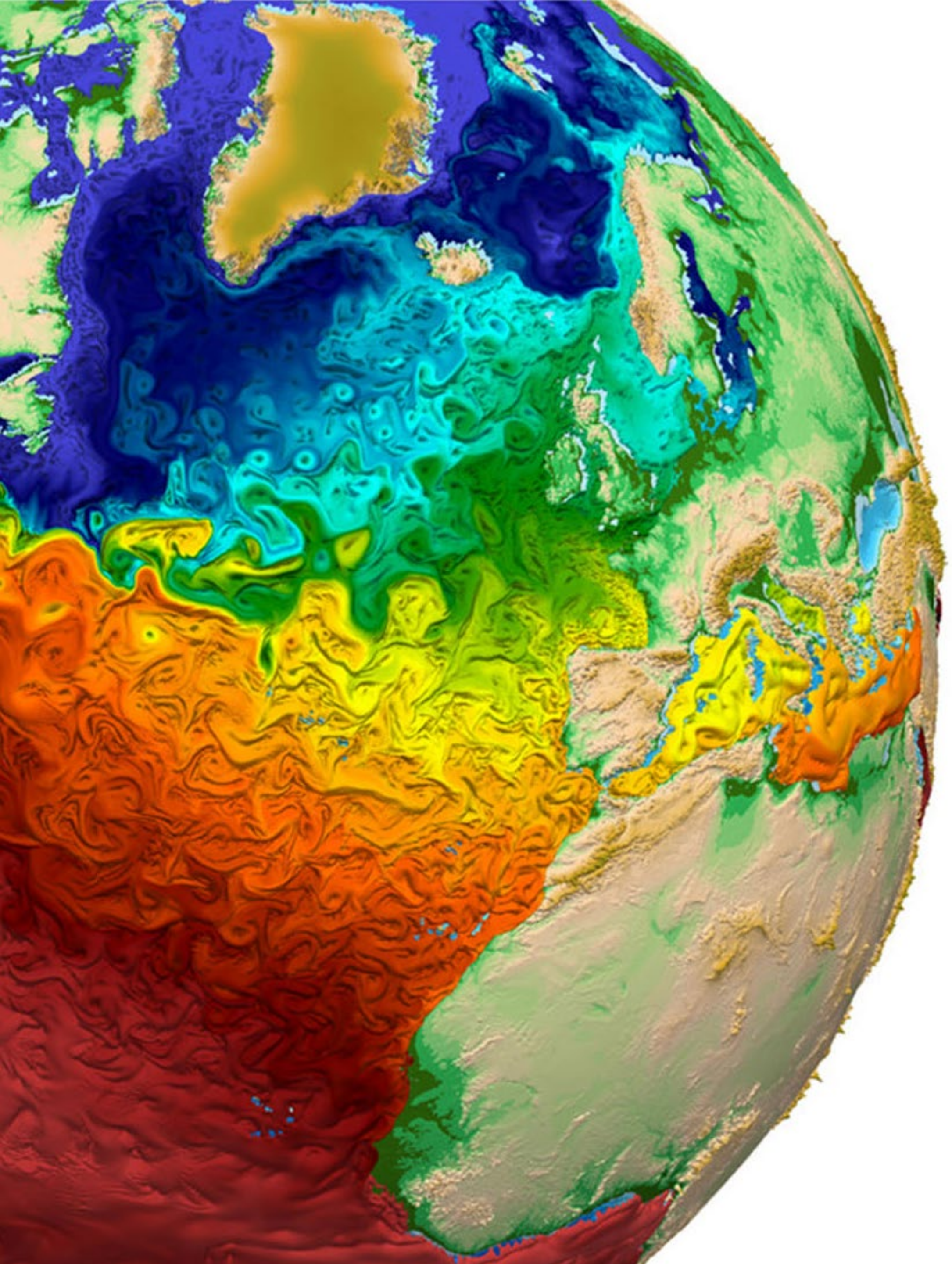


Numerical Methods

Runge-Kutta Methods

Scientific Visualization

Professor Eric Shaffer

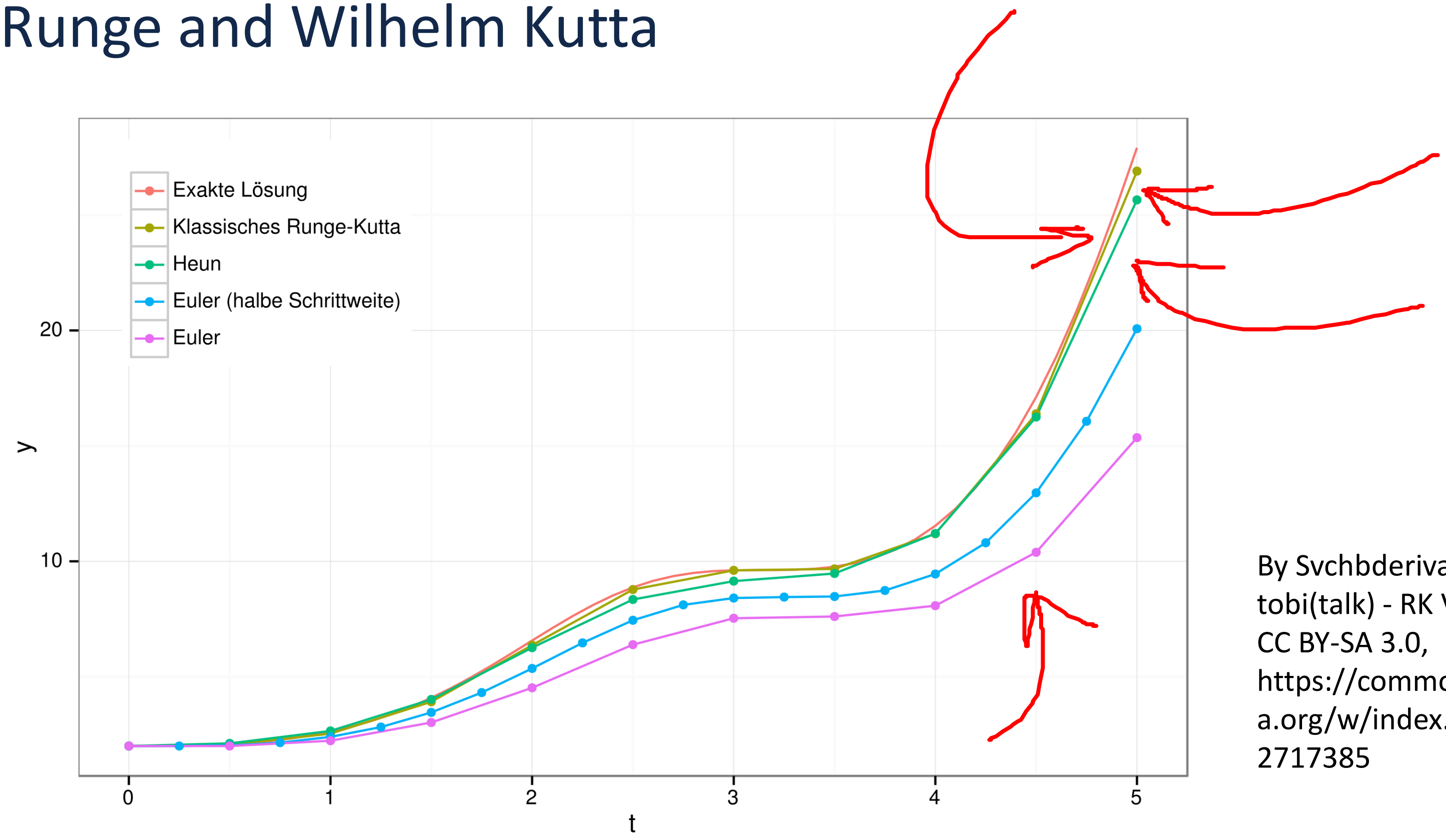
ILLINOIS

Runge-Kutta Methods

RK methods numerically solve ODEs

Developed around 1900 by Carl Runge and Wilhelm Kutta

Similar to Euler's Method
…offer higher accuracy



By Svchbderivative work: tobi(talk) - RK Verfahren, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=32717385

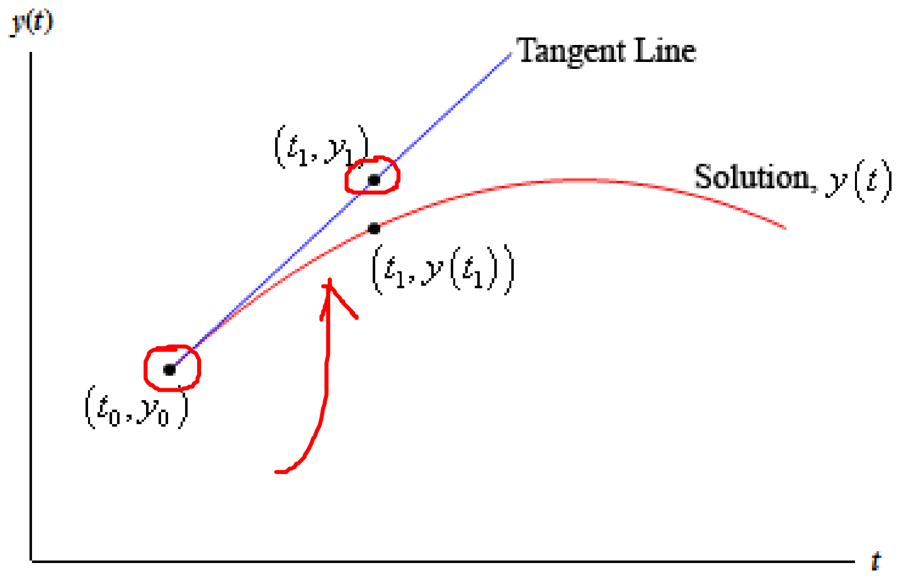
…Euler's Method can be thought of as the first order RK method

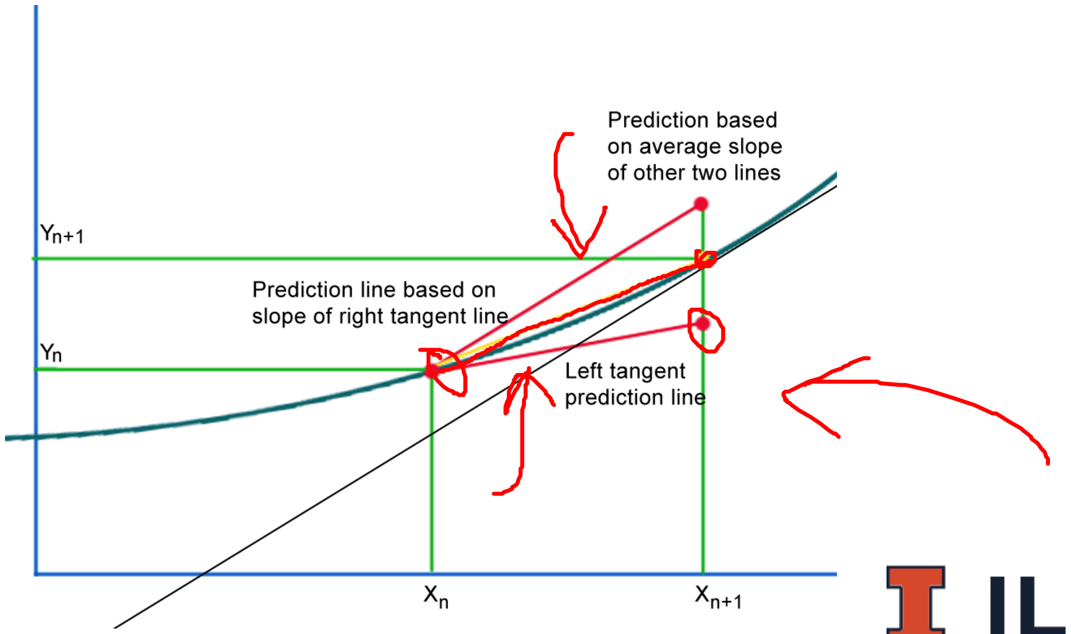
ILLINOIS

Euler's Method

Samples the rate of change at a single point each interation

Intuitively…if we sampled more points maybe we could do better





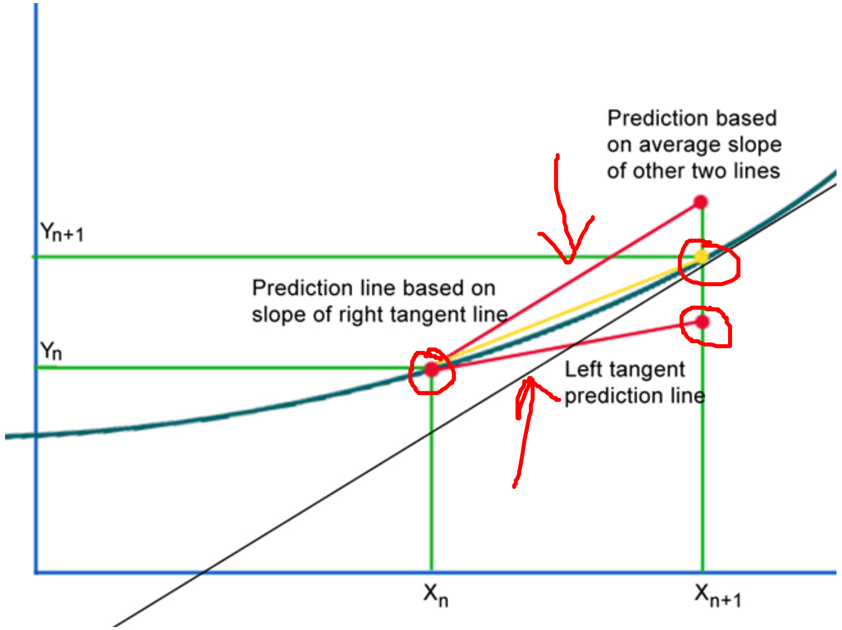
ILLINOIS

Heun's 2nd Order Method

Second order RK method

1. $\vec{k}_1 = \vec{v}(x_n, t_n)$
2. $\vec{k}_2 = \vec{v}(x_n + h_n k_1, t_n + h_n)$
3. $x_{n+1} = x_n + \frac{h_n}{2}(\vec{k}_1 + \vec{k}_2)$

Local truncation error is $O(h^3)$

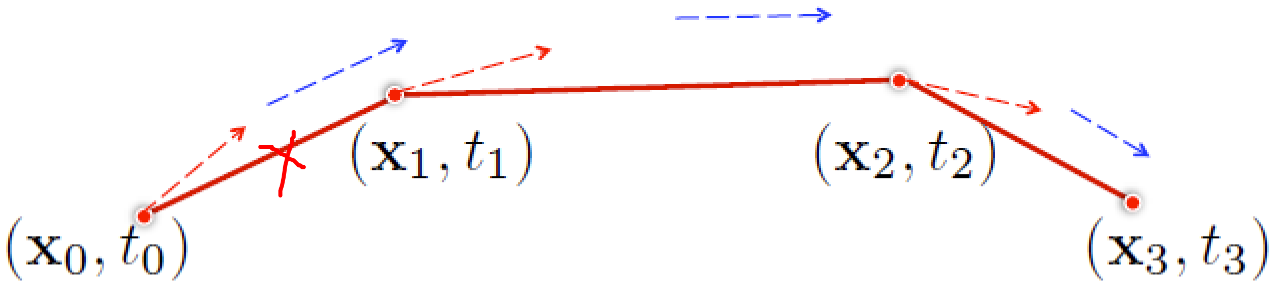


RK2 Variation....

Sometimes you will see

$$\vec{k}_1 = h\vec{v}(\mathbf{x}_n, t_n)$$
$$\vec{k}_2 = h\vec{v}(\mathbf{x}_n + \frac{1}{2}\vec{k}_1, t_n + \frac{1}{2}h)$$

$$\boxed{\mathbf{x}_{n+1} = \mathbf{x}_n + \vec{k}_2 + O(h^3)}$$



RK4

- Fourth order Runge-Kutta

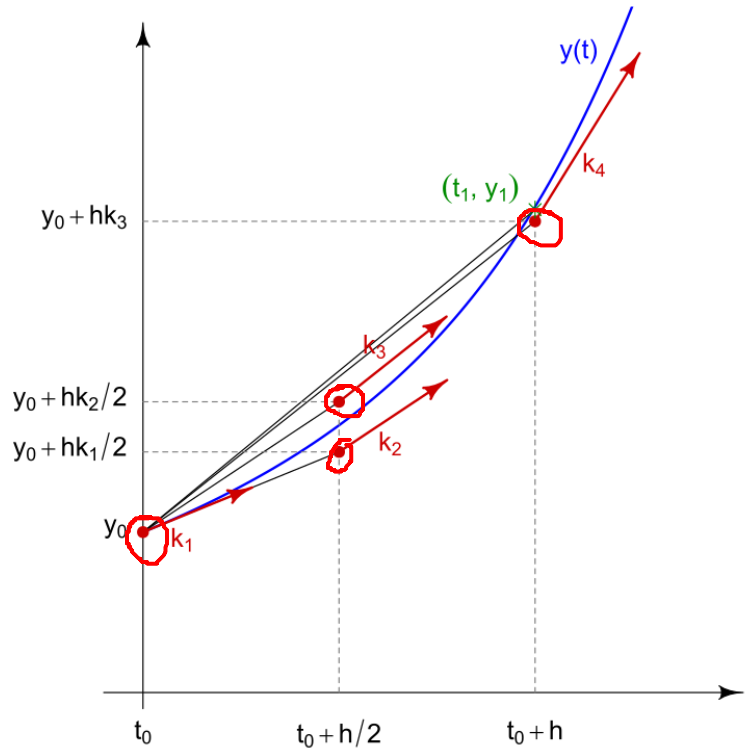
$$\vec{k}_1 = h\vec{v}(\mathbf{x}_n, t_n)$$

$$\vec{k}_2 = h\vec{v}(\mathbf{x}_n + \frac{1}{2}\vec{k}_1, t_n + \frac{1}{2}h)$$

$$\vec{k}_3 = h\vec{v}(\mathbf{x}_n + \frac{1}{2}\vec{k}_2, t_n + \frac{1}{2}h)$$

$$\vec{k}_4 = h\vec{v}(\mathbf{x}_n + \vec{k}_3, t_n + h)$$

$$\boxed{\vec{x}_{n+1} = \vec{x}_n + \frac{1}{6}\vec{k}_1 + \frac{1}{3}\vec{k}_2 + \frac{1}{3}\vec{k}_3 + \frac{1}{6}\vec{k}_4 + O(h^5)}$$



ILLINOIS

Accuracy

- RK4 requires 4 evaluations of the derivative (velocity) per step
- RK2 requires 2 evaluations per step
- Euler requires 1 evaluation per step

For RK4 to be superior it would need to be more accurate than

- RK2 using ½ the stepsize
- Euler using ¼ the stepsize

This is usually the case…but can vary by problem

Properties of Runge-Kutta Methods

- Easy to implement
- Each step requires only one previous step
- Can adjust step-size at each step…make method adaptive
  - Unfortunately no error estimate available to control step-size
  - Embedded RK methods use multiple RK methods to remedy this