’s theorem then proves convergence.
- For complex equations and / or algorithms, it is often very difficult or impossible to apply even Von Neumann stability analysis.
- If an algorithm does not work in the linear case then it won’t work for the equivalent nonlinear problem!

Chapter 19 - Eigenvalues and the power method

MATH3018/6141, Semester 1

Eigenvalues - revision

Eigenvalues, λ and their corresponding eigenvectors $\boldsymbol{u}$ are non-zero solutions to the linear system

$$\boldsymbol{A}\boldsymbol{u} = \lambda \boldsymbol{u}.$$

Matrix eigenvalue problems have important applications: e.g.

resonant modes of a system, or defining spectral radius $\varrho(M) = \max|\lambda(M)|$ which encodes information about the convergence of iterative schemes.

Eigenvalues and polynomials

Standard method to calculate eigenvalues: Find the n roots of the *characteristic polynomial*

$$\det(A - \lambda I) = 0.$$

Could compute roots e.g. by nonlinear root finding.

However:

1. Frequently do not need all eigenvalues, but only the largest one(s).
2. Computing and sorting all eigenvalues using this method excessively expensive ($O(n!)$ operations).
3. Need to solve large polynomial equations.
4. These are likely to be *badly conditioned*: a small change in the coefficients leads to a large change in the roots.

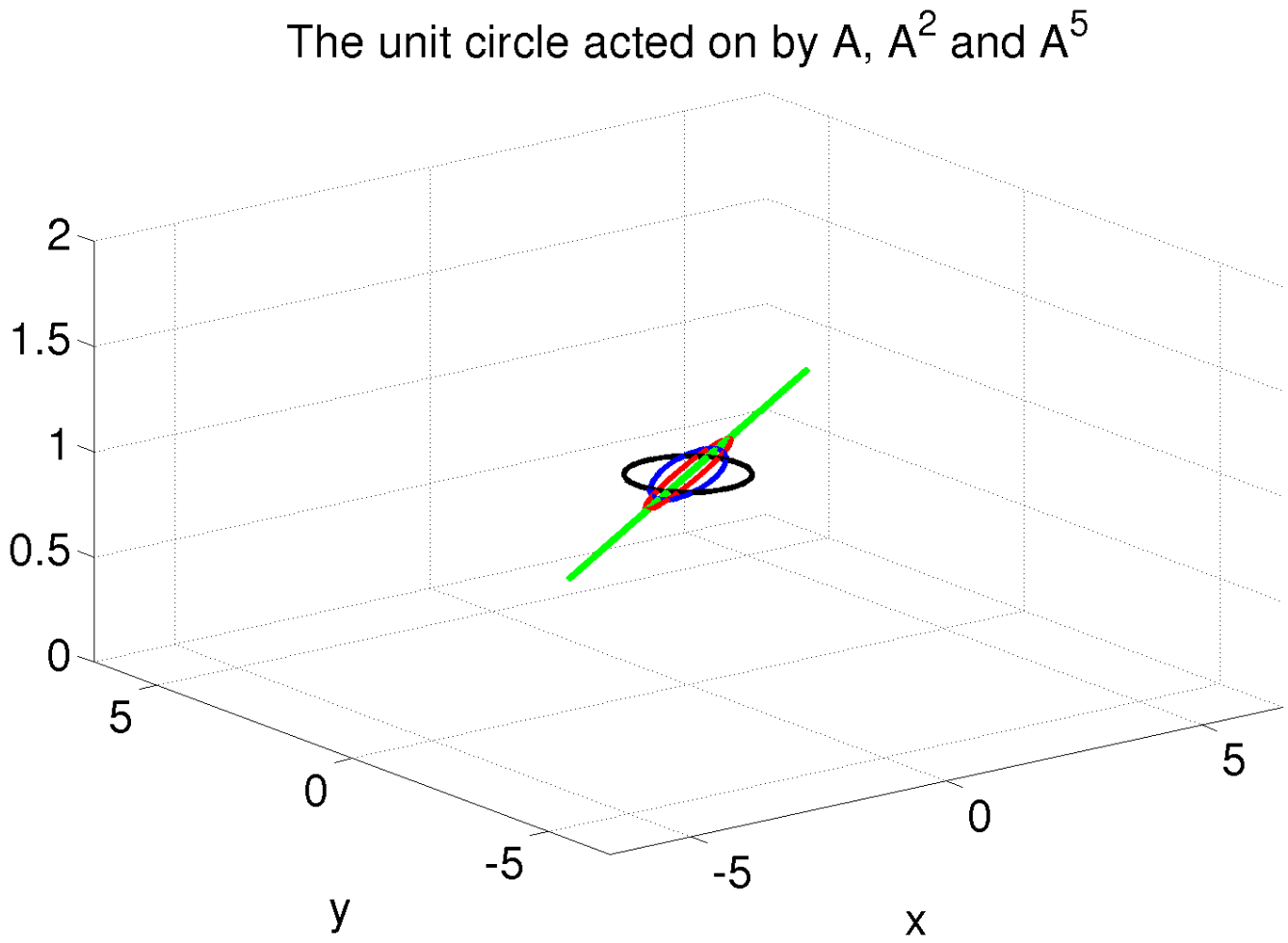
A 1% change in the last coefficient leads to massive changes for

$$p(z) = -120 + 274z - 225z^2 + 85z^3 - 15z^4 + z^5;$$

the roots $(4, 5)$ become $(4.580 \pm 0.966\sqrt{-1})$.

The power method

Want to compute largest eigenvalue without relying on characteristic polynomial. Hint from first computer lab:



The power method: basis vectors

Key to power method: *assumption* that the eigenvectors $\{\boldsymbol{u}_n\}$ form a basis for any complex vector in $\mathbb{C}^n$.

Where true, repeated action of A on *generic* vector $\boldsymbol{x}$ picks out eigenvector with largest eigenvalue.

Specifically, construct sequence of vectors $\{\boldsymbol{x}^{(n)}\}$. Initial guess $\boldsymbol{x}^{(0)}$ (nearly) arbitrary, members of sequence are

$$\boldsymbol{x}^{(k)} = A^k \boldsymbol{x}^{(0)}.$$

Write initial guess in terms of basis of eigenvectors to show

$$\boldsymbol{x}^{(0)} = \sum_{j=1}^{n} a_j \boldsymbol{u}_j \implies \boldsymbol{x}^{(k)} = \lambda_1^k \left[a_1 \boldsymbol{u}_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^k a_2 \boldsymbol{u}_2 + \cdots + \left(\frac{\lambda_n}{\lambda_1}\right)^k a_n \boldsymbol{u}_n \right]$$

If $|\lambda_j/\lambda_1| < 1 \quad \forall j > 1$ then the first term dominates.

Caveats

Some points have been glossed over:

1. Have assumed *unique* eigenvalue of maximum modulus.
2. Have assumed n eigenvectors exist and are linearly independent. This is necessary to have a basis of eigenvectors.
3. Have assumed the initial guess $\boldsymbol{x}^{(0)}$ has a nonzero component in the direction of eigenvector $\boldsymbol{u}_1$; i.e. if

$$\boldsymbol{x}^{(0)} = \sum_{j=1}^{n} a_j \boldsymbol{u}_j \qquad \Longrightarrow \qquad a_1 \neq 0.$$

Not a major problem: repeated numerical operations have floating point error, so a_1 will never be *precisely* zero. Method converges faster the closer that $\boldsymbol{x}^{(0)}$ is aligned with $\boldsymbol{u}_1$.

Error terms

Can write the iterative method given by the power method as

$$\boldsymbol{x}^{(k)} = \lambda_1^k \left(a_1 \boldsymbol{u}_1 + \boldsymbol{\varepsilon}^{(k)} \right)$$

where the term

$$\boldsymbol{\varepsilon}^{(k)} \equiv \sum_{j=2}^{n} \left(\frac{\lambda_j}{\lambda_1} \right)^k a_j \boldsymbol{u}_j$$

is expected to vanish in the limit. Explicitly,

$$\|\boldsymbol{\varepsilon}^{(k)}\| = \mathcal{O}\left(\left| \frac{\lambda_j}{\lambda_1} \right|^k \right) \xrightarrow[k\to\infty]{} 0.$$

In general expect the “error term” at each step to diminish by $|\lambda_2/\lambda_1|$, giving linear convergence, as seen later.

Algorithm: 1

The simplest (and not fully correct) algorithm defines the ratio

$$r_k = \frac{\|\boldsymbol{x}^{(k+1)}\|}{\|\boldsymbol{x}^{(k)}\|} = |\lambda_1| \frac{\|a_1 \boldsymbol{u}_1 + \boldsymbol{\mathcal{E}}^{(k+1)}\|}{\|a_1 \boldsymbol{u}_1 + \boldsymbol{\mathcal{E}}^{(k)}\|}.$$

From the convergence of the "error term" we then have that

$$\lim_{k \to \infty} r_k = |\lambda_1|.$$

Algorithm impractical: unless λ_1 *extremely* close to 1 since iterates diverge to infinity or tend to zero. Instead redefine members of sequence to have unit norm *after* computing the ratio r_k:

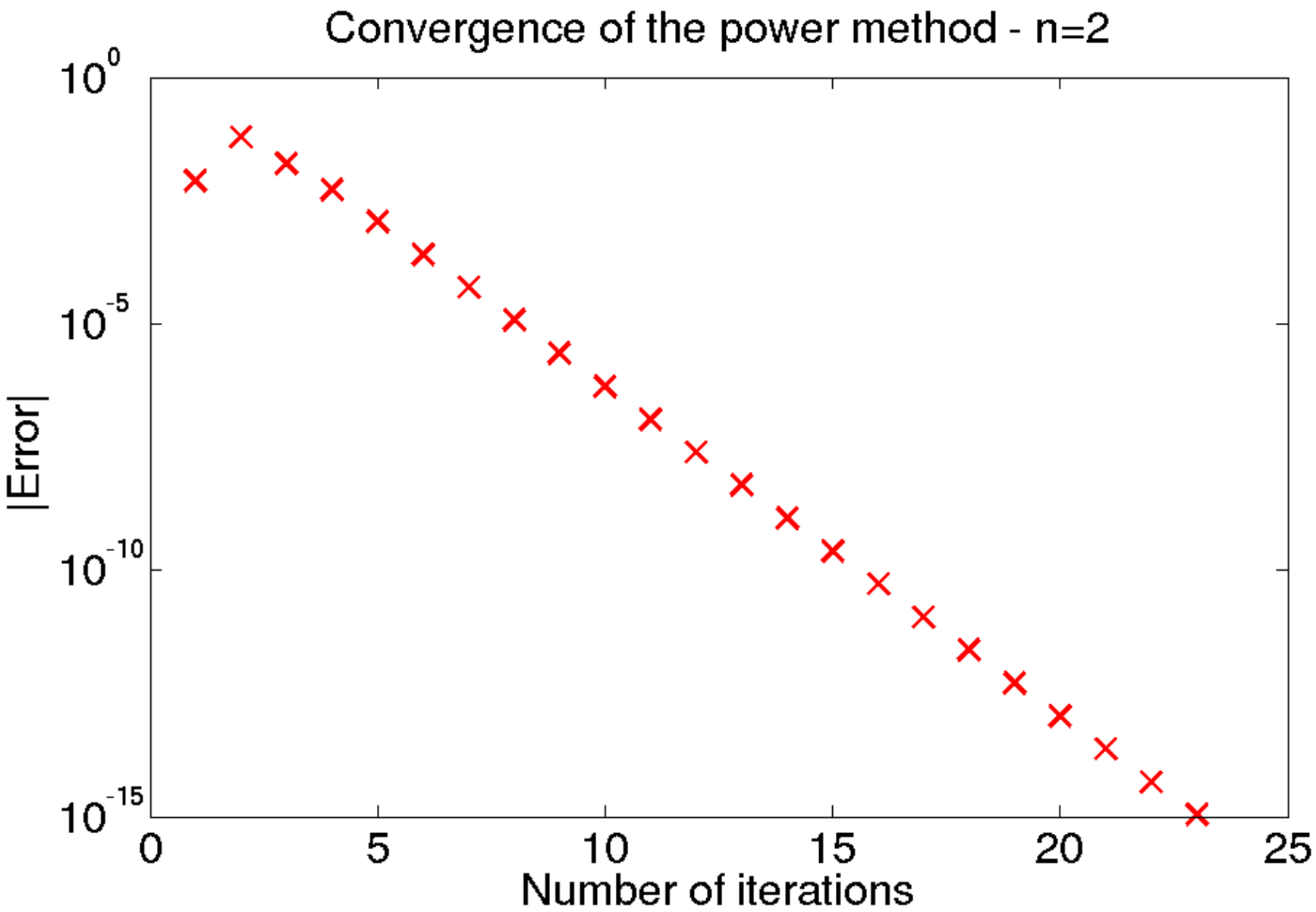
1. Pick $\boldsymbol{x}^{(0)}$ and set $k = 0$
2. Normalise $\boldsymbol{x}^{(k)}$ such that $\boldsymbol{x}^{(k)} \to \boldsymbol{x}^{(k)} / \|\boldsymbol{x}^{(k)}\|$
3. Compute $\boldsymbol{x}^{(k+1)} = A\boldsymbol{x}^{(k)}$.
4. Compute $r_k = \|\boldsymbol{x}^{(k+1)}\|$ (since $\|\boldsymbol{x}^{(k)}\| = 1$).
5. Repeat from step (2).

Example

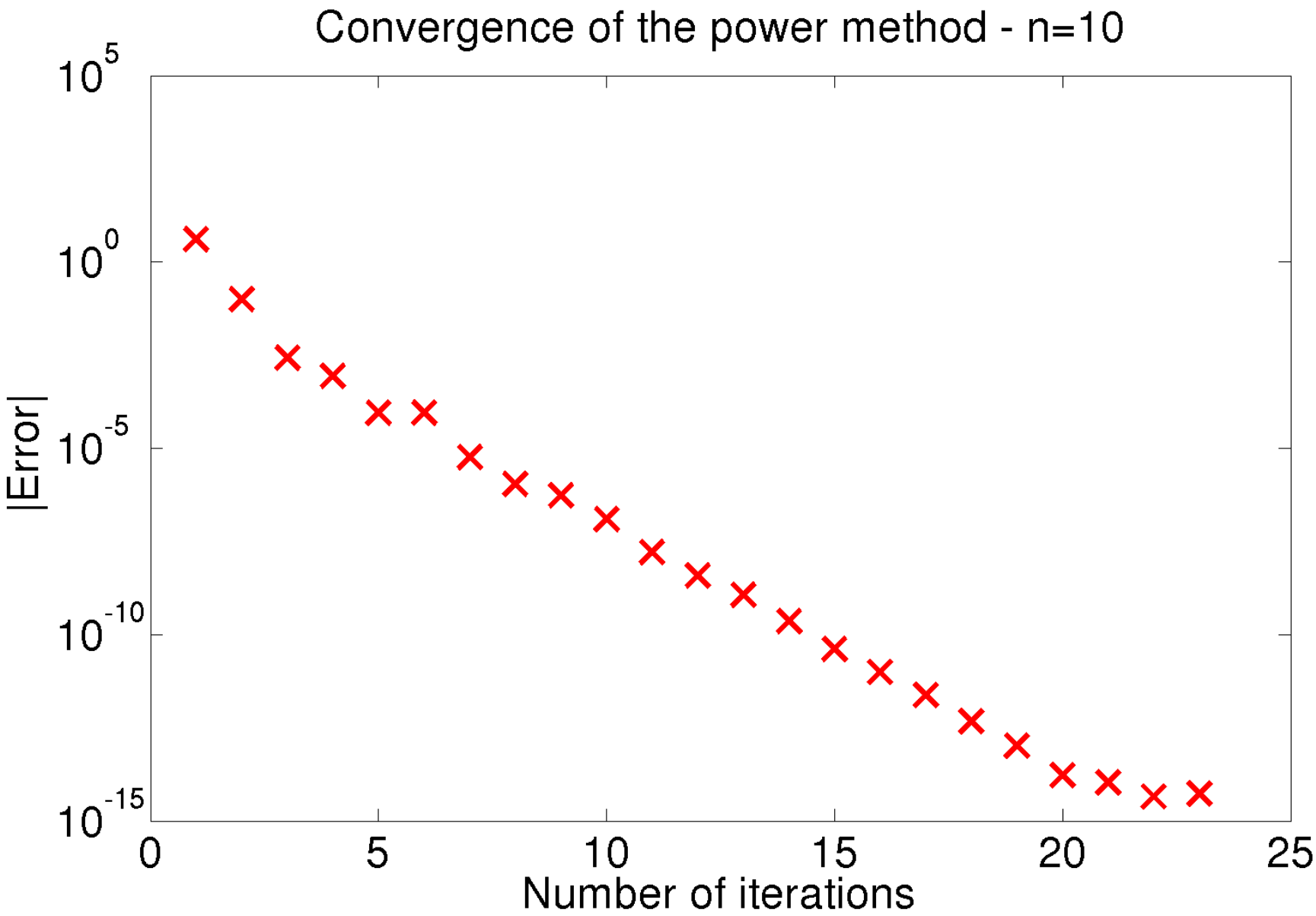
The core of a simple Python script for the power method:

```
for k in range(1, niterations_max):
    xn[:,k-1] = xn[:,k-1]/numpy.linalg.norm(xn[:,k-1])
    xn[:,k] = numpy.linalg.dot(A, xn[:, k-1])
    rn[k] = numpy.linalg.norm(xn[:,k])/numpy.linalg.nc
    if abs(rn[k] - rn[k-1]) < tol:
        break
lambda = rn[k]
```

Example 2


Convergence of the power method - n=2
10^{0}
10^{-5}
10^{-10}
10^{-15}
|Error|
0
5
10
15
20
25
Number of iterations

Example 2


Convergence of the power method - n=10
|Error|
10^{5}
10^{0}
10^{-5}
10^{-10}
10^{-15}
0
5
10
15
20
25
Number of iterations

Beyond the absolute value

Although $\max|\lambda|$ useful, straightforward to modify power method to compute actual full value.

The eigenvalue is complex (in general), so in computing just the *modulus* have lost information about the *phase*. Phase information lost when norms are computed. So replace the norms with a different *linear* functional $\phi : \mathbb{C}^n \to \mathbb{C}$.

Then have

$$r_k = \frac{\phi(\boldsymbol{x}^{(k+1)})}{\phi(\boldsymbol{x}^{(k)})} = \lambda_1 \frac{a_1\phi(\boldsymbol{u}_1) + \phi(\boldsymbol{\varepsilon}^{(k+1)})}{a_1\phi(\boldsymbol{u}_1) + \phi(\boldsymbol{\varepsilon}^{(k)})};$$

depends on the linearity of ϕ. In the limit get full eigenvalue λ_1.

One possible choice for ϕ is to simply sum the components of $\boldsymbol{x}$. Or even more simply to take ϕ to be just one component of the vector. In all cases care must be taken to avoid dividing by zero.

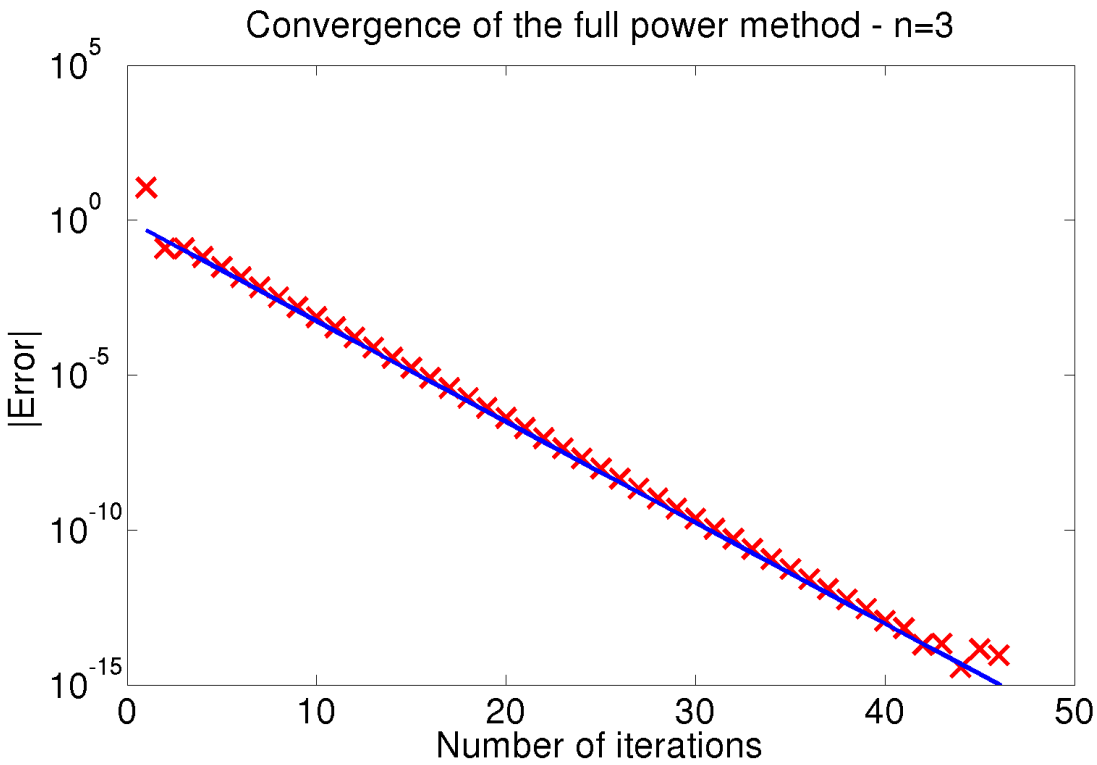
Example

Apply the power method to the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}.$$

The result converges linearly to find $\lambda = 12.1229$.

Identical convergence is seen for $-A$.



Rate of convergence

Look at behaviour near solution using Taylor's theorem.

Start by defining $\mu = \lambda_2/\lambda_1$. Use as "small parameter" in expansion. Note that

$$\left|\frac{\lambda_j}{\lambda_1}\right| < |\mu| \quad \forall j > 2.$$

Rewrite ratio as

$$r_k = \lambda_1 \frac{a_1\phi(\boldsymbol{u}_1) + \phi(\boldsymbol{\varepsilon}^{(k+1)})}{a_1\phi(\boldsymbol{u}_1) + \phi(\boldsymbol{\varepsilon}^{(k)})} = \lambda_1 \left[1 - \phi(\boldsymbol{\varepsilon}^{(k)})\right] + \mathcal{O}(\mu^{k+1}).$$

The relative error is then

$$E^{(k)} = \left|\frac{r_k - \lambda_1}{\lambda_1}\right| = \left|\phi(\boldsymbol{\varepsilon}^{(k)})\right| + \mathcal{O}(\mu^{k+1}) = c_k\mu^k.$$

Hence we have a linear decrease at each stage of a factor μ.

Example revisited

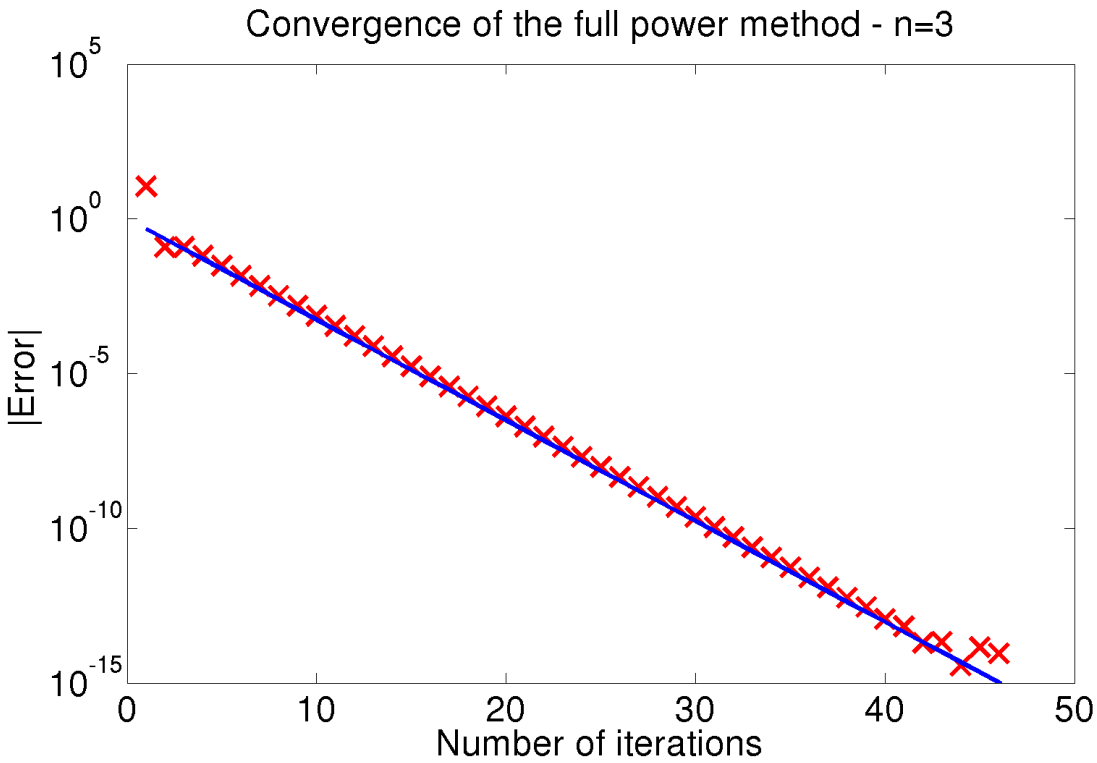
The matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}$$

has eigenvalues

$$\begin{cases} 12.1229 \\ -5.7345 \\ -0.3884 \end{cases}.$$

Therefore the slope of the line should be $\log(|\mu|) \simeq -0.749$; the actual best fit line used had slope -0.750.



Summary

- Eigenvalues of matrices contain useful information. So frequently want to compute them. The largest eigenvalues are frequently the most important for, e.g. the spectral radius.
- Computing the eigenvalues from the characteristic polynomial is expensive and may be numerically ill-conditioned.
- The power method is an iterative scheme for finding the largest eigenvalue. It assumes that
  1. There is a single largest eigenvalue;
  2. The eigenvectors are linearly independent and form a basis.
- The power method converges linearly.
- The power method works when there are repeated eigenvalues; the eigenvector cannot be found, however. With distinct “largest” eigenvalues it may fail.

Chapter 20 - More on Eigenvalues

MATH3018/6141, Semester 1

The power method revisited

We want to compute the eigenvalues (and vectors) of a general $n \times n$ matrix, which may be very large.

The power method gives the largest eigenvalue, in absolute magnitude, as long as it is unique and the eigenvectors are independent. It does this by constructing a sequence, multiplying each time by the matrix A and normalizing.

Results in a simple algorithm, and when we only need the largest eigenvalue, is a good method.

Often require more than the largest eigenvalue. There are variants on the power method that can be used to find other eigenvalues.

Inverse power method

E.g. If we want to find the *smallest* eigenvalue.

Use fact that

If λ_i are eigenvalues of $A \Rightarrow 1/\lambda_i$ are eigenvalues of A^{-1} since

$$A\boldsymbol{x_i} = \lambda_i \boldsymbol{x}_i \quad \Longrightarrow \quad \boldsymbol{x_i} = \lambda_i A^{-1} \boldsymbol{x}_i \quad \Longrightarrow \quad A^{-1}\boldsymbol{x}_i = \frac{1}{\lambda_i}\boldsymbol{x}_i.$$

So apply power method to inverse matrix:

$$A\boldsymbol{x}_{n+1} = \boldsymbol{x}_n. \quad \Longrightarrow \quad \boldsymbol{x}_{n+1} = A^{-1}\boldsymbol{x}_n$$

Converges towards eigenvector whose eigenvalue has *minimum* modulus. Again, normalize at each step.

Do *not* use A^{-1} directly, but solve linear system; decomposition methods particularly effective.

Inverse power method example

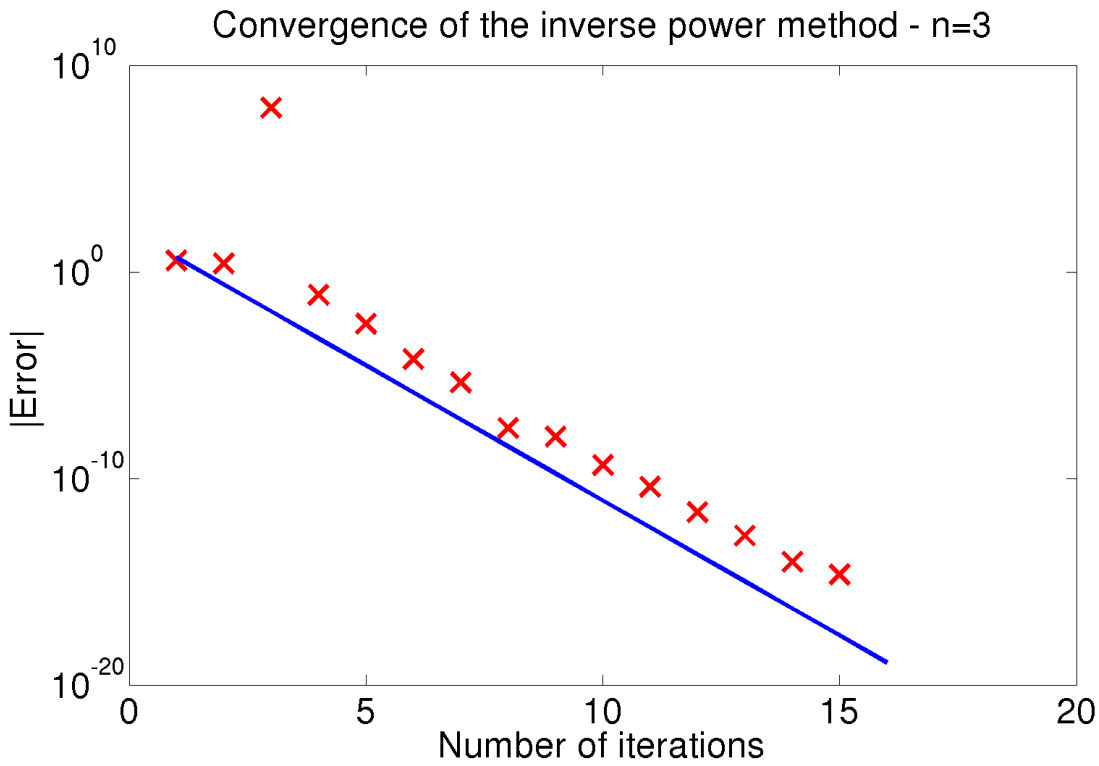
The matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}$$

has eigenvalues

$$\begin{cases} 12.1229 \\ -5.7345 \\ -0.3884 \end{cases}.$$

The inverse power method shows linear convergence towards $\lambda = -0.3884$.



Shifted power method

Another minor variant allows us to find the eigenvalue closest to a given complex number σ. We make use of fact that

If λ_i are eigenvalues of A then $\lambda_i - \sigma$ are eigenvalues of $A - \sigma I$

Therefore the smallest eigenvalue of $A - \sigma I$ is the one closest to σ.

This is just an application of the inverse power method.

Inverse power method example

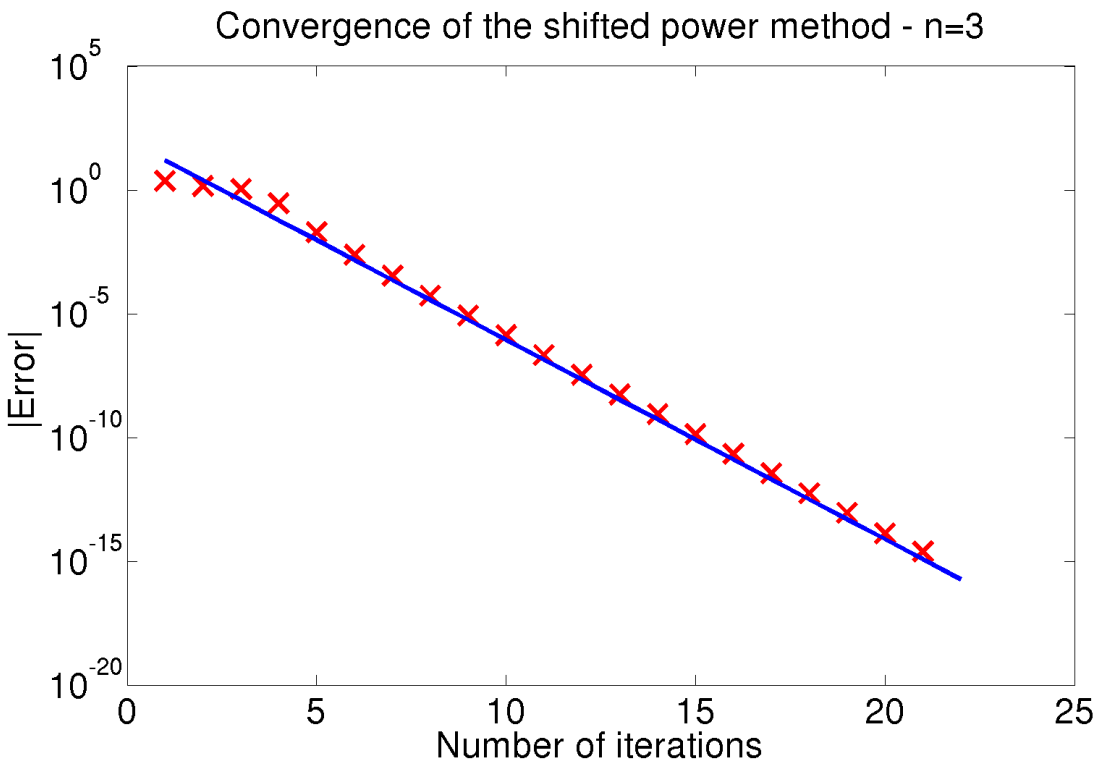
The matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}$$

has eigenvalues

$$\begin{cases} 12.1229 \\ -5.7345 \\ -0.3884 \end{cases}.$$

The shifted power method shows linear convergence to $\lambda = -5.7345$ for the eigenvalue closest to -5.



Summary

- The power method is an iterative procedure giving the eigenvalue of largest absolute modulus and its associated eigenvector.
- By looking at the inverse matrix the power method can be modified to give
  1. the eigenvalue of smallest absolute modulus (inverse power method)
  2. the eigenvalue closest to a given $\sigma \in \mathbb{C}$ (shifted inverse power method).