

COMS30017

COMPUTATIONAL NEUROSCIENCE

**LECTURE: INTRODUCTION TO NUMERICAL SOLUTIONS OF
DIFFERENTIAL EQUATIONS**

Dr. Rahul Gupta

xv20319@bristol.ac.uk

Intended learning outcomes

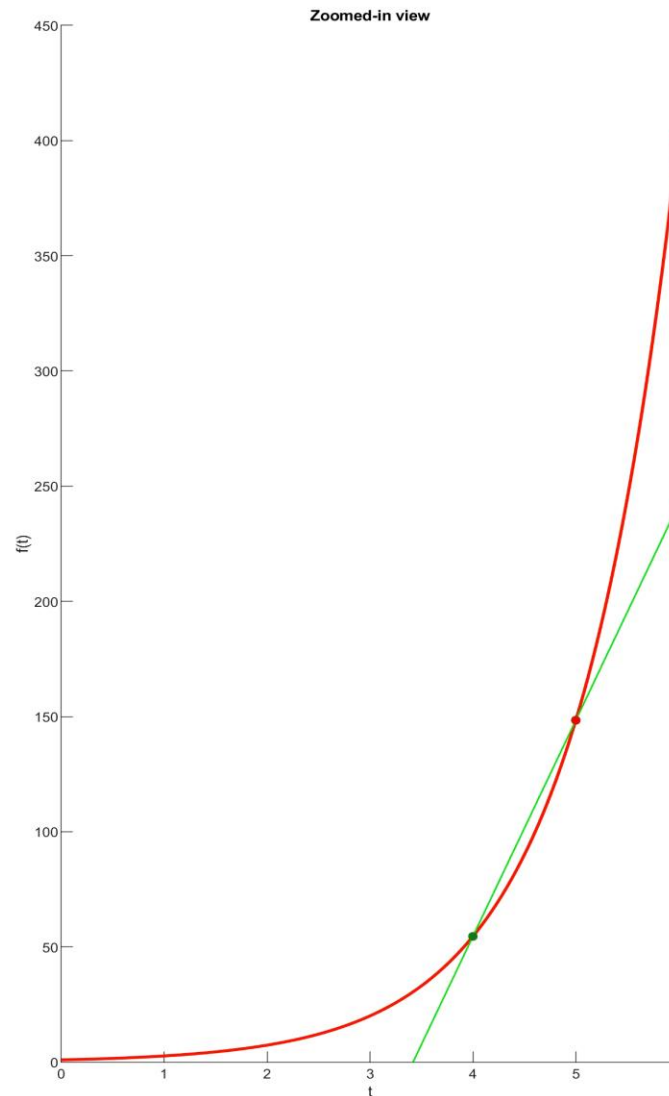
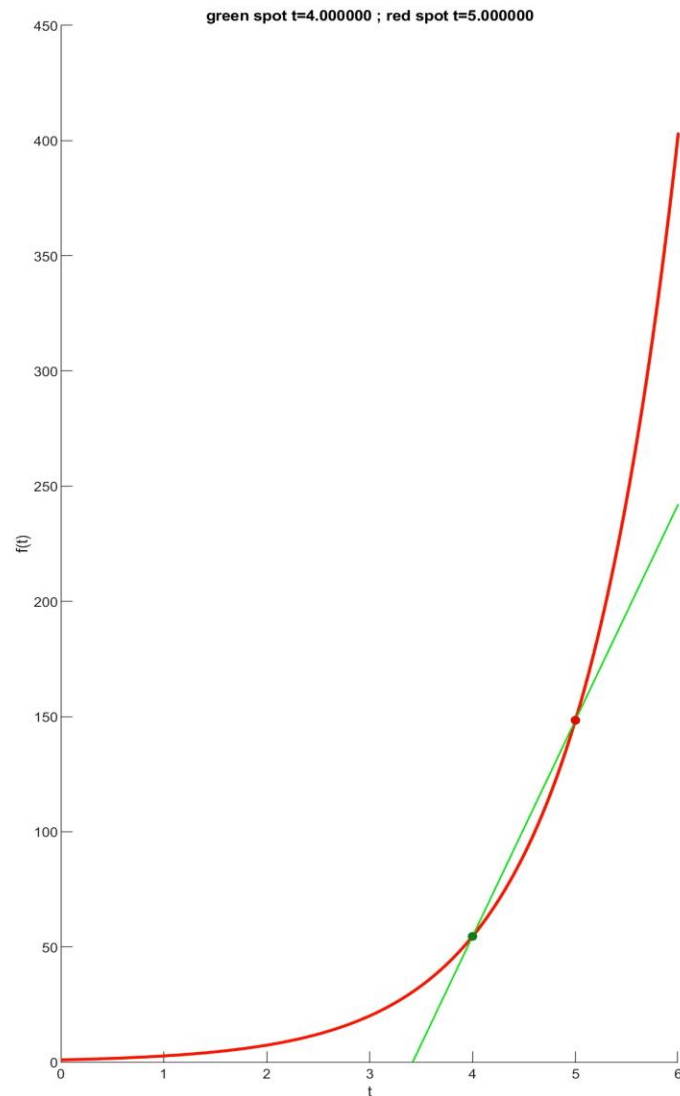
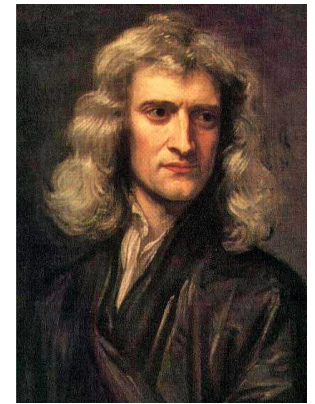
- Why do we need computer-based numerical algorithms of solving ODEs?
- Some basic methods of numerically solving ODEs
- Some important tips for applying the numerical solution of ODEs in practice

Why do we need Numerical Solutions of the ODEs

- In the previous lecture we saw how to solve an easy ODE analytically.
- However for almost any interesting differential equation model of a real-world system, we just can't solve them analytically, perhaps because
 - : - the function is nonlinear.
 - : - we don't know $g(t)$ analytically.
- Do we give up? No! Instead we use computers to approximate the solution function as a series of numbers.
- Most of the algorithms work iteratively
- There are many algorithms for doing this which vary in complexity, accuracy, and computational expense. Which algorithm performs best varies on a case-by-case basis.

Functions $f(t)$ and their Differentials $\frac{df}{dt}(t)$

Isaac
Newton
*wikipedia



$$f(t) = e^t$$

$$\frac{df}{dt}(t) = \lim_{h \rightarrow 0} \frac{f(t+h) - f(t)}{h}$$

$$\frac{df}{dt}(t) = e^t$$

Instantaneous slope at $t = 4$

!! For an extremely small interval around $t = 4$, the differential at $t = 4$ very well approximate the function.

Taylor series for $f(t)$ and Taylor approximation

Brook
Taylor
*wikipedia



If a function $f(t)$ is infinitely differentiable at a point $t=t_0$, Taylor series provides a power-series representation of the function at any other point t based on its derivatives at $t=t_0$

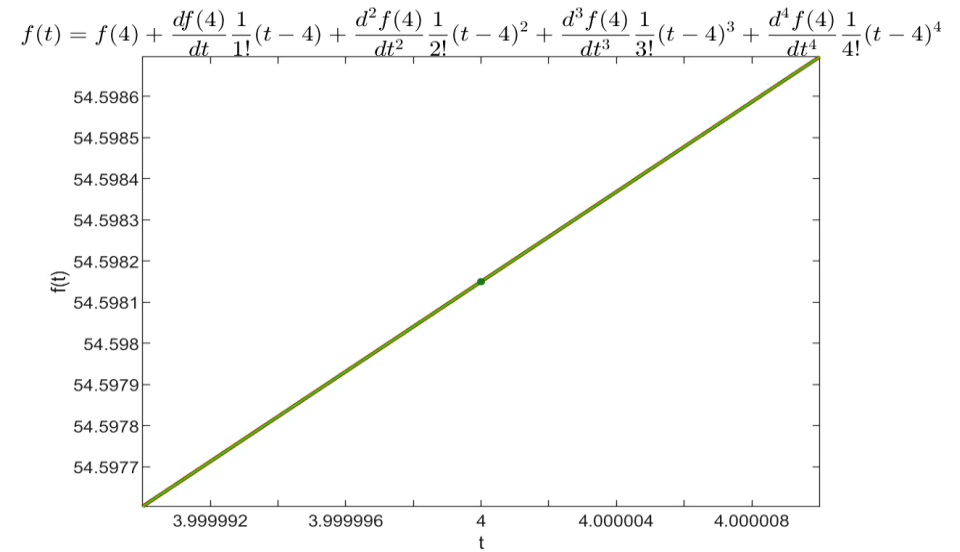
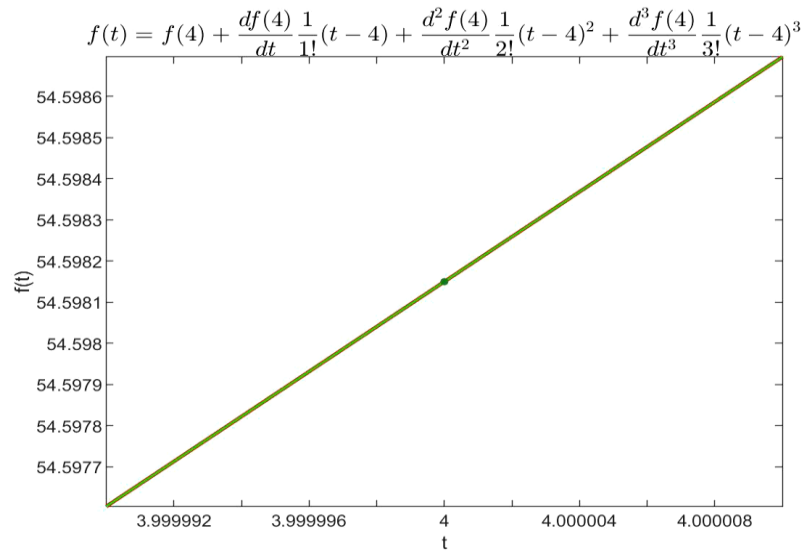
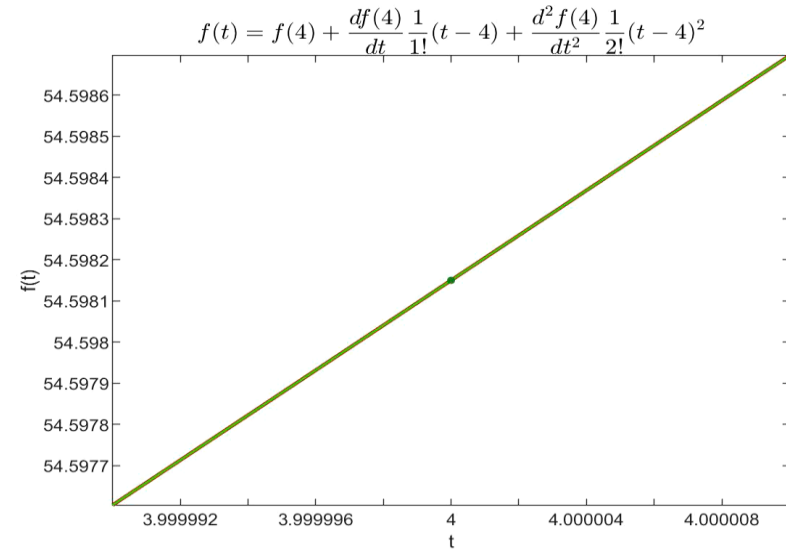
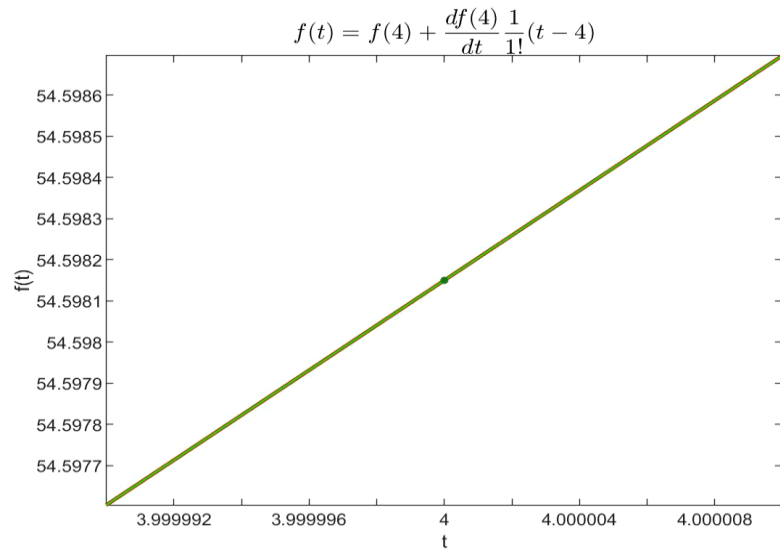
$$f(t) = \sum_{n=0}^{\infty} \frac{d^n f}{dt^n}(t_0) \frac{(t - t_0)^n}{n!}$$

It is typically used to approximate a function local to some point of interest, by truncating the series after a small number of terms:

$$f(t) \approx f(t_0) + \frac{1}{1!} \frac{df}{dt}(t - t_0) + \frac{1}{2!} \frac{d^2 f}{dt^2}(t - t_0)^2 + \frac{1}{3!} \frac{d^3 f}{dt^3}(t - t_0)^3$$

In this case, the error of the approximation is of order 4 truncation.

Taylor approximation for $f(t) = e^t$ around $t_0 = 4$



Euler's Method for Numerically Solving ODEs

Leonhard
Euler
*wikipedia



It is the most straightforward and simplest method of numerically solving ODEs.

It relies on the basic idea that, if $f(t)$ and $\frac{df}{dt}$ is known at t , the $f(t + \Delta t)$ at the next time-step $t + \Delta t$ can be linearly estimated using the tangent line at t defined by the differential $\frac{df}{dt}$:

$$f(t + \Delta t) = f(t) + \Delta t \left(\frac{df}{dt}(t) \right)$$

This strictly requires that Δt is sufficiently small enough for the linear tangent approximation to work in the neighbourhood of the point t

$$f(t + \Delta t) = f(t) + \underbrace{\frac{df(t)}{dt} \frac{\Delta t}{1!}}_{\text{Linear approximation}} + \underbrace{\frac{d^2 f(t)}{dt^2} \frac{\Delta t^2}{2!} + \frac{d^3 f(t)}{dt^3} \frac{\Delta t^3}{3!} + \frac{d^4 f(t)}{dt^4} \frac{\Delta t^4}{4!} + \dots}_{\text{Higher order terms}}$$

Linear approximation

$$y = mx + c$$

➤ Collapses to negligible when Δt is extremely small

➤ Error of approximation is $\mathcal{O}(\Delta t^2)$

Given the IVP:

Recursive Algorithm:

$$x(t_0) = x_0, \quad f(x_0, t_0)$$

$$\frac{dx}{dt} = f(x, t), \quad x(t_0) = x_0$$



$$x(t_0 + \Delta t) = x(t_0) + \Delta t f(x_0, t_0)$$



$$x(t_0 + 2\Delta t) = x(t_0 + \Delta t) + \Delta t f(x(t_0 + \Delta t), t_0 + \Delta t)$$



$$x(t_0 + 3\Delta t) = x(t_0 + 2\Delta t) + \Delta t f(x(t_0 + 2\Delta t), t_0 + 2\Delta t)$$

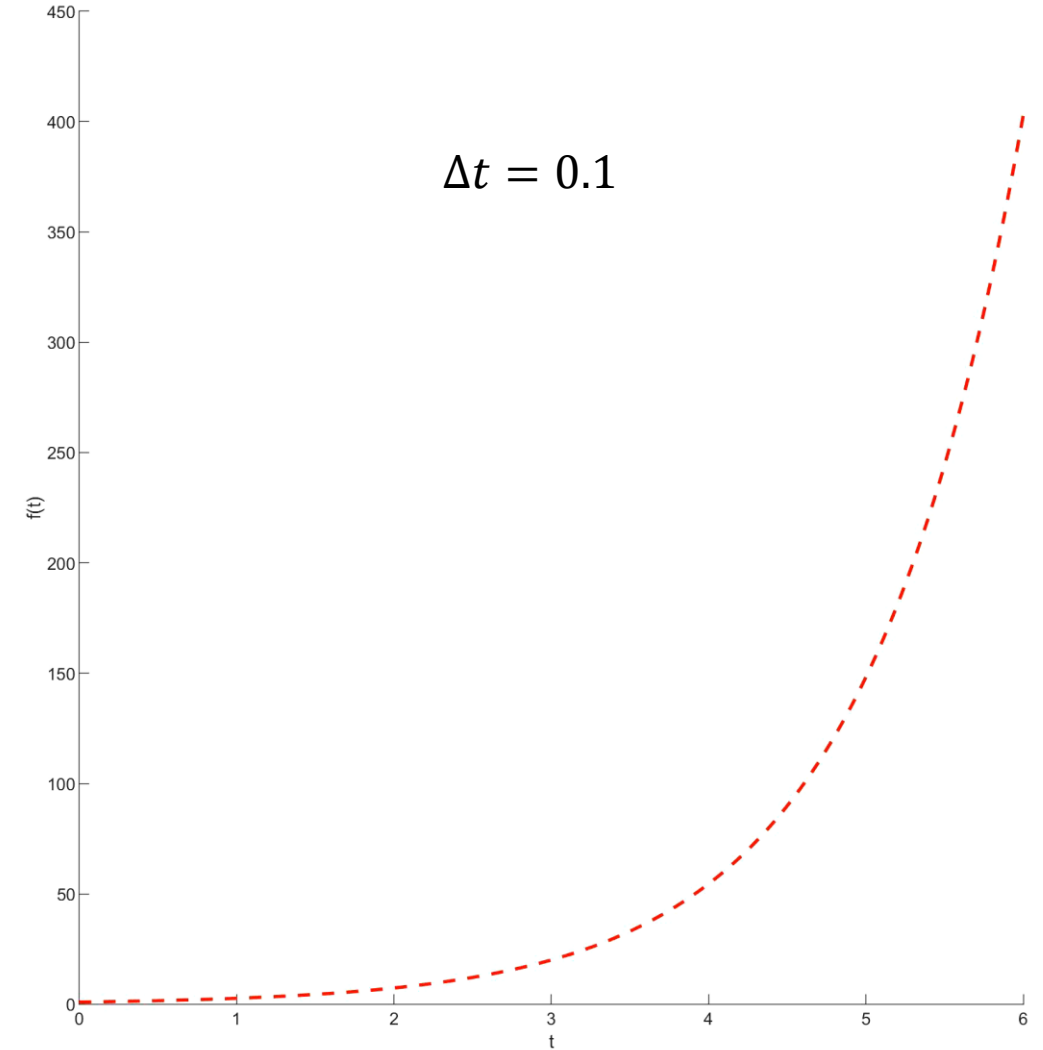
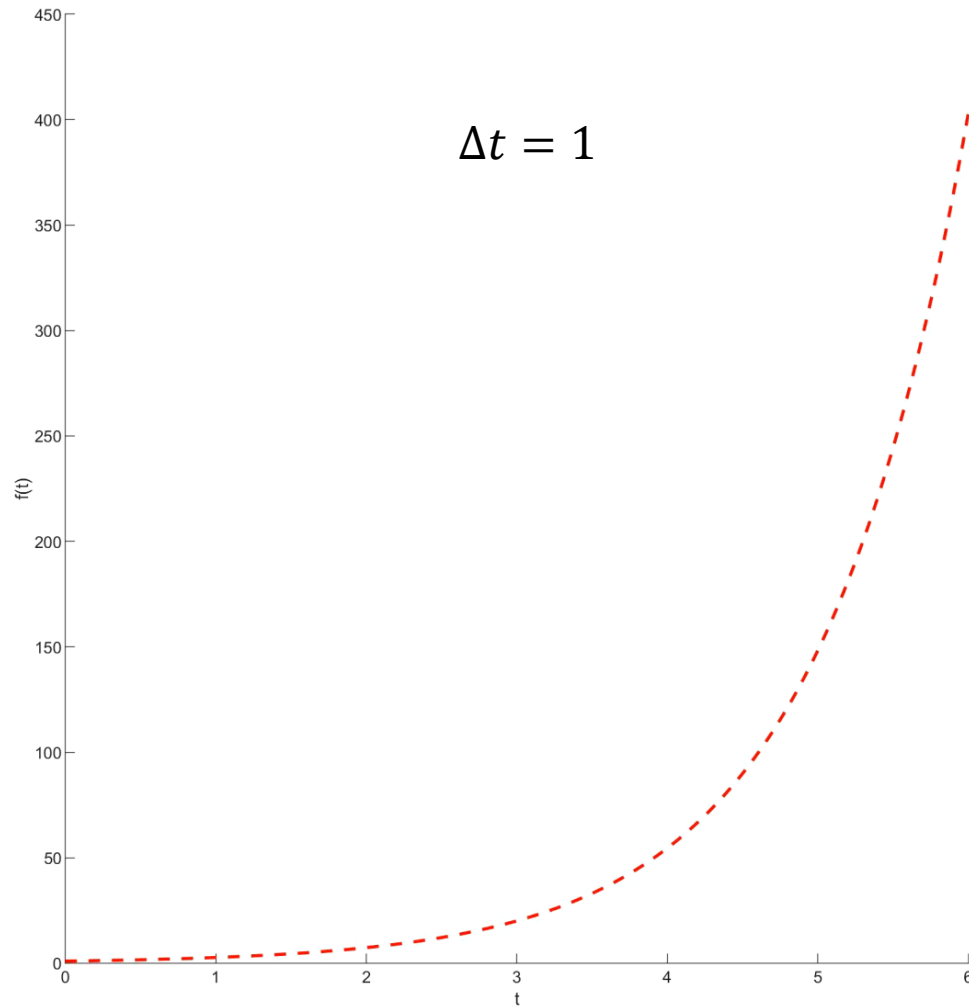


$$x(t_0 + n\Delta t) = x(t_0 + (n - 1)\Delta t) + \Delta t f(x(t_0 + (n - 1)\Delta t), t_0 + (n - 1)\Delta t)$$

$$\frac{dx}{dt}(t) = x(t), \text{ starting at } t = 0$$

Analytical Solution
➡

$$x(t) = e^t$$



!!Larger is the Δt , poorer is the approximation of the original function or solution by the numerical solution

Runge-Kutta Method for Numerically Solving ODEs

Runge-Kutta (RK) method is a family of methods with different orders. RK4 is the most popular member

Recursive Algorithm:

1. Given the IVP: $\frac{dx}{dt} = f(x, t)$, $x(t_0) = x_0$
2. Choose a step-size Δt
3. Compute $k_1 = f(x_0, t_0)$ and move to the new point $t_0 + \frac{\Delta t}{2}$ from the original point t_0 with slope k_1 ,
yielding $x\left(t_0 + \frac{\Delta t}{2}\right) = x_0 + k_1 \frac{\Delta t}{2}$
4. Now, compute $k_2 = f\left(x\left(t_0 + \frac{\Delta t}{2}\right), t_0 + \frac{\Delta t}{2}\right)$ and again move to the point $t_0 + \frac{\Delta t}{2}$ from the original point t_0 but with slope k_2 , yielding a different $x\left(t_0 + \frac{\Delta t}{2}\right) = x_0 + k_2 \frac{\Delta t}{2}$



Carl Runge

*wikipedia



Martin Kutta

*wikipedia

5. Compute $k_3 = f\left(x\left(t_0 + \frac{\Delta t}{2}\right), t_0 + \frac{\Delta t}{2}\right)$, but using $x\left(t_0 + \frac{\Delta t}{2}\right)$ obtained in step 4.

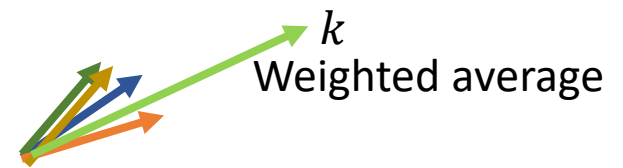
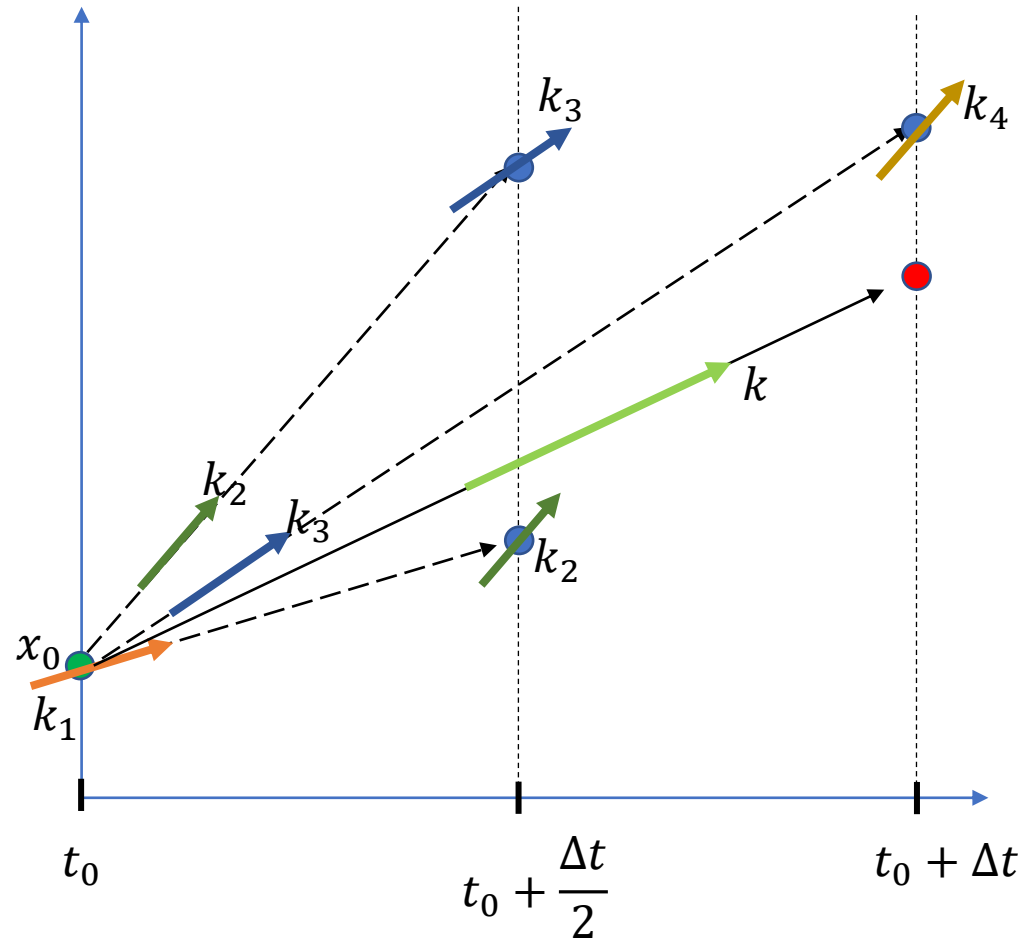
6. Now, move to the point $t_0 + \Delta t$ from the original point t_0 with slope k_3 , yielding $x(t_0 + \Delta t) = x_0 + k_3 \Delta t$

7. Compute $k_4 = f(x(t_0 + \Delta t), t_0 + \Delta t)$

8. Obtain the average slope $k = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

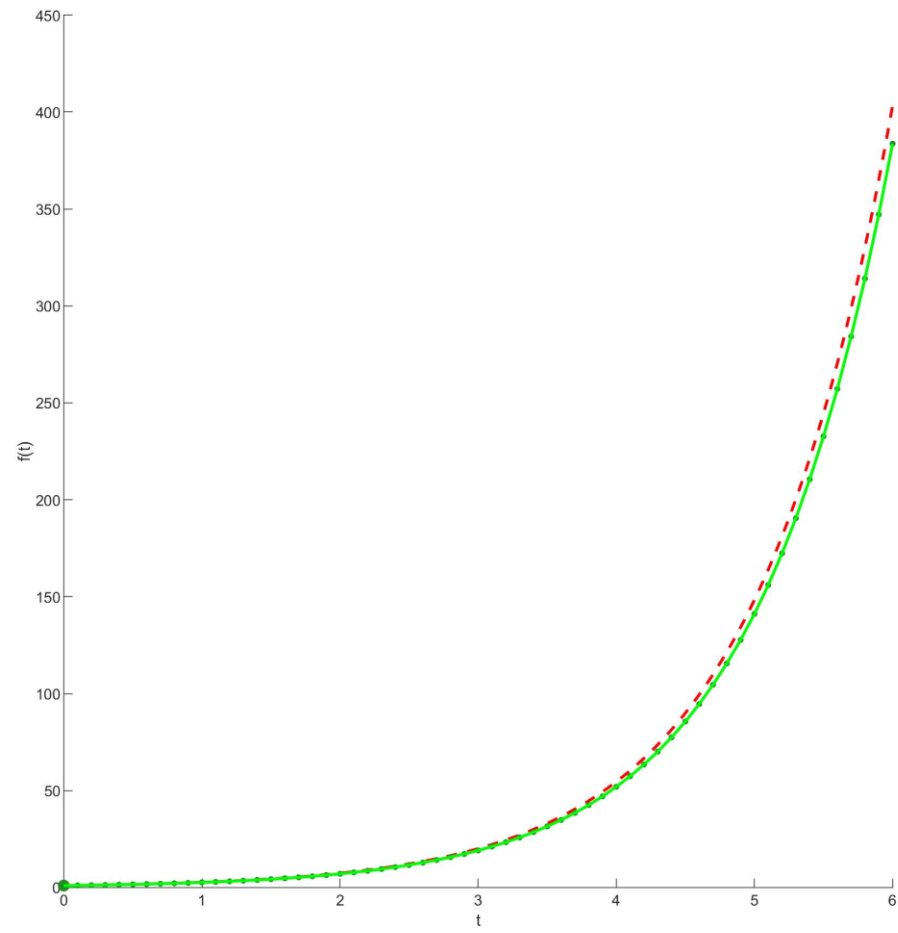
9. Make the final conclusive jump from the original point t_0 to the point $t_0 + \Delta t$, with $x(t_0 + \Delta t) = x_0 + k \Delta t$

A crude geometric depiction of RK4 Method

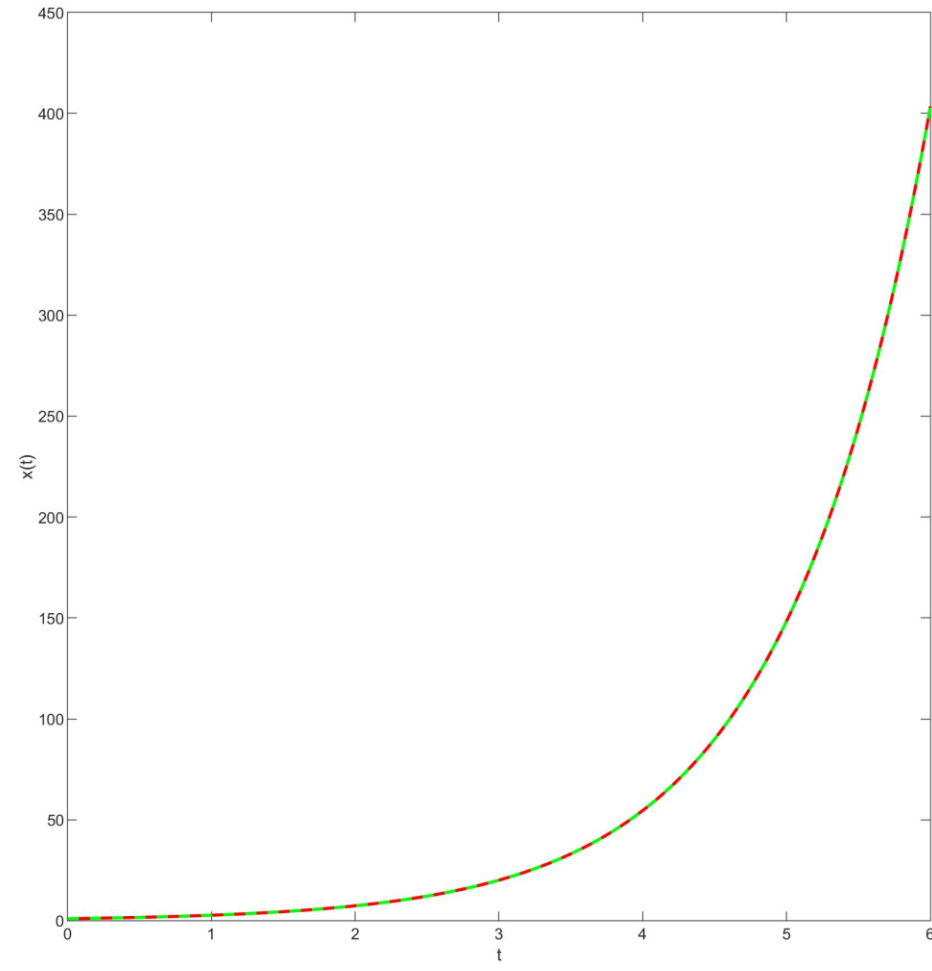


Better Accuracy: RK4 vs Euler Methods

Euler method, $\Delta t = 0.1$



RK4, $\Delta t = 0.1$



Euler vs RK in practise

- Both Euler and Runge-Kutta are widely used in practise.
- Because of its simplicity, Euler is especially common for quick-and-dirty models when people want to manually code something. You will use it in your coursework.
- Coding Runge-Kutta is more involved (usually though we just use a DE solver package that someone else has written in a black-box way).
- As you might guess Runge-Kutta is also computationally more expensive than Euler, per timestep.
- However RK's gain in accuracy over Euler usually more than offsets the increased computation time. This means that with Runge-Kutta we can get away with bigger timesteps, making it more efficient than Euler overall.