Differential Equations

- Most physical phenomena are mathematically expressed as differential equations
- Could be Ordinary Differential Equation (ODE) or Partial Differential Equations (PDE)
- For Example:
  - Radioactive decay: $\frac{dm}{dt} = -\lambda m$
  - Mass-spring-damper system: $m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$
  - Steady-state temperature: $\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} = 0$

Ordinary Differential Equations

- Only one independent variable: generally t (time-dependent), but could be x (steady-state, spatially variable). We will use t.
- The dependent variable: We will use y.
- Order of a differential equation: Highest derivative – Decay- first order, Spring- 2nd
- Degree: Power of the highest derivative.
- 1st: $\frac{d^2x}{dt^2} + a\left(\frac{dx}{dt}\right)^2 + bx = 0$ 2nd: $\left(\frac{d^2x}{dt^2}\right)^2 + a\frac{dx}{dt} + bx = 0$
- We will consider only first-degree equations.

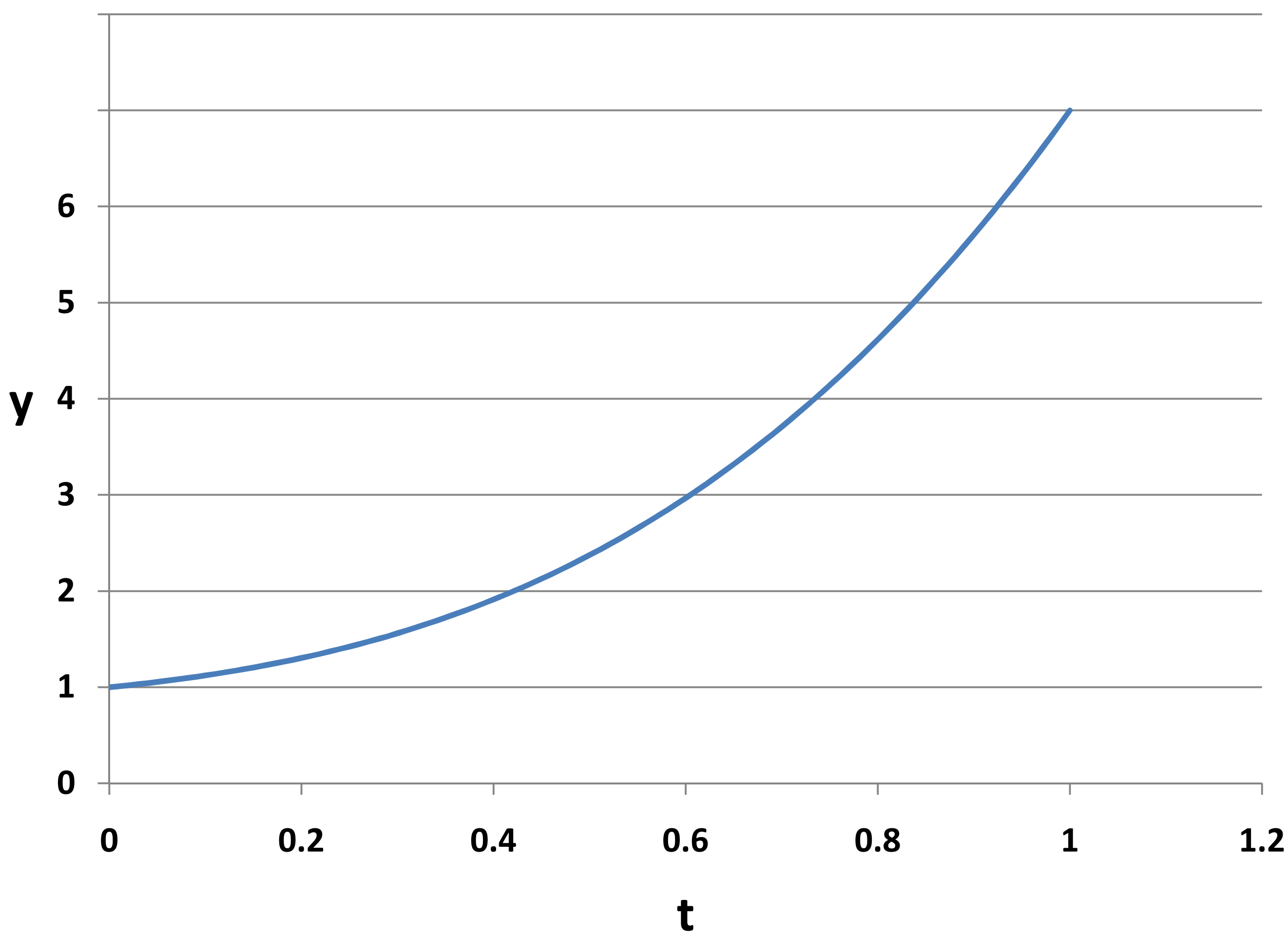
First Order ODE's

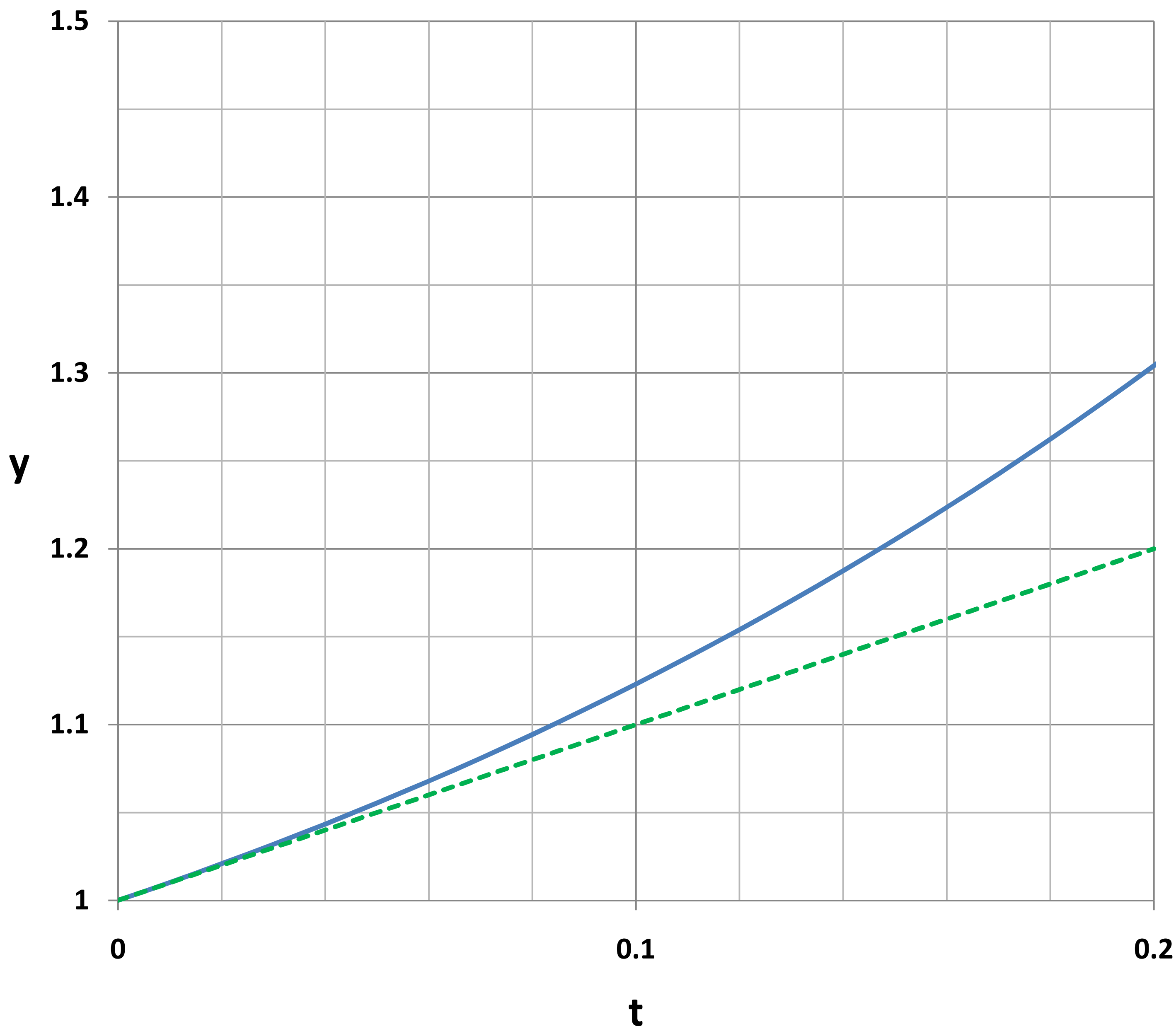
- Start with a first-order ODE: $$\frac{dy}{dt} = f(t, y)$$
- Needs one boundary/initial condition
- We take it as $y_{\text{ at } t=t_0} = y_0$
- E.g., Radioactive decay: $$\frac{dy}{dt} = -\lambda y; \;\; y_{t=0} = 1$$
- How to estimate y at all subsequent times
- Numerically, we find y at some finite points by taking time steps of Δt: we use *h* to denote Δt
- By using small *h*, we can come close to the continuous behavior

First Order ODE's

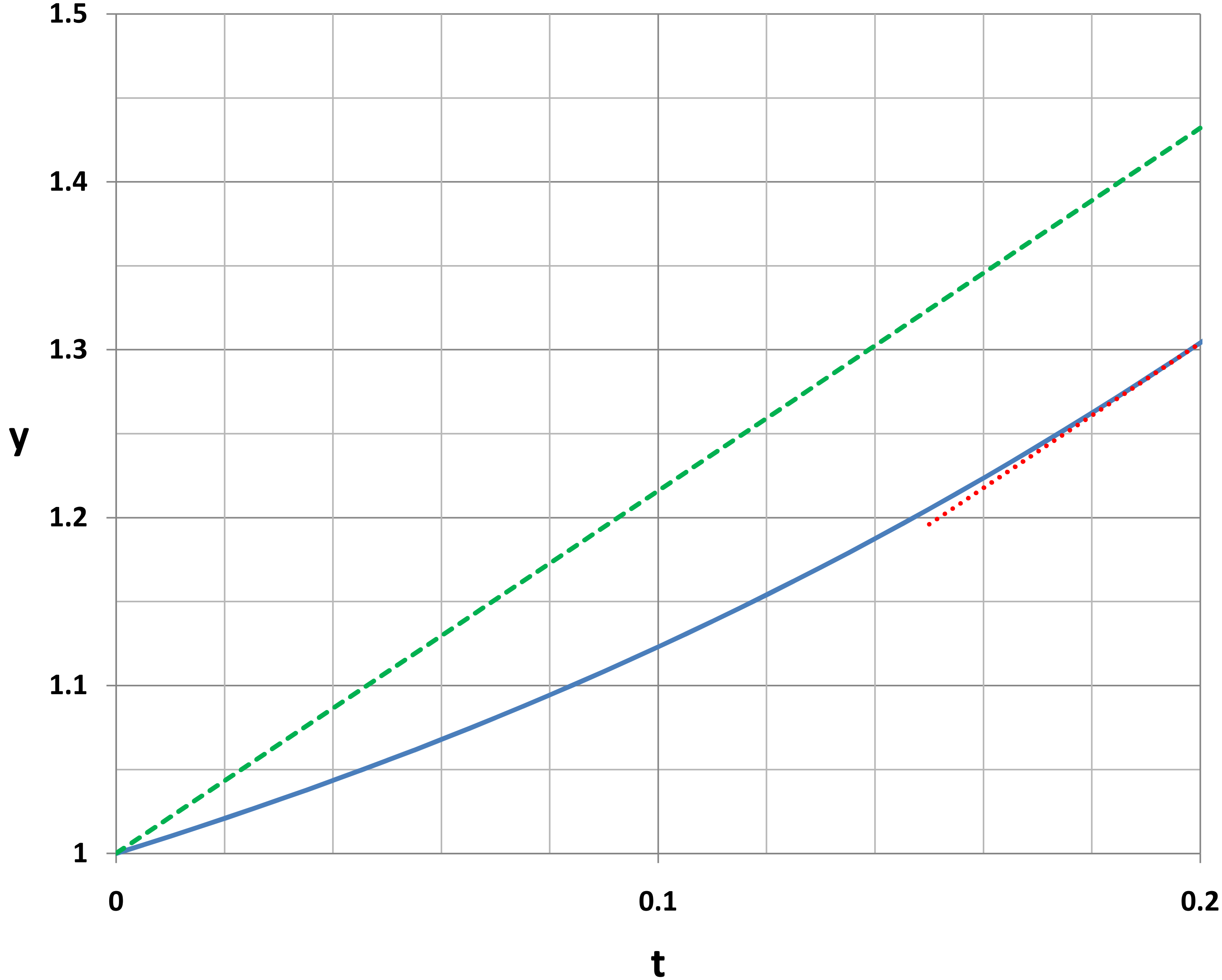
- We formulate the problem as:
  - Given: $\dfrac{dy}{dt} = f(t, y)$ and $y_{\text{ at } t=t_0} = y_0$
  - Find: $y \text{ at } t = t_0 + h$
  - Once the value of *y* is obtained at t_0+h, we take this as the "known" value and estimate the value at the next time step, and so on.
  - Note that *h* could be changed at each step, but generally it is kept constant

First Order ODE's : Graphical Representation

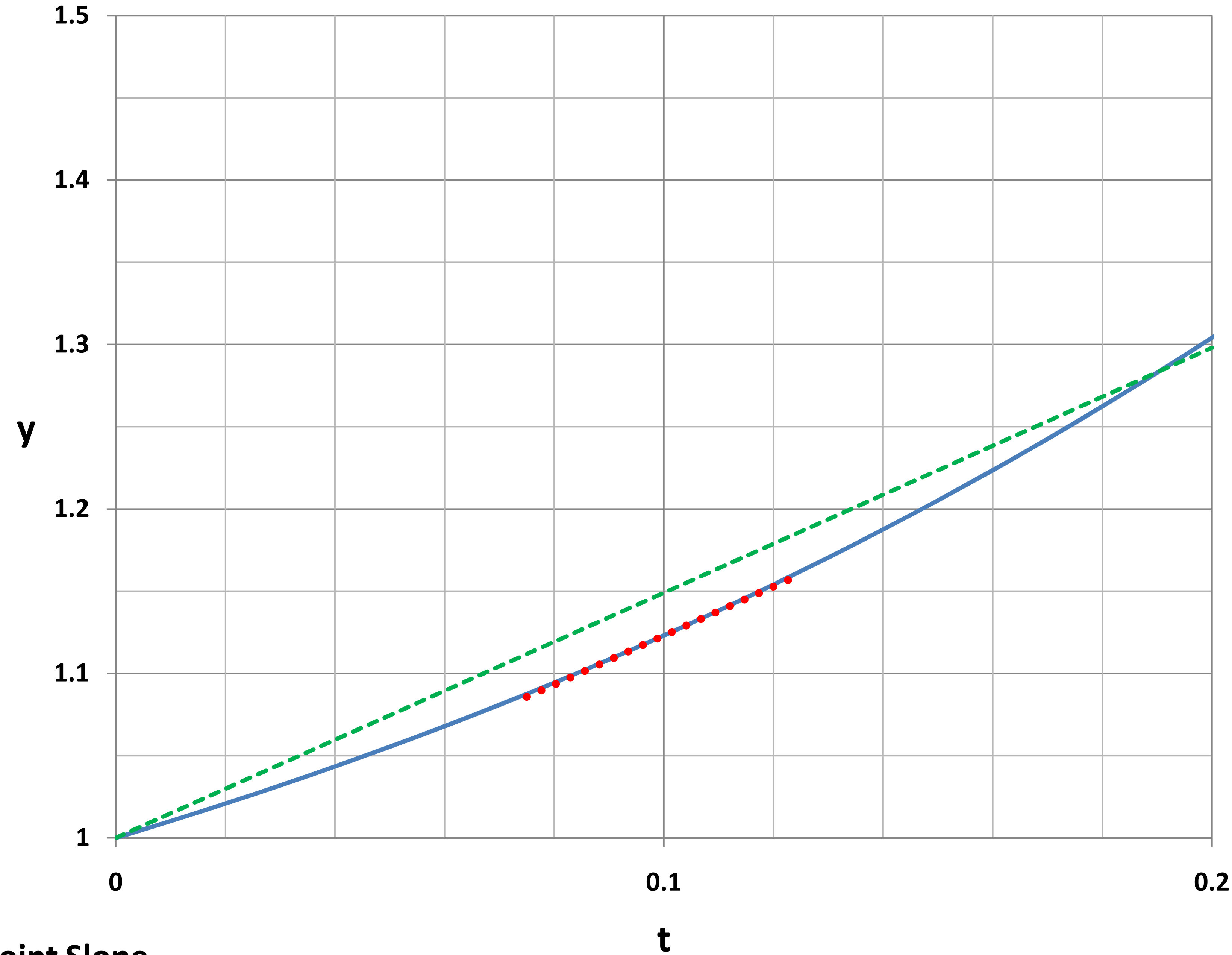




Use *h*=0.2: Forward Slope



Backward Slope



Midpoint Slope

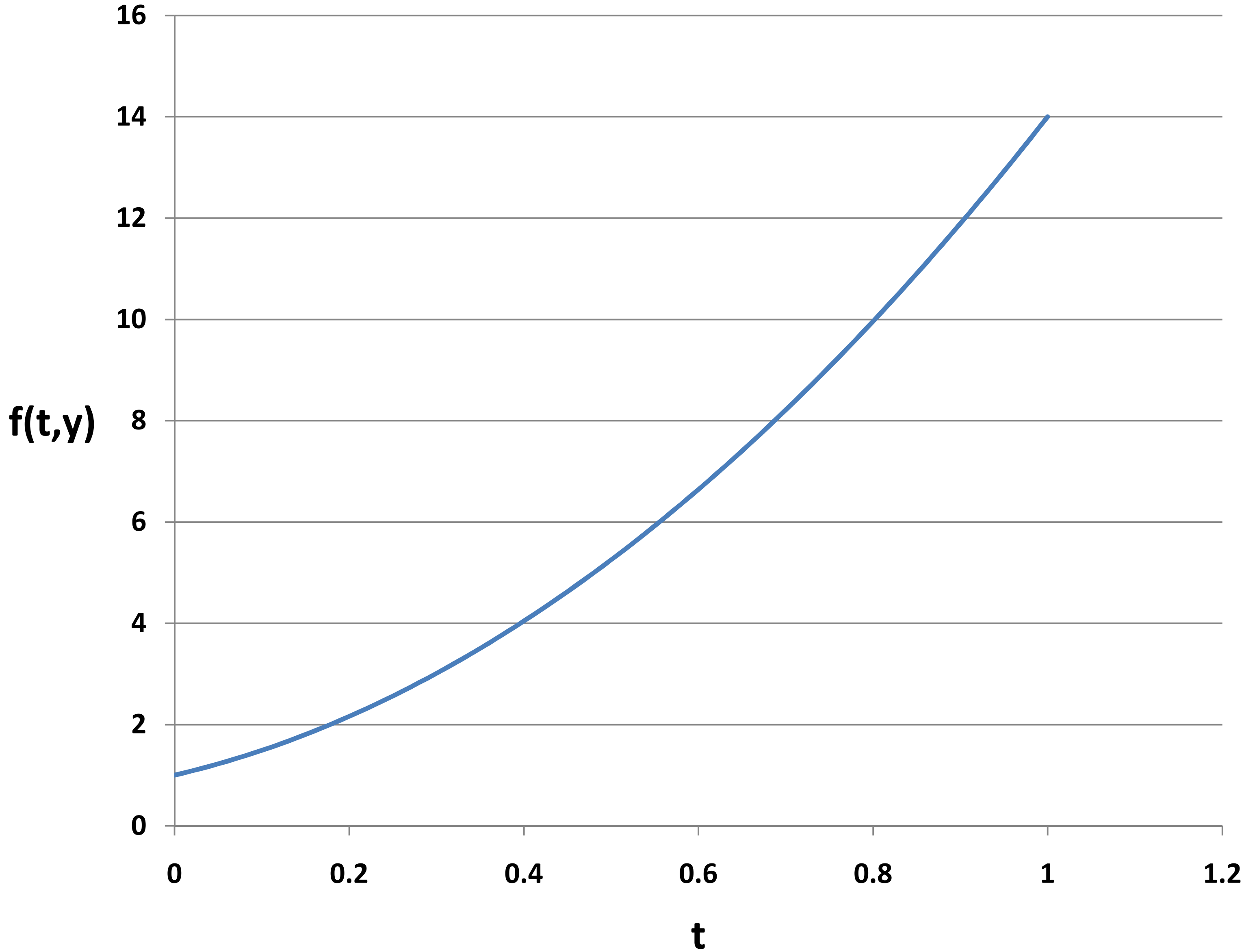
Alternative formulation: Integration

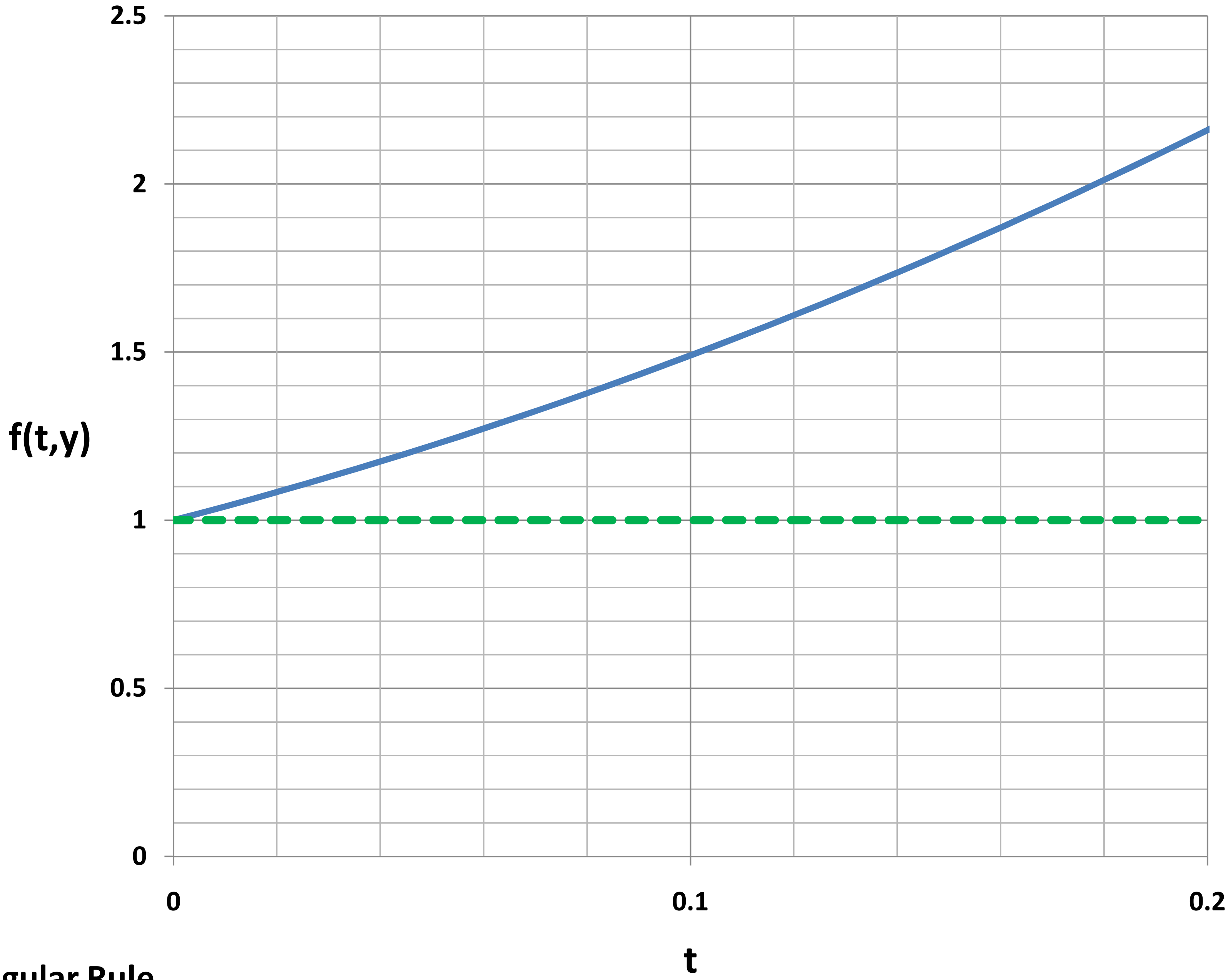
- We may re-formulate the problem as:

$$y_{t_0+h} = y_0 + \int_{t_0}^{t_0+h} f(t, y)\,dt$$

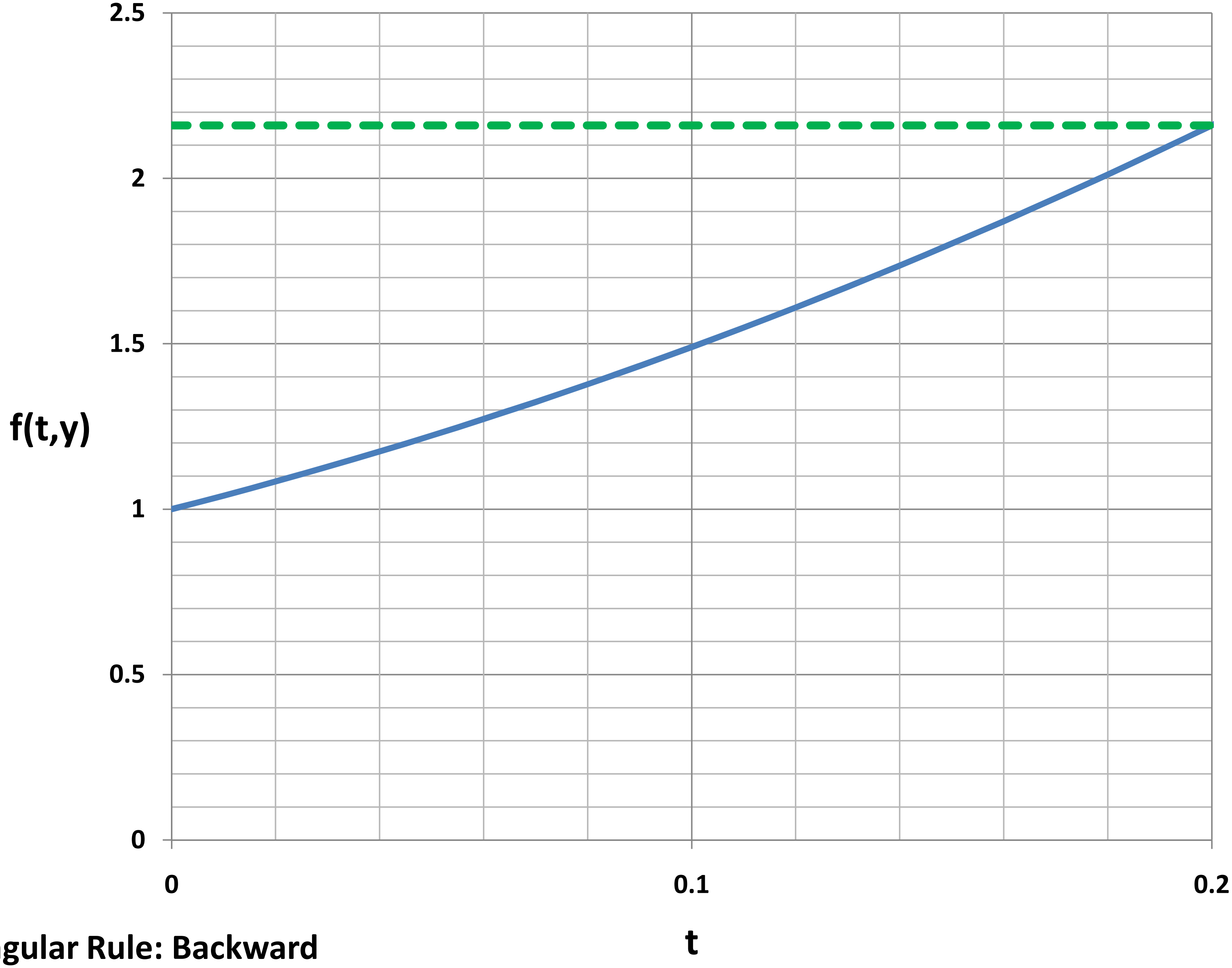
  - ➢ Numerically integrate the function

Integration: Graphical Representation

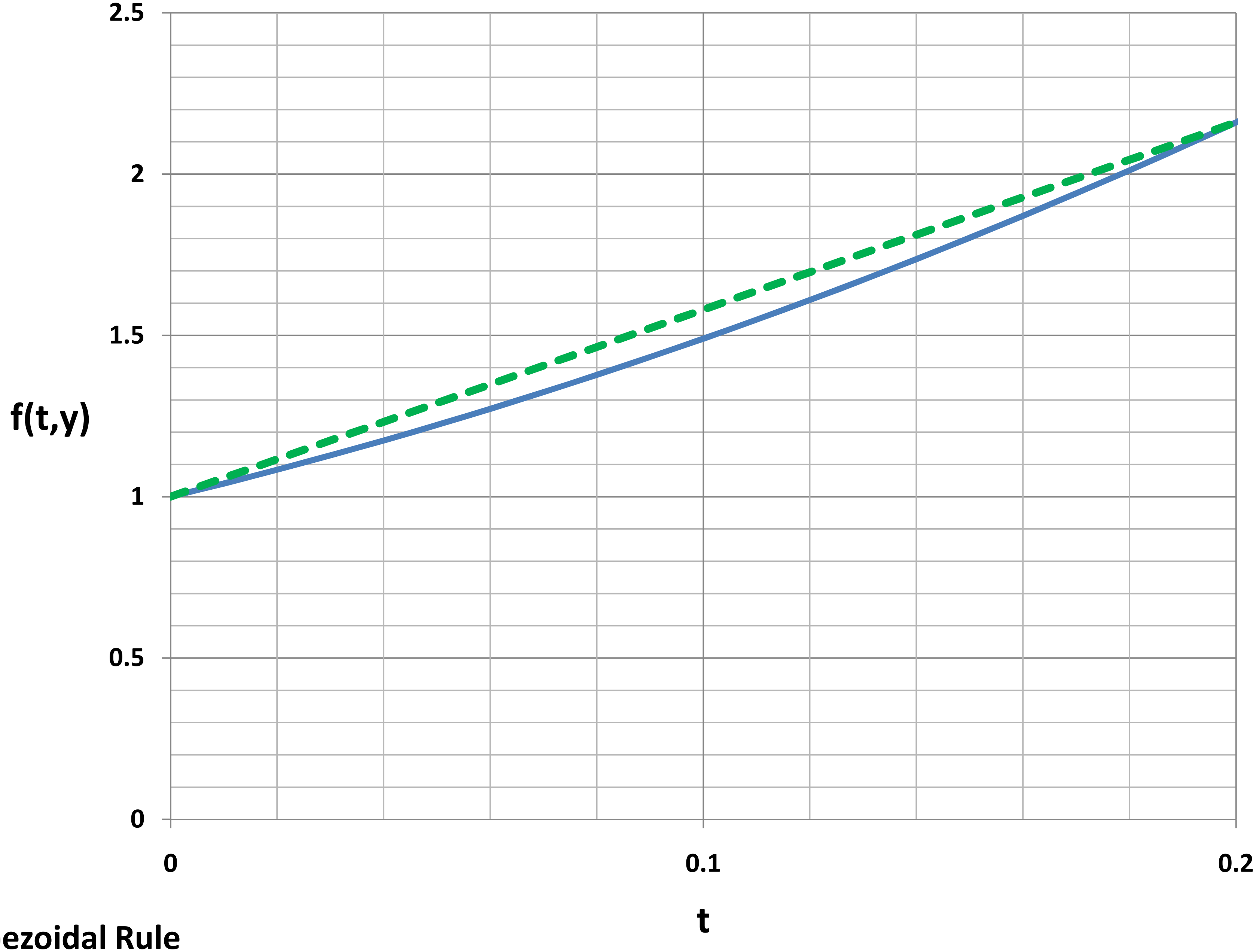




Rectangular Rule



Rectangular Rule: Backward



Trapezoidal Rule

First Order ODE's: Solution Algorithm

- Use subscript *n* for "known" point and *n*+1 for the "desired" point: given t_n, y_n, t_{n+1}, find y_{n+1}
- Euler Forward or Explicit method:

$$y_{n+1} = y_n + hf\left(t_n, y_n\right)$$

  - ➢ Slope approximated by a forward difference

$$f\left(t_n, y_n\right) = \frac{y_{n+1} - y_n}{h}$$

  - ➢ Or, integral estimated by a rectangular rule

$$\int_{t_n}^{t_{n+1}} f\left(t, y\right) dt = hf\left(t_n, y_n\right)$$

Euler Method

- Euler Backward or Implicit method:

$$y_{n+1} = y_n + hf\left(t_{n+1}, y_{n+1}\right)$$

  - ➢ Slope approximated by a backward difference

$$f\left(t_{n+1}, y_{n+1}\right) = \frac{y_{n+1} - y_n}{h}$$

  - ➢ Integral estimated by a backward rectangular rule

$$\int_{t_n}^{t_{n+1}} f\left(t, y\right) dt = hf\left(t_{n+1}, y_{n+1}\right)$$

  - ➢ Implicit: Cannot be solved directly for y_{n+1} (unless f is of a very simple form, e.g. $f=-\lambda y$)

Single- and Multi-step Methods

- Both of these methods use the slope (derivative) at a single point (*n* for the explicit and *n*+1 for the implicit): Single-step methods
- The multi-step methods use slope at more than one points. E.g., trapezoidal rule

$$\int_{t_n}^{t_{n+1}} f(t,y)dt = h\frac{f(t_n,y_n)+f(t_{n+1},y_{n+1})}{2}$$

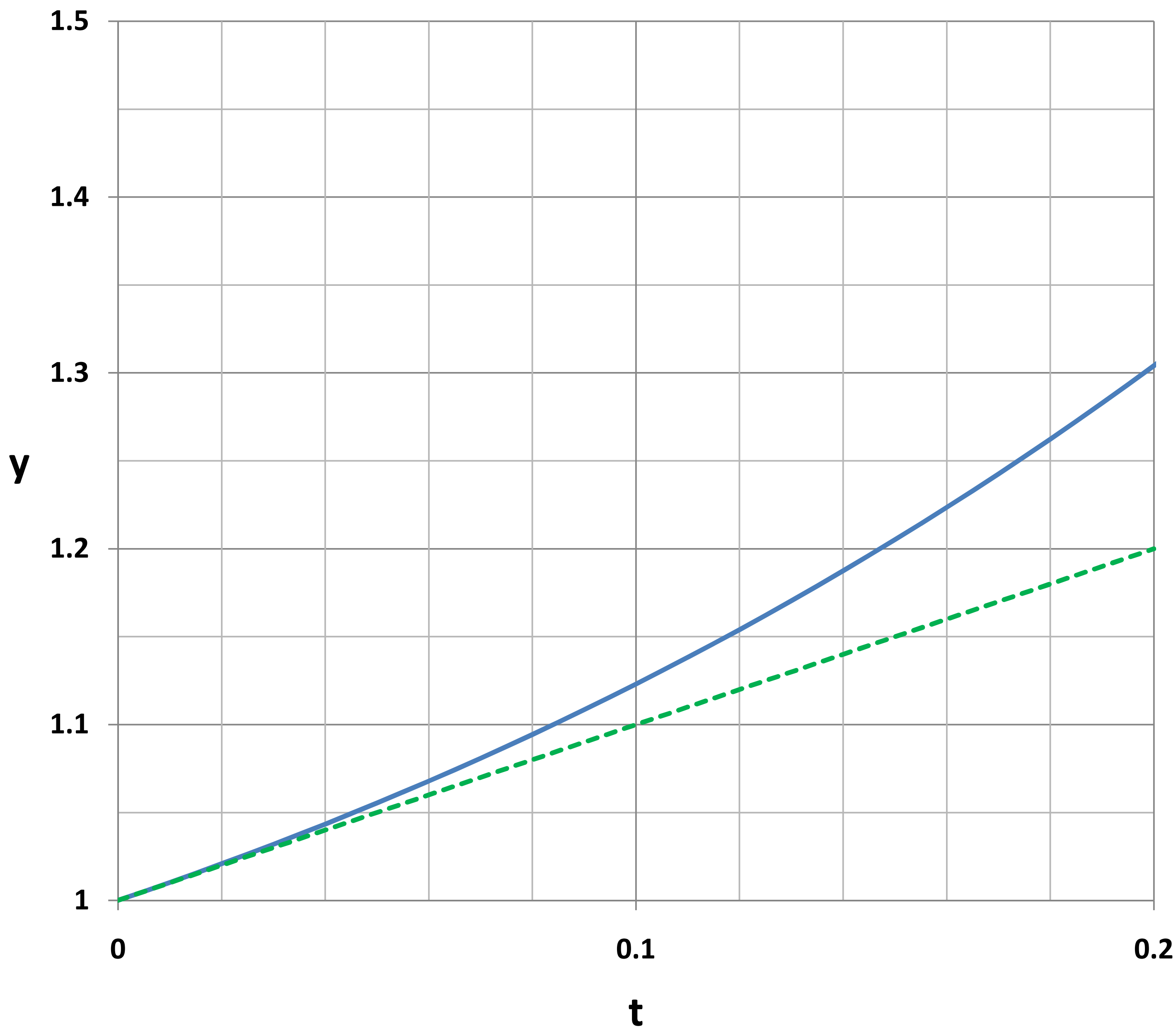
  - Resulting in the trapezoidal method or **Implicit Heun's method**:

$$y_{n+1} = y_n + h\frac{f(t_n,y_n)+f(t_{n+1},y_{n+1})}{2}$$

Single- and Multi-step Methods

- Single-step methods may be explicit or implicit, depending on which point is used:
  - If slope at *n* is used- Explicit
  - If slope at *n*+1 is used- Implicit
- Multi-step methods also may be explicit or implicit, depending on how the points are chosen:
  - If the "average" slope does not use y_{n+1}- Explicit

    E.g., $$y_{n+1} = y_n + h\frac{f(t_n, y_n) + f(t_{n+1}, y_n + hf(t_n, y_n))}{2}$$
  - Otherwise- Implicit. E.g., Trapezoidal method



Use Forward Slope to estimate “end-point” value. Then, use that estimate for trapezoidal rule

Consistency and Stability

- Consistency: The numerical approximation should represent the original equation as h->0
  - E.g., Euler Forward –

$$y_{n+1} = y_n + hf\left(t_n, y_n\right)$$

  - Taylor's series:

$$y_{n+1} = y_n + hf\left(t_n, y_n\right) + \frac{h^2}{2} f'\left(t_n, y_n\right) + \ldots$$

The method is consistent and the error in a single step (called the *Local Truncation Error*) is O(h^2).

Consistency

- Euler Backward –

$$y_{n+1} = y_n + hf\left(t_{n+1}, y_{n+1}\right)$$

- Taylor's series:

$$y_n = y_{n+1} - hf\left(t_{n+1}, y_{n+1}\right) + \frac{h^2}{2} f'\left(t_{n+1}, y_{n+1}\right) - \ldots$$

The method is consistent and the LTE (local truncation error) is O(h^2).

Both forward and backward are consistent and have same order of accuracy. Forward may not be STABLE

Stability

- Stability: The numerical solution should be bounded if the exact solution is bounded
- E.g., First-order decay: $\dfrac{dy}{dt} = -\lambda y;\ \ y_{t=0} = 1$
  - Euler Forward:

$$y_{n+1} = y_n - h\lambda y_n$$

will become unbounded if $|1 - h\lambda| > 1$, conditionally stable

  - Euler Backward:

$$y_{n+1} = y_n - h\lambda y_{n+1} \Rightarrow y_{n+1} = \frac{y_n}{1 + h\lambda}$$

will not become unbounded – unconditionally stable

- If $\dfrac{dy}{dt} = \lambda y;\ \ y_{t=0} = 1$, the exact soln is unbounded

Derivation of multi-step methods

- Given: $$\frac{dy}{dt} = f(t, y) \qquad y_{\text{ at } t=t_0} = y_0$$
  - subscript *n* is for "known" point and *n*+1 for the "desired" point: given t_n, y_n, t_{n+1}, find y_{n+1}
  - All previous points, 0,1,2…,*n*−1 are "known"
- Linear: We write the desired value, y_{n+1}, in terms of a linear combination of y_n and the "slopes"

$$y_{n+1} = y_n + h\left(\beta f_{n+1} + \sum_{i=0}^{k} \alpha_i f_{n-i}\right)$$

- Explicit if β=0, implicit otherwise. *k* =0,1,2,…,*n*