L32: Ordinary Differential Equations

Part 2: New Time-Stepping Method; Stability; Order

Lucas A. J. Bastien

E7 Spring 2017, University of California at Berkeley

April 12, 2017

Version: release

Announcements

Lab 11 is due on April 14 at 12 pm (noon)

Lab 11 is significantly shorter than most previous labs

Use the opportunity to:

- Get a lot of points on lab 11!
- Work on your project!

Project Beta Test is due on April 14 at 12 pm (noon)

Today:

- Ordinary differential equations – Part 2 (Chapter 19)

Friday

- Ordinary differential equations – Part 3
  (Review parts 1 and 2 before lecture)

Next week:

- Searching and sorting (no required reading)

Growth mindset

Consider thinking about your studies with a "**growth mindset**":

Growth mindset

Consider thinking about your studies with a "**growth mindset**":

- If you cannot solve a question the first time, it does **not** mean that you cannot solve it the second time or the third time that you try

Growth mindset

Consider thinking about your studies with a "**growth mindset**":

- If you cannot solve a question the first time, it does **not** mean that you cannot solve it the second time or the third time that you try
- Learning is an **iterative process**

Growth mindset

Consider thinking about your studies with a "**growth mindset**":

- If you cannot solve a question the first time, it does **not** mean that you cannot solve it the second time or the third time that you try
- Learning is an **iterative process**
- If you wish you had done better on a lab assignment or on an exam, think about
  - What did you learn?
  - What to change for next time?

Growth mindset

Consider thinking about your studies with a "**growth mindset**":

- If you cannot solve a question the first time, it does **not** mean that you cannot solve it the second time or the third time that you try
- Learning is an **iterative process**
- If you wish you had done better on a lab assignment or on an exam, think about
  - What did you learn?
  - What to change for next time?

As an educator, I will also think about this semester with a growth mindset:

- What did I learn?
- What would I change if I taught this class again?

Notation for first-order ordinary differential equations

General notation used here:

$$y' = F(t, y)$$

where y is the unknown function of time t

Notation for first-order ordinary differential equations

General notation used here:

$$y' = F(t, y)$$

where y is the unknown function of time t

Examples:

$$y' = 3y - 5 \quad \rightarrow \quad F(t, y) = 3y - 5$$

$$y' = 3ty - 5\cos(t) \quad \rightarrow \quad F(t, y) = 3ty - 5\cos(t)$$

Notation for first-order ordinary differential equations

General notation used here:

$$y' = F(t, y)$$

where y is the unknown function of time t

Examples:

$$y' = 3y - 5 \quad \rightarrow \quad F(t, y) = 3y - 5$$

$$y' = 3ty - 5\cos(t) \quad \rightarrow \quad F(t, y) = 3ty - 5\cos(t)$$

When applying the numerical methods seen this week, $F(t_i, y(t_i))$ gives the slope at time t_i

Numerical methods for "solving" initial value problems

We are learning methods to "solve" **first-order initial value problems**

Notation:

Generic first-order initial value problem (unknown is y, a function of t):

$$\begin{aligned} y' &= F(t, y) && \text{(ODE)} \\ y(t = t_0) &= y_0 && \text{(Initial condition)} \end{aligned}$$

Numerical methods for "solving" initial value problems

We are learning methods to "solve" **first-order initial value problems**

Notation:
Generic first-order initial value problem (unknown is y, a function of t):

$$y' = F(t, y) \qquad \text{(ODE)}$$
$$y(t = t_0) = y_0 \qquad \text{(Initial condition)}$$

General approach: estimate the function's value at discrete small intervals (*i.e.* estimate the function at points $t_0, t_1, t_2, \ldots$), starting from the known value (*i.e.* the initial condition), **assuming that the slope is constant over each interval:**

$$y(t_{i+1}) = y(t_i) + \text{slope} \times \Delta t_i$$

where $\Delta t_i = (t_{i+1} - t_i)$ is the "spacing" or "time step"

Different methods: different approximations for the slope

Summary of the methods learned so far

Summary of the methods learned so far

- **Euler explicit:**
  Use slope calculated at the beginning of the time step

Summary of the methods learned so far

- **Euler explicit:**
  Use slope calculated at the beginning of the time step
- **Euler implicit:**
  Use slope calculated at the end of the time step

Summary of the methods learned so far

- **Euler explicit:**
  Use slope calculated at the beginning of the time step
- **Euler implicit:**
  Use slope calculated at the end of the time step
- **Predictor-corrector methods:**
  Use one or more **“predictor” steps** to get a better approximation of the slope for the time step. Use the **“corrector” step** to calculate $y(t_{i+1})$

Summary of the methods learned so far

- **Euler explicit:**
  Use slope calculated at the beginning of the time step

- **Euler implicit:**
  Use slope calculated at the end of the time step

- **Predictor-corrector methods:**
  Use one or more **“predictor” steps** to get a better approximation of the slope for the time step. Use the **“corrector” step** to calculate $y(t_{i+1})$

  - **Midpoint method:** estimate the slope at the midpoint of the time step

Summary of the methods learned so far

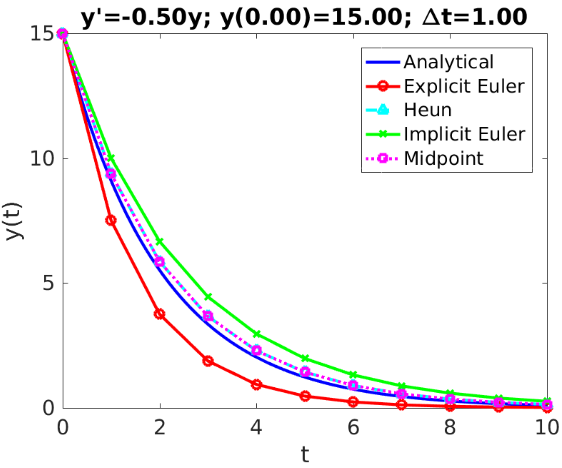
- **Euler explicit:**
  Use slope calculated at the beginning of the time step

- **Euler implicit:**
  Use slope calculated at the end of the time step

- **Predictor-corrector methods:**
  Use one or more **"predictor" steps** to get a better approximation of the slope for the time step. Use the **"corrector" step** to calculate $y(t_{i+1})$

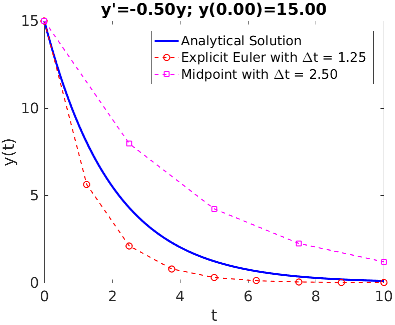
  - **Midpoint method:** estimate the slope at the midpoint of the time step
  - **Heun's method:** estimate the slope as the average of the slope at the beginning of the time step and of the slope at the end of the time step

Summary of the methods learned so far: example

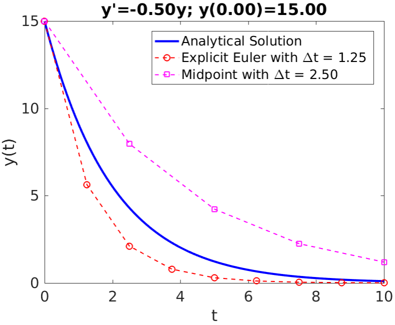


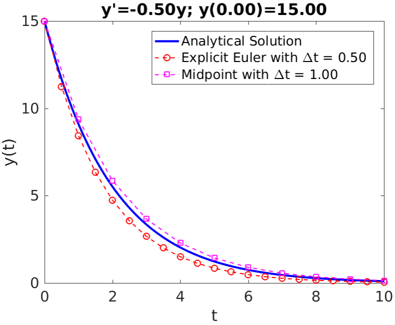
See script `ode_numerical_solving.m` (Explicit Euler and Heun's methods only, the other methods are left for you to do as an exercise)

Using the midpoint method with time step Δt and using the explicit Euler method with time step $\Delta t/2$ yield different results



Using the midpoint method with time step Δt and using the explicit Euler method with time step $\Delta t/2$ yield different results





Fourth-order Runge-Kutta method

The fourth-order Runge-Kutta method is a predictor-corrector method

Predictor step: **estimate the slope**

$$
\begin{aligned}
k_1 &= F(t_i, y(t_i)) \\
k_2 &= F(t_i + \Delta t_i/2, y(t_i) + k_1 \Delta t_i/2) \\
k_3 &= F(t_i + \Delta t_i/2, y(t_i) + k_2 \Delta t_i/2) \\
k_4 &= F(t_i + \Delta t_i, y(t_i) + k_3 \Delta t_i)
\end{aligned}
$$

$$
\text{slope} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad \text{(weighted-average)}
$$

Fourth-order Runge-Kutta method

The fourth-order Runge-Kutta method is a predictor-corrector method

Predictor step: **estimate the slope**

$$
\begin{aligned}
k_1 &= F(t_i, y(t_i)) \\
k_2 &= F(t_i + \Delta t_i/2, y(t_i) + k_1 \Delta t_i/2) \\
k_3 &= F(t_i + \Delta t_i/2, y(t_i) + k_2 \Delta t_i/2) \\
k_4 &= F(t_i + \Delta t_i, y(t_i) + k_3 \Delta t_i)
\end{aligned}
$$

$$
\text{slope} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad \text{(weighted-average)}
$$

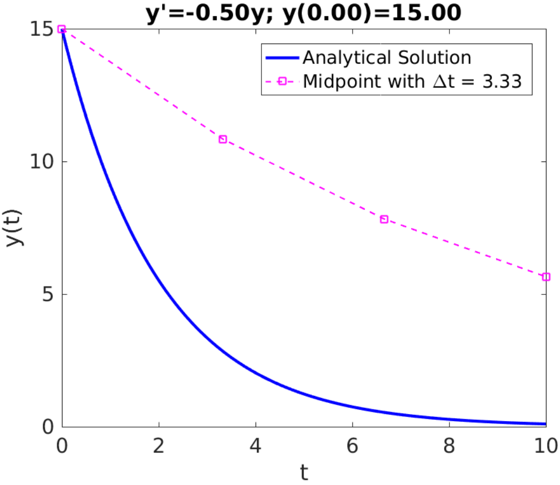
Corrector step: **calculate $y(t_{i+1})$ using this slope**

$$
y(t_{i+1}) = y(t_i) + \text{slope} \times \Delta t_i
$$

Reducing the size of the time step

In general, the numerical approximate "solution" becomes more accurate as the time step becomes smaller

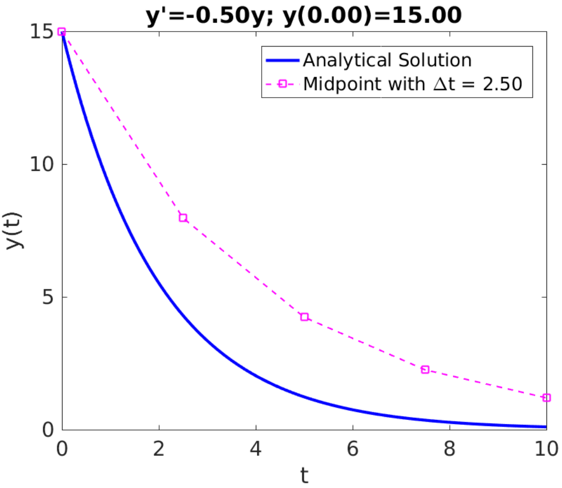
For example with the midpoint method:



Reducing the size of the time step

In general, the numerical approximate "solution" becomes more accurate as the time step becomes smaller

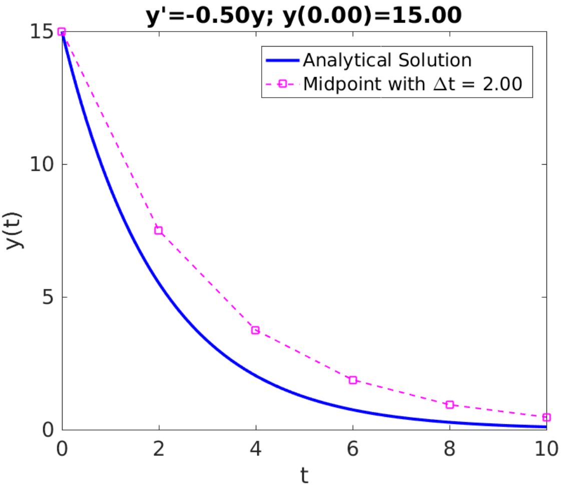
For example with the midpoint method:



Reducing the size of the time step

In general, the numerical approximate "solution" becomes more accurate as the time step becomes smaller

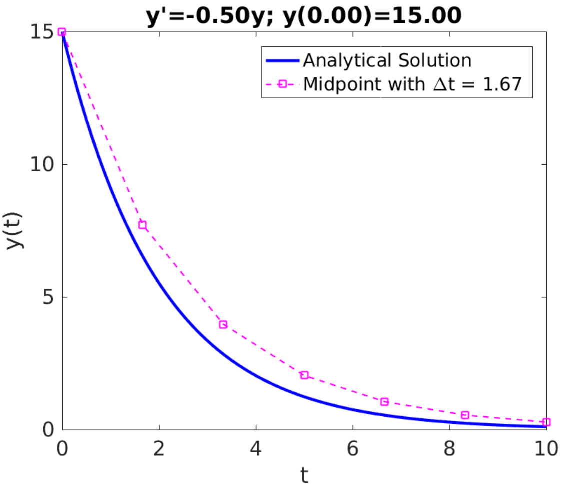
For example with the midpoint method:



Reducing the size of the time step

In general, the numerical approximate "solution" becomes more accurate as the time step becomes smaller

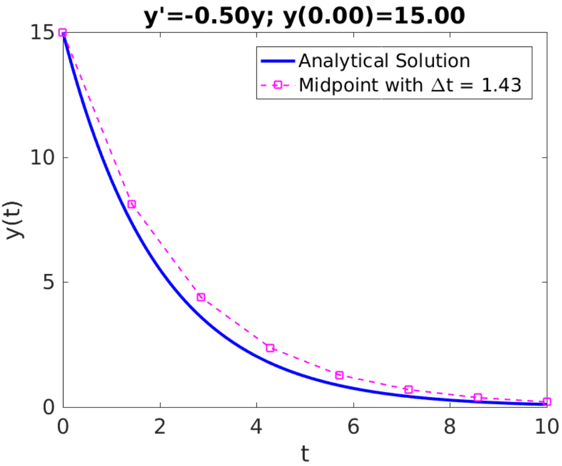
For example with the midpoint method:



Reducing the size of the time step

In general, the numerical approximate "solution" becomes more accurate as the time step becomes smaller

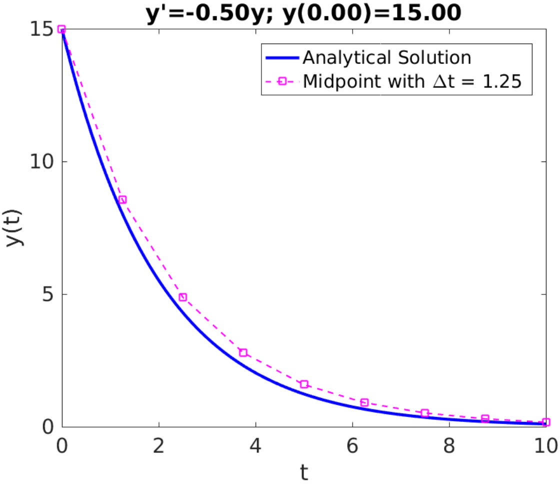
For example with the midpoint method:



Reducing the size of the time step

In general, the numerical approximate "solution" becomes more accurate as the time step becomes smaller

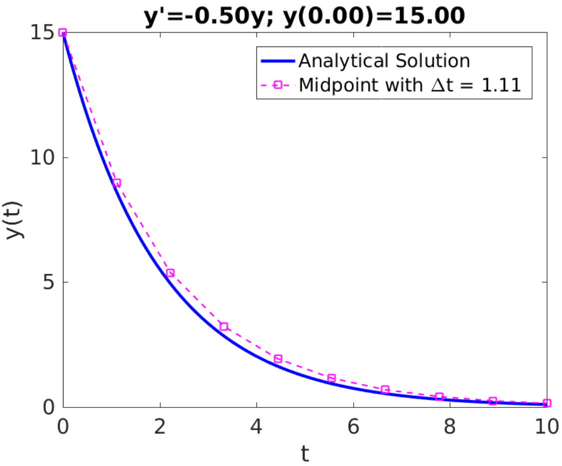
For example with the midpoint method:



Reducing the size of the time step

In general, the numerical approximate “solution” becomes more accurate as the time step becomes smaller

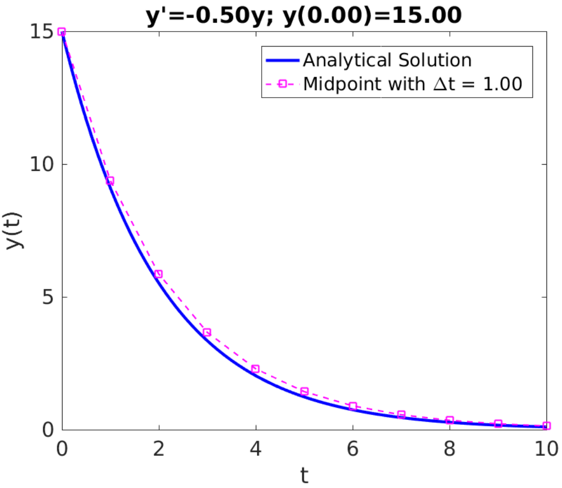
For example with the midpoint method:



Reducing the size of the time step

In general, the numerical approximate "solution" becomes more accurate as the time step becomes smaller

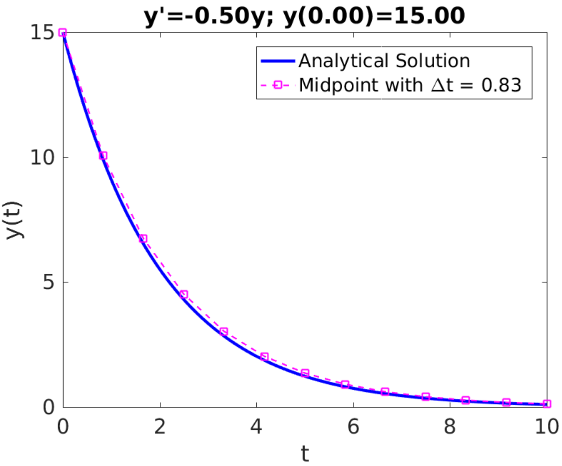
For example with the midpoint method:



Reducing the size of the time step

In general, the numerical approximate “solution” becomes more accurate as the time step becomes smaller

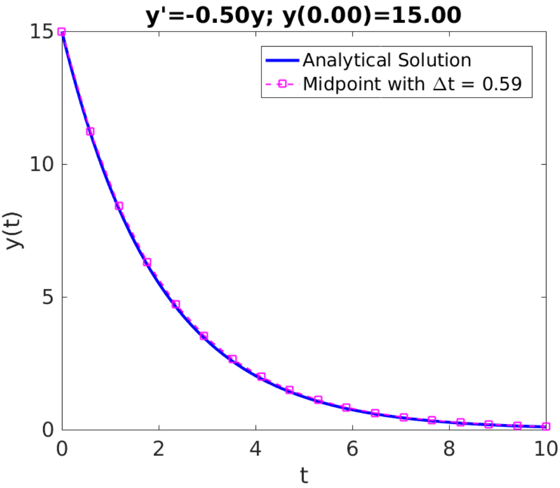
For example with the midpoint method:



Reducing the size of the time step

In general, the numerical approximate "solution" becomes more accurate as the time step becomes smaller

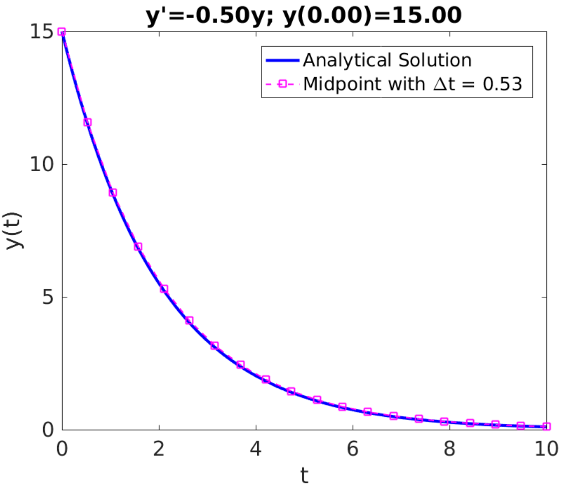
For example with the midpoint method:



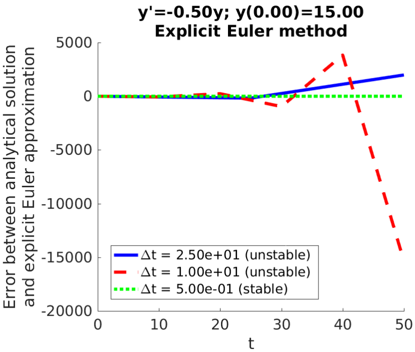
Reducing the size of the time step

In general, the numerical approximate "solution" becomes more accurate as the time step becomes smaller

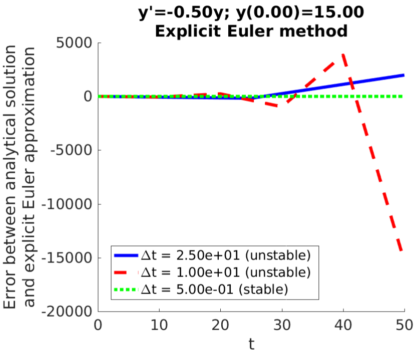
For example with the midpoint method:



The **stability** of a numerical method for "solving" ODEs describes whether the error between the numerical approximate "solution" and the analytical solution grows unbounded (*i.e.* bigger and bigger) as we keep calculating the "solution" at more and more times steps

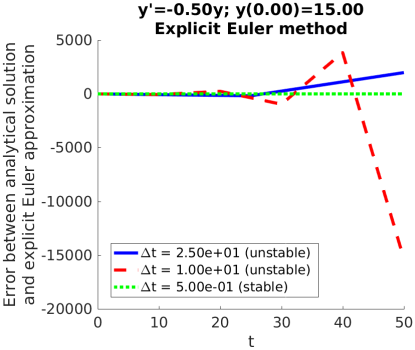


The **stability** of a numerical method for "solving" ODEs describes whether the error between the numerical approximate "solution" and the analytical solution grows unbounded (*i.e.* bigger and bigger) as we keep calculating the "solution" at more and more times steps



Stability may depend on the size of the time steps, as shown in the example on the left, where the explicit Euler method is used to solve $y' = -0.5y$ with $y(0) = 15$

The **stability** of a numerical method for "solving" ODEs describes whether the error between the numerical approximate "solution" and the analytical solution grows unbounded (*i.e.* bigger and bigger) as we keep calculating the "solution" at more and more times steps



Stability may depend on the size of the time steps, as shown in the example on the left, where the explicit Euler method is used to solve $y' = -0.5y$ with $y(0) = 15$

Some methods are always stable

Study of stability: example

Example with the explicit Euler method applied to exponential decay:

$$\text{ODE:} \qquad N' = -kN$$

Study of stability: example

Example with the explicit Euler method applied to exponential decay:

$$
\begin{aligned}
\text{ODE:} \qquad & N' = -kN \\
\text{Explicit Euler:} \qquad & N(t_{i+1}) \approx N(t_i) - kN(t_i)\Delta t \\
& = N(t_i)(1 - k\Delta t)
\end{aligned}
$$

Study of stability: example

Example with the explicit Euler method applied to exponential decay:

$$
\begin{aligned}
&\text{ODE:} && N' = -kN \\
&\text{Explicit Euler:} && N(t_{i+1}) \approx N(t_i) - kN(t_i)\Delta t \\
& && = N(t_i)(1 - k\Delta t)
\end{aligned}
$$

The value of $N(t_i)$ is an approximation since it has been calculated with the explicit Euler method. Call ϵ_i the error on $N(t_i)$ compared to the exact value $N_{\text{exact}}(t_i)$

$$
\begin{aligned}
N(t_{i+1}) &\approx N(t_i)(1 - k\Delta t) \\
&= (N_{\text{exact}}(t_i) + \epsilon_i)(1 - k\Delta t)
\end{aligned}
$$

Study of stability: example

Example with the explicit Euler method applied to exponential decay:

$$\begin{aligned}
\text{ODE:} \qquad && N' &= -kN \\
\text{Explicit Euler:} \qquad && N(t_{i+1}) &\approx N(t_i) - kN(t_i)\Delta t \\
&& &= N(t_i)(1 - k\Delta t)
\end{aligned}$$

The value of $N(t_i)$ is an approximation since it has been calculated with the explicit Euler method. Call ϵ_i the error on $N(t_i)$ compared to the exact value $N_{\text{exact}}(t_i)$

$$\begin{aligned}
N(t_{i+1}) &\approx N(t_i)(1 - k\Delta t) \\
&= (N_{\text{exact}}(t_i) + \epsilon_i)(1 - k\Delta t)
\end{aligned}$$

- If $|1 - k\Delta t| > 1$: the error is amplified, the method is unstable
- If $|1 - k\Delta t| < 1$: the error is damped, the method is stable

Study of stability: example

Example with the explicit Euler method applied to exponential decay:

$$\begin{aligned} \text{ODE:} \qquad & N' = -kN \\ \text{Explicit Euler:} \qquad & N(t_{i+1}) \approx N(t_i) - kN(t_i)\Delta t \\ & \qquad\qquad = N(t_i)(1 - k\Delta t) \end{aligned}$$

The value of $N(t_i)$ is an approximation since it has been calculated with the explicit Euler method. Call ϵ_i the error on $N(t_i)$ compared to the exact value $N_{\text{exact}}(t_i)$

$$\begin{aligned} N(t_{i+1}) &\approx N(t_i)(1 - k\Delta t) \\ &= (N_{\text{exact}}(t_i) + \epsilon_i)(1 - k\Delta t) \end{aligned}$$

- If $|1 - k\Delta t| > 1$: the error is amplified, the method is unstable
- If $|1 - k\Delta t| < 1$: the error is damped, the method is stable

Here stability depends on the step size Δt

Order of the numerical methods for ODE solving

Example with explicit Euler method:

$$y(t_{i+1}) \approx y(t_i) + y'(t_i)\Delta t$$

Note: the explanation presented here is not a rigorous mathematical proof

Order of the numerical methods for ODE solving

Example with explicit Euler method:

$$y(t_{i+1}) \approx y(t_i) + y'(t_i)\Delta t$$

From Taylor series:

$$\begin{aligned} y(t_{i+1}) &= y(t_i) + y'(t_i)\Delta t + \frac{y''(t_i)}{2!}(\Delta t)^2 + \frac{y'''(t_i)}{3!}(\Delta t)^3 + \dots \\ &= y(t_i) + y'(t_i)\Delta t + \mathcal{O}((\Delta t)^2) \end{aligned}$$

Note: the explanation presented here is not a rigorous mathematical proof

Order of the numerical methods for ODE solving

Example with explicit Euler method:

$$y(t_{i+1}) \approx y(t_i) + y'(t_i)\Delta t$$

From Taylor series:

$$\begin{aligned} y(t_{i+1}) &= y(t_i) + y'(t_i)\Delta t + \frac{y''(t_i)}{2!}(\Delta t)^2 + \frac{y'''(t_i)}{3!}(\Delta t)^3 + \dots \\ &= y(t_i) + y'(t_i)\Delta t + \mathcal{O}((\Delta t)^2) \end{aligned}$$

The method is second-order for a single time step. When approximating the solution over an interval $[t_0, t_f]$, we take $n = (t_f - t_0)/\Delta t$ time steps. The overall order of the method is:

Note: the explanation presented here is not a rigorous mathematical proof

Order of the numerical methods for ODE solving

Example with explicit Euler method:

$$y(t_{i+1}) \approx y(t_i) + y'(t_i)\Delta t$$

From Taylor series:

$$\begin{aligned} y(t_{i+1}) &= y(t_i) + y'(t_i)\Delta t + \frac{y''(t_i)}{2!}(\Delta t)^2 + \frac{y'''(t_i)}{3!}(\Delta t)^3 + \dots \\ &= y(t_i) + y'(t_i)\Delta t + \mathcal{O}((\Delta t)^2) \end{aligned}$$

The method is second-order for a single time step. When approximating the solution over an interval $[t_0, t_f]$, we take $n = (t_f - t_0)/\Delta t$ time steps. The overall order of the method is:

$$\text{Overall order} = \sum_{i=1}^{n} \text{individual orders}$$

Note: the explanation presented here is not a rigorous mathematical proof

Order of the numerical methods for ODE solving

Example with explicit Euler method:

$$y(t_{i+1}) \approx y(t_i) + y'(t_i)\Delta t$$

From Taylor series:

$$\begin{aligned} y(t_{i+1}) &= y(t_i) + y'(t_i)\Delta t + \frac{y''(t_i)}{2!}(\Delta t)^2 + \frac{y'''(t_i)}{3!}(\Delta t)^3 + \dots \\ &= y(t_i) + y'(t_i)\Delta t + \mathcal{O}((\Delta t)^2) \end{aligned}$$

The method is second-order for a single time step. When approximating the solution over an interval $[t_0, t_f]$, we take $n = (t_f - t_0)/\Delta t$ time steps. The overall order of the method is:

$$\text{Overall order} = \sum_{i=1}^{n} \text{individual orders} = \sum_{i=1}^{n} \mathcal{O}((\Delta t)^2)$$

Note: the explanation presented here is not a rigorous mathematical proof

Order of the numerical methods for ODE solving

Example with explicit Euler method:

$$y(t_{i+1}) \approx y(t_i) + y'(t_i)\Delta t$$

From Taylor series:

$$\begin{aligned} y(t_{i+1}) &= y(t_i) + y'(t_i)\Delta t + \frac{y''(t_i)}{2!}(\Delta t)^2 + \frac{y'''(t_i)}{3!}(\Delta t)^3 + \dots \\ &= y(t_i) + y'(t_i)\Delta t + \mathcal{O}((\Delta t)^2) \end{aligned}$$

The method is second-order for a single time step. When approximating the solution over an interval $[t_0, t_f]$, we take $n = (t_f - t_0)/\Delta t$ time steps. The overall order of the method is:

$$\text{Overall order} = \sum_{i=1}^{n} \text{individual orders} = \sum_{i=1}^{n} \mathcal{O}((\Delta t)^2)$$

$$\approx n \times \mathcal{O}((\Delta t)^2)$$

Note: the explanation presented here is not a rigorous mathematical proof

Order of the numerical methods for ODE solving

Example with explicit Euler method:

$$y(t_{i+1}) \approx y(t_i) + y'(t_i)\Delta t$$

From Taylor series:

$$\begin{aligned} y(t_{i+1}) &= y(t_i) + y'(t_i)\Delta t + \frac{y''(t_i)}{2!}(\Delta t)^2 + \frac{y'''(t_i)}{3!}(\Delta t)^3 + \dots \\ &= y(t_i) + y'(t_i)\Delta t + \mathcal{O}((\Delta t)^2) \end{aligned}$$

The method is second-order for a single time step. When approximating the solution over an interval $[t_0, t_f]$, we take $n = (t_f - t_0)/\Delta t$ time steps. The overall order of the method is:

$$\begin{aligned} \text{Overall order} &= \sum_{i=1}^{n} \text{individual orders} = \sum_{i=1}^{n} \mathcal{O}((\Delta t)^2) \\ &\approx n \times \mathcal{O}((\Delta t)^2) = \frac{t_f - t_0}{\Delta t} \times \mathcal{O}((\Delta t)^2) \end{aligned}$$

Note: the explanation presented here is not a rigorous mathematical proof

Order of the numerical methods for ODE solving

Example with explicit Euler method:

$$y(t_{i+1}) \approx y(t_i) + y'(t_i)\Delta t$$

From Taylor series:

$$\begin{aligned} y(t_{i+1}) &= y(t_i) + y'(t_i)\Delta t + \frac{y''(t_i)}{2!}(\Delta t)^2 + \frac{y'''(t_i)}{3!}(\Delta t)^3 + \dots \\ &= y(t_i) + y'(t_i)\Delta t + \mathcal{O}((\Delta t)^2) \end{aligned}$$

The method is second-order for a single time step. When approximating the solution over an interval $[t_0, t_f]$, we take $n = (t_f - t_0)/\Delta t$ time steps. The overall order of the method is:

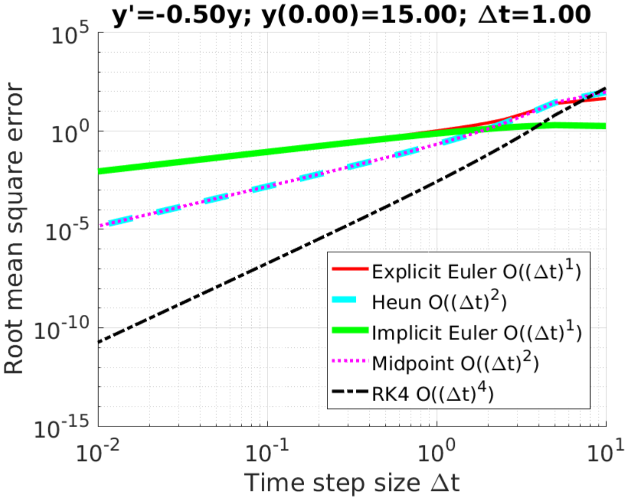
$$\begin{aligned} \text{Overall order} &= \sum_{i=1}^{n} \text{individual orders} = \sum_{i=1}^{n} \mathcal{O}((\Delta t)^2) \\ &\approx n \times \mathcal{O}((\Delta t)^2) = \frac{t_f - t_0}{\Delta t} \times \mathcal{O}((\Delta t)^2) \\ &= \mathcal{O}(\Delta t) \end{aligned}$$

Note: the explanation presented here is not a rigorous mathematical proof

Order of the numerical methods for ODE solving

Method	Order of the error at each time step	Overall order of the method
Explicit Euler	$\mathcal{O}((\Delta x)^2)$	$\mathcal{O}(\Delta x)$
Implicit Euler	$\mathcal{O}((\Delta x)^2)$	$\mathcal{O}(\Delta x)$
Midpoint rule	$\mathcal{O}((\Delta x)^3)$	$\mathcal{O}((\Delta x)^2)$
Heun's rule	$\mathcal{O}((\Delta x)^3)$	$\mathcal{O}((\Delta x)^2)$
Runge-Kutta 4	$\mathcal{O}((\Delta x)^5)$	$\mathcal{O}((\Delta x)^4)$

Order of the numerical methods for ODE solving



The slope of the line in a log-log plot indicates the order of the method