Numerical Methods for Differential Equations

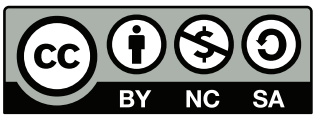
Ferran Arqué

2018

Notes for the course *Numerical Methods for Differential Equations* taught by Jaume Soler and Esther Sala at Facultat de Matemàtiques i Estadística of Universitat Politècnica de Catalunya.

Contents

This work is licensed under a Creative Commons "Attribution-NonCommercial-ShareAlike 4.0 International" license.



ORDINARY DIFFERENTIAL EQUATIONS

1 Ordinary Differential Equations. Basic concepts

1.1 Introduction and some notation

Given $y' = f(x, y)$, where $\begin{cases} y(x) \in \mathbb{R}^n \\ f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n \end{cases}$

Definition. We denote by $y(x)$ the exact solution of the ODE system above.

Definition. y_k is the approximation of $y(x_k)$ (after k steps).

Objective. We want to approximate $y(x)$ within a given interval $[x_0, x_n]$.

We know $\begin{cases} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{cases}$ We'd like to know $\begin{cases} y(x_0) \\ y(x_1) \\ y(x_2) \\ \vdots \\ y(x_n) \end{cases}$ We find $\begin{cases} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{cases}$ (given by a method)

Definition. $\|y(x_n) - y_n\|$ is the **global error**.

Definition. We define the **local truncation error** as the error caused by one iteration, i.e.

$$LTE = \|y(x_k) - y_k\| \qquad \big(\text{assuming the } \textit{localizing assumption}\text{: } y_{k-1} = y(x_{k-1})\big)$$

Definition. A method is of **order** p if the LTE is of the form

$$\text{LTE} = Ch^{p+1} + \mathcal{O}(h^{p+2})$$

1.2 Euler's method

1.2.1 Formulation

Consider the differential equation system

$$y'(x) = f(y(x)), \qquad y(x_0) = y_0 \qquad (\text{with } f : \mathbb{R}^n \to \mathbb{R}^n) \tag{1.1}$$

We'll also assume, for later results, that f satisfies a Lipschitz condition

$$\|f(y) - f(z)\| \leq L \|y - z\|$$

Now, we want to find an approximation of (1.1) on an interval $[x_0, x_N]$, so we'll create a set of $N + 1$ evenly spaced points between x_0 and x_N, where N is the number of steps and $h = \dfrac{x_N - x_0}{N}$ the stepsize.

To do that, we can use Euler's method

$$y_{n+1} = y_n + hf(y_n)$$

1.2.2 Local truncation error

We want to find $LTE = \|y(x_{n+1}) - y_{n+1}\|$. Let's expand it

$$\begin{aligned} y(x_{n+1}) - y_{n+1} = y(x_n + h) - y_{n+1} &= y(x_n) + hy'(x_n) + \mathcal{O}(h^2) - y_n - hf(y_n) \underset{\text{Loc.ass.}}{=} \\ &= y(x_n) + hy'(x_n) + \mathcal{O}(h^2) - y(x_n) - hf(y(x_n)) \\ &= hy'(x_n) + \mathcal{O}(h^2) - hy'(x_n) = \mathcal{O}(h^2) \end{aligned}$$

So LTE $= \mathcal{O}(h^2)$, and the method has order $p = 1$.

1.2.3 Convergence

For the method to be convergent, it has to verify

$$\|y(x_N) - y_N\| \underset{h\to 0}{\longrightarrow} 0$$

Similar to what we did with the LTE, we'll find a bound for the error in one step, but this time without using the localizing assumption:

$$\begin{aligned} \|y(x_{n+1}) - y_{n+1}\| = \|y(x_n + h) - y_n - hf(y_n)\| &\underset{\text{Taylor}}{=} ||y(x_n) + h\underbrace{y'(x_n)}_{f(y(x_n))} + \tilde{c}h^2 - y_n - hf(y_n)|| \\ &\leq \|y(x_n) - y_n\| + h\underbrace{\|f(y(x_n)) - f(y_n)\|}_{\underset{L\|y(x_n)-y_n\|}{|\wedge}} + ch^2 \leq (1 + hL)\|y(x_n) - y_n\| + ch^2 \quad \forall n \end{aligned}$$

Thus

$$\begin{aligned} &\|y(x_1) - y_1\| \leq (1 + hL)\|y(x_0) - y_0\| + c_1h^2 \\ &\|y(x_2) - y_2\| \leq (1 + hL)\|y(x_1) - y_1\| + c_2h^2 \leq (1 + hL)^2\|y(x_0) - y_0\| + (1 + hL)c_1h^2 + c_2h^2 \\ &\|y(x_3) - y_3\| \leq (1 + hL)^3\|y(x_0) - y_0\| + (1 + hL)^2c_1h^2 + (1 + hL)c_2h^2 + c_3h^2 \\ &\quad\vdots \end{aligned}$$

Let $C = \max_i c_i$

Then

$$\|y(x_N) - y_N\| \leq (1 + hL)^N\|y(x_0) - y_0\| + \underbrace{\left(1 + (1 + hL) + (1 + hL)^2 + \ldots + (1 + hL)^{N-1}\right)}_{\underset{\frac{(1+hL)^N-1}{hL}}{\text{geom}\to\|}} Ch^2$$

$$\implies \|y(x_n) - y_N\| \leq (1 + hL)^N \underbrace{\|y(x_0) - y_0\|}_{\underset{0}{(1.1)\|}} + \frac{(1 + hL)^N - 1}{\not{h}L} \cdot C\not{h}^2 \leq Ah \to 0$$

Remark. In the last inequality we used $(1 + hL)^N = e^{N\log(1+hL)} \underset{\substack{\log(1+x)\\ \leq x+cx^2}}{\leq} e^{N(hL+ch^2L^2)} =$

$= e^{L(x_N-x_0)} \cdot e^{\tilde{\alpha}(x_N-x_0)h} = \tilde{A} \cdot e^{\alpha h} \to \tilde{A} \cdot 1 = \tilde{A} \qquad (Nh = x_N - x_0)$

1.3 Improved Euler's method

1.3.1 Formulation

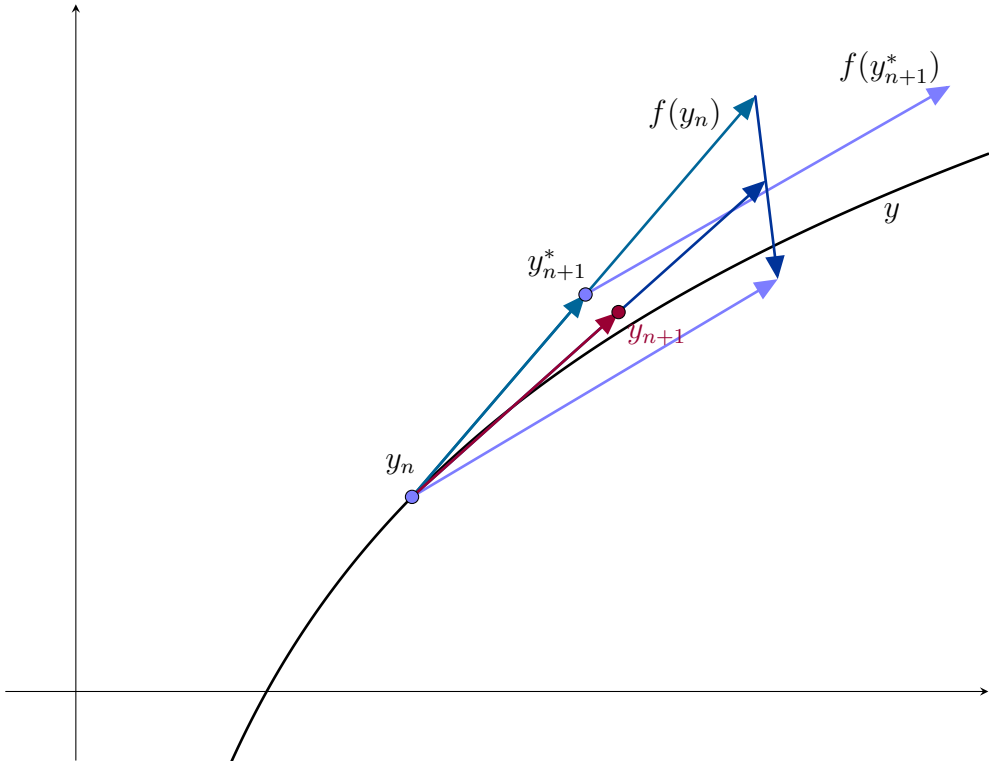


Figure 1.1: One step of the enhanced Euler's method

Given $y'(x) = f(y(x)), \quad y : \mathbb{R} \to \mathbb{R}, \quad f : \mathbb{R} \to \mathbb{R}$, we'll go through the steps to deduce the improved Euler's method (also called Heun's method) with the help of the scheme in Figure 1.1.

The auxiliary point y^*_{n+1} can be found doing one step of the standard Euler's method, so

$$y^*_{n+1} = y_n + h \cdot f(y_n)$$

To get the point y_{n+1} we compute the average vector of $f(y^*_{n+1})$ and $f(y_n)$, and with this new vector, we can apply again a step of Euler's method, ending up with our method

$$\boxed{y_{n+1} = y_n + \frac{h}{2} \cdot \big(f(y^*_{n+1}) + f(y_n)\big)}$$

1.3.2 Local truncation error

The Local Truncation Error is given by:

$$LTE = \|method - exact\ solution\|$$

Then

$$LTE = \|y_{n+1} - y(x_{n+1})\| = \left\| y_n + \frac{h}{2} f(y_n) + \frac{h}{2} f(y^*_{n+1}) - \underbrace{y(x_{n+1})}_{y(x_n+h)} \right\| = (*)$$

Applying Taylor on $f(y^*_{n+1}) = f\left(y_n + hf(y_n)\right)$, we have

$$f\left(y_n + hf(y_n)\right) = f(y_n) + hf(y_n)f'(y_n) + \mathcal{O}(h^2)$$

and on $y(x_n + h)$ we have

$$y(x_n + h) = y(x_n) + h \cdot y'(x_n) + \frac{h^2}{2} \cdot y''(x_n) + \mathcal{O}(h^3)$$

(In this case, we expand to second order for later simplifications)

$$(*) = \left\| y_n + \frac{h}{2} f(y_n) + \frac{h}{2} \left(f(y_n) + hf(y_n)f'(y_n) + \mathcal{O}(h^2) \right) - \left(y(x_n) + h \cdot y'(x_n) + \frac{h^2}{2} \cdot y''(x_n) + \mathcal{O}(h^3) \right) \right\| \quad (1)$$

Now, given $y'(x) = f(y(x))$, we have

$$\begin{aligned} y''(x) &= f'(y(x)) \cdot y'(x) \\ &= f'(y(x)) \cdot f(y(x)) \end{aligned}$$

and we can rewrite the following expression as:

$$\frac{h^2}{2} f(y(x))f'(y(x)) = \frac{h^2}{2} y''(x)$$

With that and the localising assumption ($y_n = y(x_n)$), we can simplify most of the terms in (1) and we end up with

$$\left\| \cancel{y_n} + \cancel{\frac{h}{2} f(y_n)} + \cancel{\frac{h}{2} f(y_n)} + \cancel{\frac{h^2}{2} f(y_n) f'(y_n)} + \mathcal{O}(h^3) - \left(\cancel{y(x_n)} + \cancel{h \cdot y'(x_n)} + \cancel{\frac{h^2}{2} \cdot y''(x_n)} + \mathcal{O}(h^3) \right) \right\| = \mathcal{O}(h^3)$$

So LTE $= \mathcal{O}(h^3)$

Remark. Of course, this method also works for $y : \mathbb{R} \to \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}^n$

1.4 Final remarks

- There's an enhanced Euler's method of order 2 similar to the previous one:

$$\boxed{y_{n+1} = y_n + hf\left(\frac{y^*_{n+1} + y_n}{2}\right)}$$

- If we have a method of order ≥ 2, and we want the value of $y(x^*)$, with x^* off the mesh ($x^* \neq k \cdot h$), we have some options:

  -Step back and take a step with the right h.

  -Interpolate with the right order.

  -Use a continuous Runge-Kutta method.

2 Runge-Kutta and Linear Multistep Methods

2.1 General Runge-Kutta methods

Runge-Kutta methods are a family of iterative methods, which include the previously seen Euler's Method. Let's define an RK method with s stages:

Given $x \in \mathbb{R}$, $y \in \mathbb{R}^n$, $y' = f(x, y)$

Definition. The **Butcher Tableau** is

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ c_s & a_{s1} & \dots & \dots & a_{ss} \\ \hline & b_1 & \dots & \dots & b_s \end{array}$$

With these coefficients given by the table, we can now define our Runge-Kutta method:

$$\left.\begin{aligned} k_1 &= f\Big(x_n + c_1 h, y_n + h(a_{11}k_1 + a_{12}k_2 + \ldots + a_{1s}k_s)\Big) \\ k_2 &= f\Big(x_n + c_2 h, y_n + h(a_{21}k_1 + a_{22}k2 + \ldots + a_{2s}k_s)\Big) \\ &\vdots \\ k_s &= f\Big(x_n + c_s h, y_n + h(a_{s1}k_1 + a_{s2}k2 + \ldots + a_{ss}k_s)\Big) \end{aligned}\right\} \text{System of equations with unknowns } k_1,k_2,\ldots,k_s$$

$$\boxed{y_{n+1} = y_n + h(b_1k_1 + b_2k_2 + \ldots + b_sk_s)}$$

Cases:

(1) Explicit ($a_{ij} = 0$ for $j \geq i$ and $c_1 = 0$)

(2) Semi-implicit ($a_{ij} = 0$ for $j > i$)

(3) Implicit

Remark. (2) and (3) are used for stiff problems.

Theorem 2.1.1. An explicit s-stage Runge-Kutta method can't have order $> s$.

Theorem 2.1.2. There is no explicit 5-stage RK of order 5.

Theorem 2.1.3.

Let

$A =$ order p for $y' = f(y)$, $f : \mathbb{R}^m \to \mathbb{R}^m$, $m > 1$
$B =$ order p for $y' = f(x, y)$, $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
$C =$ order p for $y' = f(y)$, $f : \mathbb{R} \to \mathbb{R}$

Then

For $1 \le p \le 3$, $A \iff B \iff C$
For $p = 4$, $A \iff B \implies C$ but $C \not\Rightarrow B$
For $p \ge 5$, $A \implies B \implies C$ but $C \not\Rightarrow B$, $B \not\Rightarrow A$

2.1.1 Embedded Runge-Kutta methods

Embedded methods combine two methods of order p and $p+1$ to obtain an estimate of the LTE. This error estimate is useful for adaptive stepsize methods.

In this case, the *Butcher tableau* is

$$\begin{array}{c|cccc|l}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & & \ddots & & \\
c_s & a_{s1} & \cdots\cdots & a_{s,s-1} & & \\
\hline
 & b_1 & \cdots\cdots & b_{s-1} & b_s & \text{order } p \\
 & \overline{b}_1 & \cdots\cdots & \overline{b}_{s-1} & \overline{b}_s & \text{order } p+1
\end{array}$$

This kind of method has the advantage that it usually requires less steps even though there are more evaluations for each step.

Matlab's current `ode45` solver uses an embedded RK method with orders 4 and 5.

2.2 Linear multistep methods

Definition. A **linear k-step method** is a method of the form

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f_{n+j}$$

with α_j, $\beta_j \in \mathbb{R}$, $\quad \alpha_k = 1$ and $\alpha_0 \neq 0$ *or* $\beta_0 \neq 0$.

Notation: $f_m = f(x_m, y_m)$, $x_m = x_0 + mh$

Remark. If $\beta_k = 0$, then the method is explicit.

Remark. With these methods, adapting the step size can be complicated. It may even change the order.

Example 2.2.1
Our initial value problem is

$$\begin{cases} y' = f(x, y) \\ y_0 = y(x_0) \end{cases}$$

And we want to solve it using the following multistep method:

$$y_{n+2} - y_n = \frac{h}{2}(f_{n+2} + 4f_{n+1} + f_n)$$

How do we compute y_1?

Usually we first use a one-step method like Euler's to compute the initial terms needed, and then we apply the multistep method.

Example 2.2.2
Let's see how we compute the order of $y_{n+2} - y_n = h(\beta_1 f_{n+1} + \beta_0 f_n)$:

$$\text{LTE} = \|y(x_{n+2}) - y_{n+2}\|$$

$$\begin{aligned} y(x_n + 2h) - y_{n+2} &= \cancel{y(x_n)} + 2hy'(x_n) + \frac{1}{2}y''(x_n)4h^2 + \frac{1}{3!}y'''(x_n)8h^3 + \mathcal{O}(h^4) - \\ &\quad - \left(\cancel{y(x_n)} + h\beta_1 f(y_{n+1}) + h\beta_0 f(y_n)\right) \\ &= 2hy'(x_n) + \frac{1}{2}y''(x_n)4h^2 + \frac{1}{3!}y'''(x_n)8h^3 + \mathcal{O}(h^4) - \\ &\quad - \Big(h\beta_1 \underbrace{y'(x_n + h)}_{y'(x_n)+hy''(x_n)+\ldots} + h\beta_0 y'(x_n)\Big) \end{aligned}$$

Grouping by powers of h we get:

$h \longrightarrow 2y'(x_n) - \beta_1 y'(x_n) - \beta_0 y'(x_n) = 0 \implies \beta_0 + \beta_1 = 2$

$h^2 \longrightarrow 2y''(x_n) - \beta_1 y''(x_n) = 0 \implies b_1 = 2 \implies \beta_0 = 0$

So LTE $= \mathcal{O}(h^3)$

2.2.1 Generalities

- **Adams–Bashforth methods**: These are explicit methods with coefficients $\alpha_{k-1} = -1$ and $\alpha_{k-2} = \alpha_{k-3} = \ldots = \alpha_0 = 0$. The β_j's are chosen such that the method has order k $(j = 0, \ldots, k)$.

  For example, the Adams–Bashforth method with $k = 2$ is

  $$y_{n+2} = y_{n+1} + h\left(\frac{3}{2}f_{n+1} - \frac{1}{2}f_n\right)$$

- **Adams–Moulton methods**: These have the same coefficients as the Adams–Bashforth methods, but they are implicit methods. Also, for a k-step Adams–Moulton method, if $\beta_k \neq 0$, it can reach order $k + 1$, while an A-B method has only order k.

  The Adams–Moulton method with $k = 2$ is

  $$y_{n+2} = y_{n+1} + h\left(\frac{5}{12}f_{n+2} + \frac{2}{3}f_{n+1} - \frac{1}{12}f_n\right)$$

- **Backward differentiation formulas (BDF)**: These are implicit methods with $b_{k-1} = \ldots = b_0 = 0$ and the rest of the coefficients are chosen such that the method has order k (only exist for $k < 7$). BDF methods are useful for stiff problems.

- **Predictor-Corrector method**: A Predictor-Corrector method uses an explicit method for the predictor step (P) and an implicit method for the corrector step (C). They usually have the same order.

  $$y_n \underset{P}{\longrightarrow} y^*_{n+1} \underset{C}{\longrightarrow} y_{n+1}$$

  They have the advantage that the LTE is easy to estimate ($|predictor - corrector|$) The improved Euler's method (1.3) is an example of a P-C method.

2.2.2 Richardson's extrapolation

Richardson's extrapolation is used to improve the rate of convergence of a method.

Suppose we have a one-step explicit method. Then

- Compute one step with stepsize h: $y_n \to y_{n+1}$
- Compute two steps with stepsize $\frac{h}{2}$: $y_n \to y_{n+\frac{1}{2}} \to \hat{y}_{n+1}$

Then

$$\text{LTE} \simeq \|y_{n+1} - \hat{y}_{n+1}\|$$

If the method is of order p, we have

$$y(x_{n+1}) - y_{n+1} = Ch^{p+1} + \mathcal{O}(h^{p+2}) \tag{2.1}$$

$$y(x_{n+1}) - \hat{y}_{n+1} = 2C\left(\frac{h}{2}\right)^{p+1} + \mathcal{O}(h^{p+2}) \tag{2.2}$$

And subtracting (2.1) - (2.2), we end up with

$$\hat{y}_{n+1} - y_{n+1} = C\left(h^{p+1} - 2\frac{h^{p+1}}{2^{p+1}}\right) = Ch^{p+1}\underbrace{\left(1 - \frac{1}{2^p}\right)}_{\frac{2^p-1}{2^p}} \simeq \text{LTE}$$

and

$$C = \frac{\|\hat{y}_{n+1} - y_{n+1}\|}{h^{p+1}} \cdot \frac{2^p}{2^p - 1}$$

Now, while applying our method, we want to keep the LTE below a tolerance TOL. If the LTE is larger than TOL, we want to reduce the stepsize, and if it's significantly smaller, we don't want to waste computational time, so we should increase the stepsize. So an automatic adjustment of h can be computed as follows:

Let h^* be a new stepsize such that the LTE is equal to the tolerance ($C \cdot (h^*)^{p+1} = TOL$).

Then

$$(h^*)^{p+1} = \frac{TOL}{C} = \frac{TOL}{\|\hat{y}_{n+1} - y_{n+1}\|} \cdot \frac{2^p - 1}{2^p} h^{p+1}$$

And the new stepsize is given by

$$h^* = 0.9h\left(\frac{TOL}{\|\hat{y}_{n+1} - y_{n+1}\|} \cdot \frac{2^p - 1}{2^p}\right)^{\frac{1}{p+1}}$$

2.2.3 Convergence of a linear multistep method

Theorem 2.2.1. Given a general linear multistep method

$$y_{n+k} + \alpha_{k-1} y_{n+k-1} + \ldots + \alpha_0 y_n = h\big(\beta_k f_{n+k} + \ldots + \beta_0 f_n\big)$$

and the polynomials

$$\rho(z) = z^k + \alpha_{k-1} z^{k-1} + \ldots + \alpha_0$$
$$\sigma(z) = \beta_k z^k + \beta_{k-1} z^{k-1} + \ldots + \beta_0$$

Then, a necessary and sufficient condition for this method to be convergent is:

1. It is of order 1 at least.
2. The roots of the first stability polynomial ($\rho(z)$) are on the unit ball, and if they are on the boundary, they are simple (as roots).

Remark. 1 is always a root of $\rho(z)$.

Let's see an example of divergence using a linear multistep method:

Example 2.2.3

Given the method

$$\boldsymbol{y_{n+2} + a_1 y_{n+1} + a_0 y_n = h(b_1 f_{n+1} + b_0 f_n)}$$

1) Find a_0, a_1, b_0, b_1 so that the method above has the highest possible order.

2) Try it on

$$\begin{cases} \boldsymbol{y' = -y} \\ \boldsymbol{y(0) = 1} \end{cases} \qquad (\boldsymbol{y_0 = 1, y_1 = e^{-h}})$$

and prove the method diverges.

1) We want $y(x_n + 2h) - y_{n+2}$

We assume $y_{n+1} = y(x_n + h), y_n = y(x_n)$ (localizing assumption).

$$y(x_n + 2h) - y_{n+2} = y(x_n + 2h) - \Big[- a_1 y_{n+1} - a_0 y_n + h\big(b_1 f(y_{n+1}) + b_0 f(y_n)\big)\Big] \underset{\text{loc.as.}}{=}$$
$$= y(x_n + 2h) - \Big[- a_1 y(x_n + h) - a_0 y(x_n) + h b_1 \underbrace{f\left(y(x_n + h)\right)}_{y'(x_n+h)} + h b_0 \underbrace{f\left(y(x_n)\right)}_{y'(x_n)} \Big]$$

As usual, we expand in powers of h. We'll expand to order 3

$$\begin{aligned}
&y(x_n) + 2hy'(x_n) + \frac{4h^2}{2}y''(x_n) + \frac{8h^3}{6}y'''(x_n) + o(h^4) - \\
&\quad - \Bigg[- a_1 \left(y(x_n) + hy'(x_n) + \frac{h^2}{2}y''(x_n) + \frac{h^3}{6}y'''(x_n) + o(h^4) \right) - \\
&\qquad - a_0 y(x_n) \\
&\qquad + hb_1 \left(y'(x_n) + hy''(x_n) + \frac{h^2}{2}y'''(x_n) + o(h^3) \right) + \\
&\qquad + hb_0 y'(x_n) \Bigg]
\end{aligned}$$

Let's group by powers of h and assume the right conditions to obtain the highest possible order:

$$h^0 \longrightarrow y(x_n) + a_1 y(x_n) + a_0 y(x_n) = 0$$

$$h^1 \longrightarrow 2hy'(x_n) + a_1 hy'(x_n) - hb_1 y'(x_n) - hb_0 y'(x_n) = 0$$

$$h^2 \longrightarrow 2h^2 y''(x_n) + a_1 \frac{1}{2}h^2 y''(x_n) - b_1 h^2 y''(x_n) = 0$$

$$h^3 \longrightarrow \frac{8h^3}{6}y'''(x_n) + a_1 \frac{h^3}{6}y'''(x_n) - b_1 h \left(\frac{h^2}{2}y'''(x_n) \right) = 0$$

With that, we get the system of equations

$$\begin{cases}
1 + a_1 + a_0 = 0 \\
2 + a_1 - b_1 - b_0 = 0 \\
2 + \dfrac{a_1}{2} - b_1 = 0 \\
\dfrac{8}{6} + \dfrac{a_1}{6} - \dfrac{b_1}{2} = 0
\end{cases}$$

And we end up with

$$a_0 = -5, \quad a_1 = 4, \quad b_0 = 2, \quad b_1 = 4$$

2) Our method is

$$y_{n+2} + 4y_{n+1} - 5y_n = h(4f_{n+1} + 2f_n)$$

and with

$$\begin{cases} y' = -y \\ y(0) = 1,\ y(h) = e^{-h} \end{cases} \qquad (y(x) = e^{-x})$$

we have

$$y_{n+2} + 4y_{n+1} - 5y_n = h(-4y_{n+1} - 2y_n)$$

We'll find a solution of the form

$$y_n = c_1(\quad)^n + c_2(\quad)^n$$

and we'll see that it diverges.

$$\lambda^2 + 4\lambda - 5 + 4h\lambda + 2h = 0$$
$$\lambda^2 + (4(1+h))\lambda + (2h-5) = 0$$

$$\lambda = \frac{-4(1+h) \pm \sqrt{4^2(1+h)^2 - 4(2h-5)}}{2}$$

Let's expand the discriminant

$$\sqrt{4^2(1+2h+h^2) - 8h + 20} = \sqrt{36 + 24h + 16h^2} = 6\sqrt{1 + \frac{4}{6}h + \frac{4^2}{6^2}h^2} \underset{\text{Taylor}}{=}$$
$$= 6\left(1 + \frac{1}{2}\left(\frac{4}{6}h + \frac{4^2}{6^2}h^2\right) + o(h^2)\right) =$$
$$= 6\left(1 + \frac{1}{3}h + o(h^2)\right)$$

So

$$\lambda = \frac{-4 - 4h \pm (6 + 2h + o(h^2))}{2} = \begin{array}{l} \nearrow\ 1 - h + o(h^2) \\ \searrow\ -5 - 3h + o(h^2) \end{array}$$

$$\implies y_n = c_1\big(1 - h + o(h^2)\big)^n + c_2\big(-5 - 3h + o(h^2)\big)^n$$

Let's find c_1 and c_2 imposing the initial conditions

$$\begin{cases} 1 = c_1 + c_2 \implies c_1 = 1 - c_2 \\ e^{-h} = c_1\big(1 - h + o(h^2)\big) + c_2\big(-5 - 3h + o(h^2)\big) \end{cases}$$

$$\implies e^{-h} = (1 - h + o(h^2)) + c_2\big(\underbrace{-5 - 3h + o(h^2) - 1 + h + o(h^2)}_{-6-2h+o(h^2)}\big)$$

$$\implies c_2 = \frac{e^{-h} - 1 + h + o(h^2)}{-6 - 2h + o(h^2)} \underset{\text{Taylor } e^{-h}}{=} \frac{1 - h + o(h^2) - 1 + h + o(h^2)}{-6 - 2h + o(h^2)} = \frac{o(h^2)}{-6 - 2h + o(h^2)}$$

$$\implies c_1 = 1 + \frac{1}{6 + 2h + o(h^2)} = \frac{7 + 2h + o(h^2)}{6 + 2h + o(h^2)}$$

So

$$y_n = \frac{7 + 2h + o(h^2)}{6 + 2h + o(h^2)}\big(1 - h + o(h^2)\big)^n + \frac{o(h^2)}{-6 - 2h + o(h^2)}\big(-5 - 3h + o(h^2)\big)^n$$

and the term $(-5)^n$ will cause the solution to diverge.

3 Stiff Problems

In some ODEs, the step size taken by an adaptive method is forced to be unreasonably small even in regions where the solution curve is smooth. In these cases, it takes a large amount of steps to go through a short time interval.

These types of equations are called **stiff ODEs**.

Example 3.0.1 (Van der Pol equation)

Given the Van der Pol equation

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0 \qquad (\text{with } x(0) = 2)$$

the larger the constant μ, the stiffer is the problem.

Trying to solve it using an explicit adaptive stepsize method like Matlab's `ode45` yields

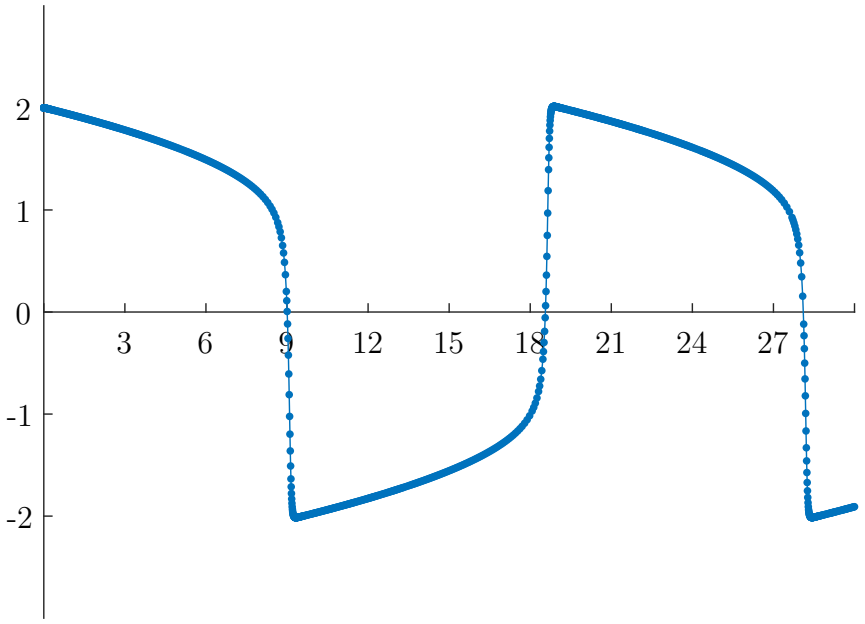


Figure 3.1: Van der Pol equation solution with ode45 ($\mu = 10$)

With 873 steps needed, and a minimum stepsize of $2.5119 \cdot 10^{-5}$.

Now, using an implicit method like Matlab's `ode15s`, we have

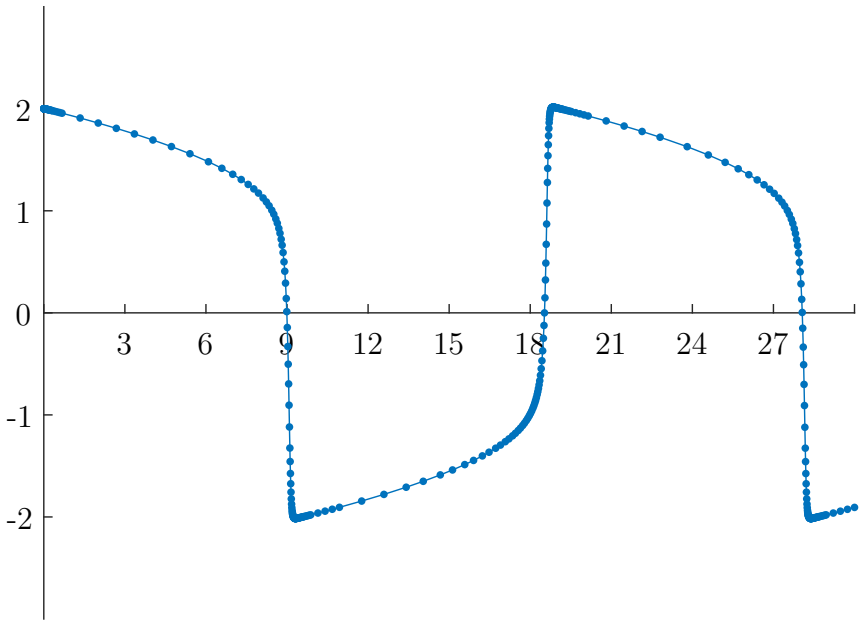


Figure 3.2: Van der Pol equation solution with ode15s ($\mu = 10$)

Which clearly uses less steps to pass through the stiff areas (a total of 326 with minimum stepsize 0.00014607).

If we where to solve it with a larger μ, for example $\mu = 1000$, the number of steps needed using `ode45` is 5.495.393 which is too much compared to the 586 needed with `ode15s`.

Definition. The set of values of the stepsize h such that $\lim_{h\to 0} y_n = 0$ is called the **absolute stability region** (Ω).

Let's see what happens with a general linear multistep method applied to a linear system:

Our method is

$$\underbrace{\sum_{j=0}^{k} \alpha_j y_{n+j}}_{\rho} = h \underbrace{\sum_{j=0}^{k} \beta_j f_{n+j}}_{\sigma}$$

We apply it to

$$y' = Ay$$

where we'll assume that the eigenvalues of A ($\lambda_1, \ldots, \lambda_n$) are all different (so that it's diagonalizable) and have negative real parts.

$y = e^{Ax}$ is the fundamental solution, and if $y(0) = y_0$, the solution is

$$y(x) = e^{Ax} y_0$$

with

$$\lim_{x\to\infty} e^{Ax} = \begin{pmatrix} 0 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 0 \end{pmatrix}$$

Now, with the system $y' = Ay$, our general method looks like

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j A y_{n+j}$$

Let's rewrite it:

$$\sum_{j=0}^{k} (\alpha_j I - h\beta_j A) y_{n+j} = 0$$

$$\Updownarrow$$

$$\sum_{j=0}^{k} \left(\alpha_j I - h\beta_j \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} \right) y_{n+j} = 0$$

$$\Updownarrow$$

$$\sum_{j=0}^{k} (\alpha_j I - h\beta_j \lambda_i) y_{n+j} = 0 \qquad (\forall i = 1, \ldots, n)$$

A finite difference equation looks like

$$a_k y_{n+k} + a_{k-1} y_{n+k-1} + \ldots + a_0 y_n = 0$$

Which has roots $r_1, \ldots, r_k$. So the solution is:

$$y_n = c_1 r_1^n + c_2 r_2^n + \ldots + c_k r_k^n \xrightarrow[\text{if } Re(r_k)<0]{} 0$$

Proposition 3.0.1. Given

$$\hat{h} = h\lambda_i, \quad \rho(z) = \sum \alpha_j z^j, \quad \sigma(z) = \sum \beta_j z^j$$

the method is absolutely stable for values of $\hat{h}$ such that the roots of $\rho(z) - \hat{h}\sigma(z) = 0$ have negative real parts.

Let's see the one dimensional case with Euler's method:

Example 3.0.2 (Stability region with Euler's method)

$$\left.\begin{aligned} y_{n+1} &= y_n + hf(y_n) \\ f(y) &= \lambda y \end{aligned}\right\} \rightsquigarrow y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda) y_n$$

$$\implies y_n = y_0 c (1 + h\lambda)^n \xrightarrow[\text{if } |1+\lambda h|<1]{} 0$$

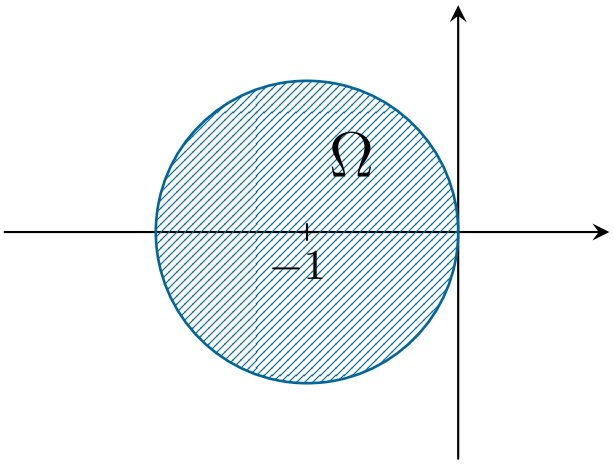


Figure 3.3: For all eigenvalues, $\hat{h}$ must be in Ω (stability region)

Particular example

$$\begin{cases} y_1' = -2y_1 + y_2 + 2\sin t \\ y_2' = 998y_1 - 999y_2 + 999(\cos t - \sin t) \end{cases}$$

This is a stiff system, with

$$J = \begin{pmatrix} -2 & 1 \\ 998 & -999 \end{pmatrix} \qquad \text{Eigenvalues}\,(J) = \{-1000, -1\}$$

So the stepsize h needs to verify

$$|1 + h(-1000)| < 1$$

$$\implies h \leq \frac{1}{500}$$

Let's do it now with backwards Euler:

Example 3.0.3 (Stability region with backwards Euler)

$$y_{n+1} = y_n + \hat{h}y_{n+1} \rightsquigarrow (1 - \hat{h})y_{n+1} = y_n \rightsquigarrow y_{n+1} = \frac{1}{1 - \hat{h}} y_n$$

$$\implies \left\| 1 - \hat{h} \right\| = \left\| \hat{h} - 1 \right\| > 1$$

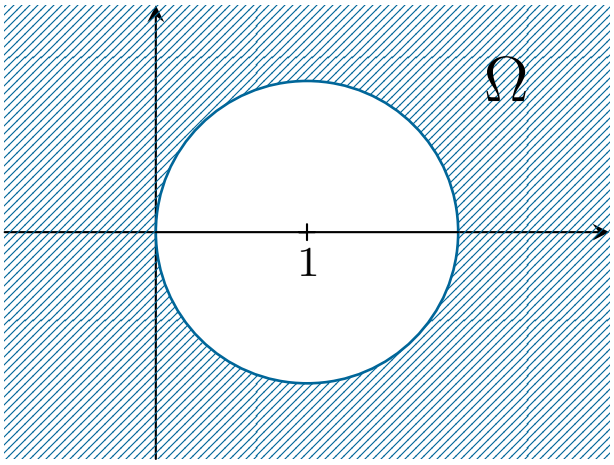


Figure 3.4: Stability region Ω with backwards Euler

Remark. If all eigenvalues are < 1, we can take any h, as all $\hat{h}$ are < 0.

Example 3.0.4 (Dekker-Verwer p.7-8)

PARTIAL DIFFERENTIAL EQUATIONS

4 Partial Differential Equations. Generalities on their solution

4.1 Finite differences

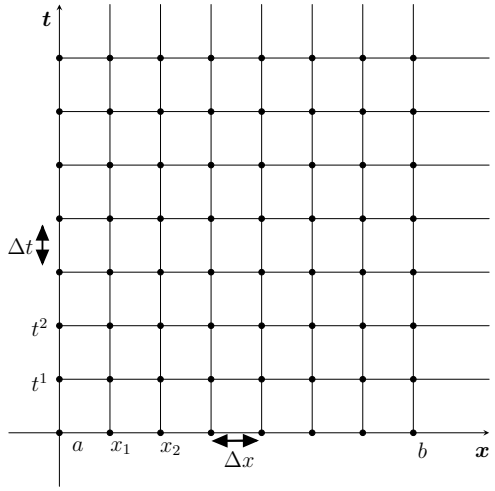
Example 4.1.1 (1D Heat equation)

Our problem is:

$$\begin{cases} u_t - u_{xx} = f \\ u(x,0) = u_0(x) & \longleftarrow \text{Initial condition (IC)} \\ \left.\begin{array}{l} u(a,t) = u_a \\ u(b,t) = u_b \end{array}\right\} & \longleftarrow \text{Boundary conditions (BC)} \end{cases}$$

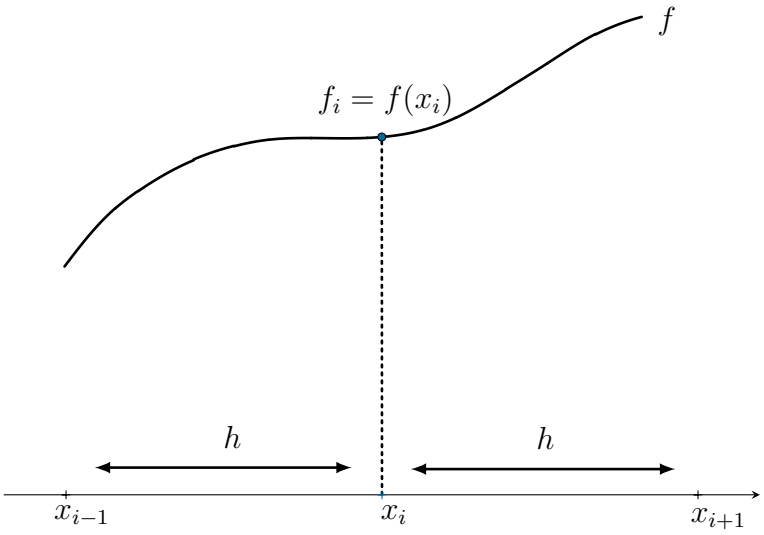
With $t \geq 0$, $x \in [a,b]$

We discretize x and t:



Idea: We'll impose $U_t(x_i,t^n) - U_{xx}(x_i,t^n) = f(x_i,t^n)$

4.1.1 Numerical derivatives



We'll use Taylor's expansion on f to find an approximation for the derivatives:

(1) $f_{i+1} = f_i + hf_i' + \frac{h^2}{2}f_i'' + \frac{h^3}{3!}f_i''' + \dots$

(2) $f_{i-1} = f_i - hf_i' + \frac{h^2}{2}f_i'' - \frac{h^3}{3!}f_i''' + \dots$

Approximation of the first derivative:

- Forward differences (1)
$$f_i' = \frac{f_{i+1} - f_i}{h} + \mathcal{O}(h)$$
- Backwards differences (2)
$$f_i' = \frac{f_i - f_{i-1}}{h} + \mathcal{O}(h)$$
- Centered differences (1)-(2)
$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} + \mathcal{O}(h^2)$$

Remark. With centered differences, it converges faster, but sometimes the other two methods are preferable for ease in computations.

Approximation of the second derivative (1)+(2)

$$f_i'' = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + \mathcal{O}(h^2)$$

There are higher order approximations, but these are the standard, and are more than enough.

Now, we'll present some notation to differentiate between the problem we want to solve and the numerical problem.

Definition.

The **continuous problem** is $\begin{cases} \mathcal{L}(u) = f & \text{in } \Omega,\ t > 0 \\ \mathcal{B}(u) = q & \text{in } \partial\Omega, t > 0 \\ u(x,0) = u_0(x) \end{cases}$

Where $\mathcal{L}$ is any differential operator.

It's what we want to solve.

The **discrete problem** is $\begin{cases} L(u) + \tau = f \\ B(u) + \overline{\tau} = q \\ u(x,0) = u_0(x) \end{cases}$

The **numerical problem** is $\begin{cases} L(U) = f \\ B(U) = q \\ U(x,0) = u_0(x) \end{cases}$

Let's go back to our 1D heat equation:

The continuous problem is

$$\begin{cases} u_t - u_{xx} = f \quad t > 0, x \in (a,b) \\ u(a,t) = u_a \\ u(b,t) = u_b \\ u(x,0) = u_0(x) \end{cases}$$

Notation: $u(x_i, t^n) = u_i^n$
$x_i = a + i\Delta x \quad i = 0, \dots N \quad x_0 = a,\ x_N = b \qquad$ (sometimes $\Delta x = h$).
$t^n = n\Delta t\, n \geq 0$

We have the following explicit method:

4.1.2 Forward Time Centered Space method (FTCS)

- Approximate u_t using forward differences (FT)
- Approximate u_{xx} using centered differences (CD)

Our discretized problem is

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \mathcal{O}(\Delta t) - \frac{u_{i+1}^n - u_i^n + u_{i-1}^n}{\Delta x^2} + \mathcal{O}(\Delta x^2) = f_i^n$$

We define $\tau = \mathcal{O}(\Delta t, \Delta x^2)$

Neglecting τ, the numerical problem we want to solve is

$$\begin{cases} \dfrac{U_i^{n+1} - U_i^n}{\Delta t} - \dfrac{U_{i+1}^n - U_i^n + U_{i-1}^n}{\Delta x^2} = f_i^n \\ U_0^{n+1} = u_a(t^{n+1}) \\ U_N^{n+1} = u_b(t^{n+1}) \\ U_i^0 = u_0(x_i) \end{cases}$$

Now we define r as

$$r = \frac{\Delta t}{\Delta x^2}$$

So

$$U_i^{n+1} = rU_{i-1}^n + (1-2r)U_i^n + rU_{i+1}^n + \Delta t f_i^n \qquad i = 1, \dots, N-1$$

For $i = 1$,

$$U_1^{n+1} = (1-2r)U_1^n + rU_2^n + \Delta t f_1^n + ru_a^n$$

And for $i = N-1$,

$$U_{N-1}^{n+1} = rU_{N-2}^n + (1-2r)U_{N-1}^n + \Delta t f_{N-1}^n + ru_b^n$$

Thus,

$$\overline{U^{n+1}} = \underset{\sim}{B}\overline{U^n} + \overline{F} + \overline{G}$$

Where

$$\overline{U^p} = \left(U_1^p, U_2^p, \dots, U_{N-1}^p\right)^T \qquad \overline{F} = \Delta t \left(f_1^n, f_2^n, \dots, f_{N-1}^n\right)^T \qquad \overline{G} = (ru_a^n, 0, \dots, 0, ru_b^n)^T$$

$$\underset{\sim}{B} = \begin{pmatrix} 1-2r & r & & & \\ r & 1-2r & r & & \\ & \ddots & \ddots & \ddots & \\ & & r & 1-2r & r \\ & & & r & 1-2r \end{pmatrix}$$

...

5 Numerical Solution of PDEs with the Finite Elements Method

5.1 From the strong form to the weak form

Given the following problem in **strong form**:

$$\begin{cases} \nabla \cdot \left(\underset{\sim}{A} \nabla u \right) = -s & \text{in } \Omega \\ u = u_D & \text{on } \Gamma_D \\ \underset{\sim}{A} \frac{\partial u}{\partial n} = q & \text{on } \Gamma_N \end{cases}$$

Where

$$\Gamma_D \cup \Gamma_N = \partial\Omega$$
$$\Gamma_D \cap \Gamma_N = \varnothing$$

s is the source term (it's a function), but for the moment we'll assume that both s and $\underset{\sim}{A}$ don't depend on u.

We'll derive the equivalent weak form of the problem using **weighted residuals**:

$$\int_\Omega \omega \nabla \cdot \left(\underset{\sim}{A} \nabla u \right) \, d\Omega = -\int_\Omega \omega s \, d\Omega \qquad \forall \omega \text{ such that } \omega = 0 \text{ on } \Gamma_D$$

Where ω is the **test function** (or **weighting function**).

We integrate by parts:

$$-\int_\Omega \nabla\omega \cdot \left(\underset{\sim}{A} \nabla u \right) \, d\Omega + \int_{\partial\Omega} \omega \underset{\sim}{A} \nabla u \cdot \bar{n} \, d\Gamma = -\int_\Omega \omega s \, d\Omega$$

Let's have a look at the term on $\partial\Omega$

$$\int_{\partial\Omega} \omega \underset{\sim}{A} \nabla u \cdot \bar{n} \, d\Gamma = \int_{\Gamma_D} \cancelto{0 \text{ (by def)}}{\omega} \underset{\sim}{A} \nabla u \cdot \bar{n} \, d\Gamma + \int_{\Gamma_N} \omega \underbrace{\underset{\sim}{A} \nabla u \cdot \bar{n}}_{q} \, d\Gamma = \int_{\Gamma_N} \omega q \, d\Gamma$$

And with that, we get

Weak form

We have to find u such that $u = u_D$ on Γ_D and

$$\int_\Omega \nabla\omega \cdot \left(\underset{\sim}{A} \nabla u \right) \, d\Omega = \int_\Omega \omega s \, d\Omega + \int_{\Gamma_N} \omega q \, d\Gamma \qquad \forall \omega \text{ such that } \omega = 0 \text{ on } \Gamma_D$$

So

$$u \in \mathcal{H}^1_{\Gamma_D}(\Omega) = \left\{ u \in \mathcal{H}^1(\Omega) \,\middle|\, u = u_D \text{ on } \Gamma_D \right\}$$

and

$$\omega \in \mathcal{H}^1_{\Gamma_D,0}(\Omega) = \left\{ u \in \mathcal{H}^1(\Omega) \,\middle|\, u = 0 \text{ on } \Gamma_D \right\}$$

Where

$$\mathcal{H}^1(\Omega) = \left\{ f \in L^2(\Omega) \,\middle|\, \frac{\partial f}{\partial x_i} \in L^2(\Omega) \quad \forall i \right\} \qquad L^2(\Omega) = \left\{ f \,\middle|\, \int_\Omega f^2 \, d\Omega < \infty \right\}$$

Now that we have the weak form, what we do is discretize the domain.

5.2 Discretisation

The mesh is defined by a set of points and triangles like:

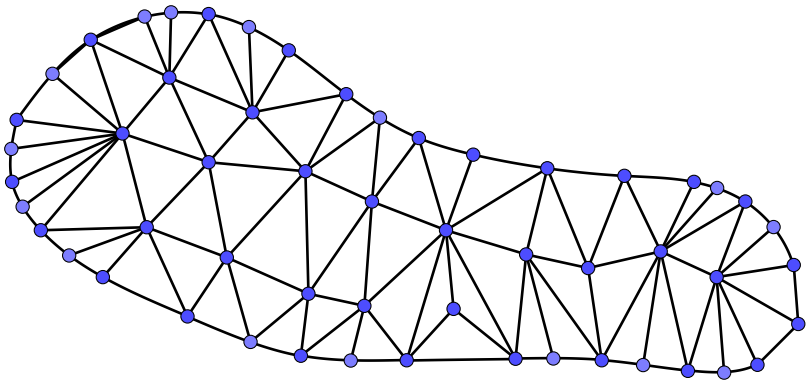


Figure 5.1: Mesh of a domain Ω

We want the nodal values of the solution noted u_i $(i = 1, \ldots, n_{\text{nodes}})$.

We'll approximate the solution u by:

$$u \simeq \sum_i N_i(x) u_i$$

Definition. We define the **shape function** $N_i(x)$ as

$$N_i(x_j) = \delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ otherwise} \end{cases}$$

where x_i is a node on the mesh.

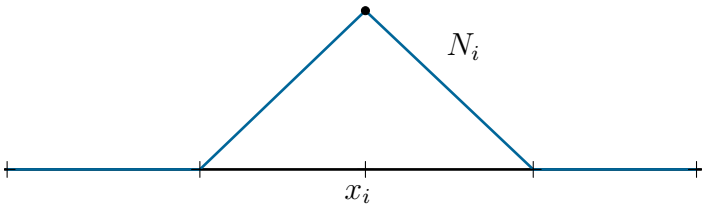


Figure 5.2: N_i are piecewise polynomials with compact support

Previously, we had to choose ω as the weighting function. The common approach is with the *Galerkin approximation*:

$$\omega = N_i \qquad i = 1, \ldots, n_{\text{nodes}}$$

For every ω chosen, we have an equation, so we choose as many ω's as unknowns.

Then

$$\int_\Omega \nabla N_i \cdot \underset{\sim}{A} \left(\sum_j \nabla N_j u_j \right) \, d\Omega = \int_\Omega N_i s \, d\Omega + \int_{\Gamma_N} N_i q \, d\Gamma$$

$$\sum_j \underbrace{\int_\Omega \left(\nabla N_i \cdot \underset{\sim}{A} \nabla N_j \right)}_{K_{ij}} d\Omega \; u_j = \underbrace{\int_\Omega N_i s \, d\Omega + \int_{\Gamma_N} N_i q \, d\Gamma}_{f_i}$$

So we end up with the system

$$\sum_j K_{ij} u_j = f_i \qquad i = 1, \ldots, n_{\text{nodes}}$$

and in matrix form

$$\underset{\sim}{K} \bar{u} = \bar{f}$$

$\underset{\sim}{K}$ is singular, so we need to impose the Dirichlet conditions:

$$\sum_{j=1}^{n_{\text{nodes}}} N_j(x) u_j = \sum_{j \in U} N_j(x) u_j + \sum_{j \in D} N_j(x) u_j \qquad (u_j = u_D(x_j))$$

where $\begin{cases} j \in D \text{ if } x_j \in \Gamma_D \\ j \in U \text{ if } x_j \notin \Gamma_D \end{cases}$

Example 5.2.1
We'll work with the following 1D problem:

$$\begin{cases} -u_{xx} = f \\ u(0) = 0,\ u(1) = 1 \end{cases}$$

with

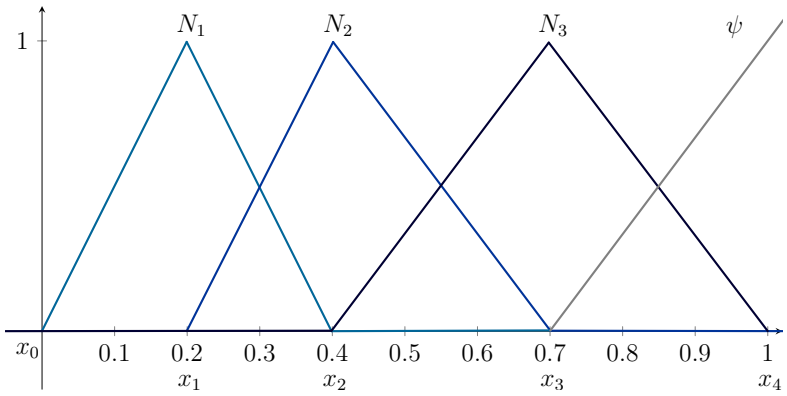


Figure 5.3

First we'll write the weak form to approximate $u(x)$ by functions in $\mathcal{H}_0^1(0,1)$ to get

$$u(x) \simeq u_1 N_1(x) + u_2 N_2(x) + u_3 N_3(x) + \psi(x) \tag{5.1}$$

Let's proceed

$$-\int_0^1 u_{xx}\omega\, dx = \int_0^1 f\omega\, dx$$

We integrate the left hand side by parts:

$$-\int_0^1 u_{xx}\omega\, dx = -\left[\omega u_x\right]_0^1 + \int_0^1 u_x\omega_x\, dx$$

$\omega \in \mathcal{H}_0^1(0,1) \implies \omega(0) = \omega(1) = 0 \implies \omega u_x\big|_0^1 = 0$

<u>Weak form</u>

We have to find u such that $u(0) = 0$, $u(1) = 1$ and

$$\int_0^1 u_x\omega_x\, dx = \int_0^1 f\omega\, dx \qquad \forall \omega \in \mathcal{H}_0^1(0,1)$$

Let's rewrite $\int_0^1 u_x \omega_x \, dx$ using our approximation of u (5.1):

$$\int_0^1 u_x \omega_x \, dx = \int_0^1 (u_1 N_1'(x) + u_2 N_2'(x) + u_3 N_3'(x) + \psi'(x)) \cdot N_i'(x) \, dx \qquad i = 1, 2, 3$$

Thus

$$\sum_{j=1}^{3} u_j \underbrace{\int_0^1 N_j'(x) \cdot N_i'(x) \, dx}_{K_{ij}} = \underbrace{\int_0^1 f N_i(x) \, dx - \int_0^1 \psi'(x) N_i'(x)}_{f_i} \qquad i = 1, 2, 3$$

Let's find the explicit form of the $N_i's$ with the discretisation in Figure 5.3:

$$N_1(x) = \begin{cases} 5x & 0 \le x \le 0.2 \\ -5(x - 0.4) & 0.2 \le x \le 0.4 \\ 0 & \text{otherwise} \end{cases}$$

$$N_2(x) = \begin{cases} 5(x - 0.2) & 0.2 \le x \le 0.4 \\ -\frac{10}{3}(x - 0.7) & 0.4 \le x \le 0.7 \\ 0 & \text{otherwise} \end{cases}$$

$$N_3(x) = \begin{cases} \frac{10}{3}(x - 0.4) & 0.4 \le x \le 0.7 \\ -\frac{10}{3}(x - 1) & 0.7 \le x \le 1 \\ 0 & \text{otherwise} \end{cases}$$

Now we can find the K_{ij}'s:

$$K_{11} = \int_0^1 N_1'(x) \cdot N_1'(x) \, dx = \int_0^{0.2} 5 \cdot 5 \, dx + \int_{0.2}^{0.4} (-5) \cdot (-5) \, dx = 10$$

$$K_{12} = K_{21} = \int_{0.2}^{0.4} (-5) \cdot 5 \, dx = -5$$

$$K_{13} = K_{31} = 0$$

$$K_{22} = \int_{0.2}^{0.4} 5 \cdot 5 \, dx + \int_{0.4}^{0.7} \frac{-10}{3} \cdot \frac{-10}{3} \, dx = \frac{25}{3}$$

$$K_{23} = K_{32} = \int_{0.4}^{0.7} \frac{-10}{3} \cdot \frac{10}{3} \, dx = -\frac{10}{3}$$

$$K_{33} = \int_{0.4}^{0.7} \frac{10}{3} \cdot \frac{10}{3} \, dx + \int_{0.7}^{1} \frac{-10}{3} \cdot \frac{-10}{3} \, dx = \frac{20}{3}$$

So

$$\underset{\sim}{K} = \begin{pmatrix} 10 & -5 & 0 \\ -5 & \frac{25}{3} & -\frac{10}{3} \\ 0 & -\frac{10}{3} & \frac{20}{3} \end{pmatrix}$$

and the system we have to solve is:

$$\begin{pmatrix} 10 & -5 & 0 \\ -5 & \frac{25}{3} & -\frac{10}{3} \\ 0 & -\frac{10}{3} & \frac{20}{3} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$

6 Introduction to Boundary Value Problems

7 Quality Control of Solutions