Numerics!

CS2 RECITATION SERIES
FRIDAY, FEBRUARY 27, 2015

Administrivia

- The Othello Project starts soon
  - Sign up for your groups!
  - Even if you're soloing, sign up with a group name
- The Othello Project will <u>require</u> submission via Git
  - Details will be provided in the assignment instructions on Moodle
- The Othello Project will require Java to test
  - You can write your player in your choice of language (Python, C, C++, or Java)
  - Support code is provided for C++ only

This week's assignment

- Numerical integration
  - Simulating motion of things
  - Three variants of Euler's method: forward, backward, symplectic
  - We ask you to implement them
- Root finding
  - Finding x_0 so that $f(x_0) = 0$
  - Bisection method and Newton-Raphson method
- Discrete and fast Fourier transforms

Numerical Integration

- Suppose we have $x(t), v(t) = \frac{dx}{dt}, a(t) = \frac{dv}{dt} = f(x, t)$
  - We want to simulate the evolution of x over time
  - If there's no analytic solution, we need to simulate x numerically (with timestep δ)

Euler methods

- Forward Euler
  - $v(t+\delta) = v(t) + \delta \cdot f\big(t, x(t)\big)$
  - $x(t+\delta) = x(t) + \delta \cdot v(t)$
- Backward Euler
  - $v(t+\delta) = v(t) + \delta \cdot f\big(t+\delta, x(t+\delta)\big)$
  - $x(t+\delta) = x(t) + \delta \cdot v(t+\delta)$
  - Next x and v depend and next x and v … ???
  - Do some algebra :) (there are hints in the assignment)
- Symplectic Euler
  - $v(t+\delta) = v(t) + \delta \cdot f\big(t, x(t)\big)$
  - $x(t+\delta) = x(t) + \delta \cdot v(t+\delta)$
  - Next x depends on next v

Forward Euler

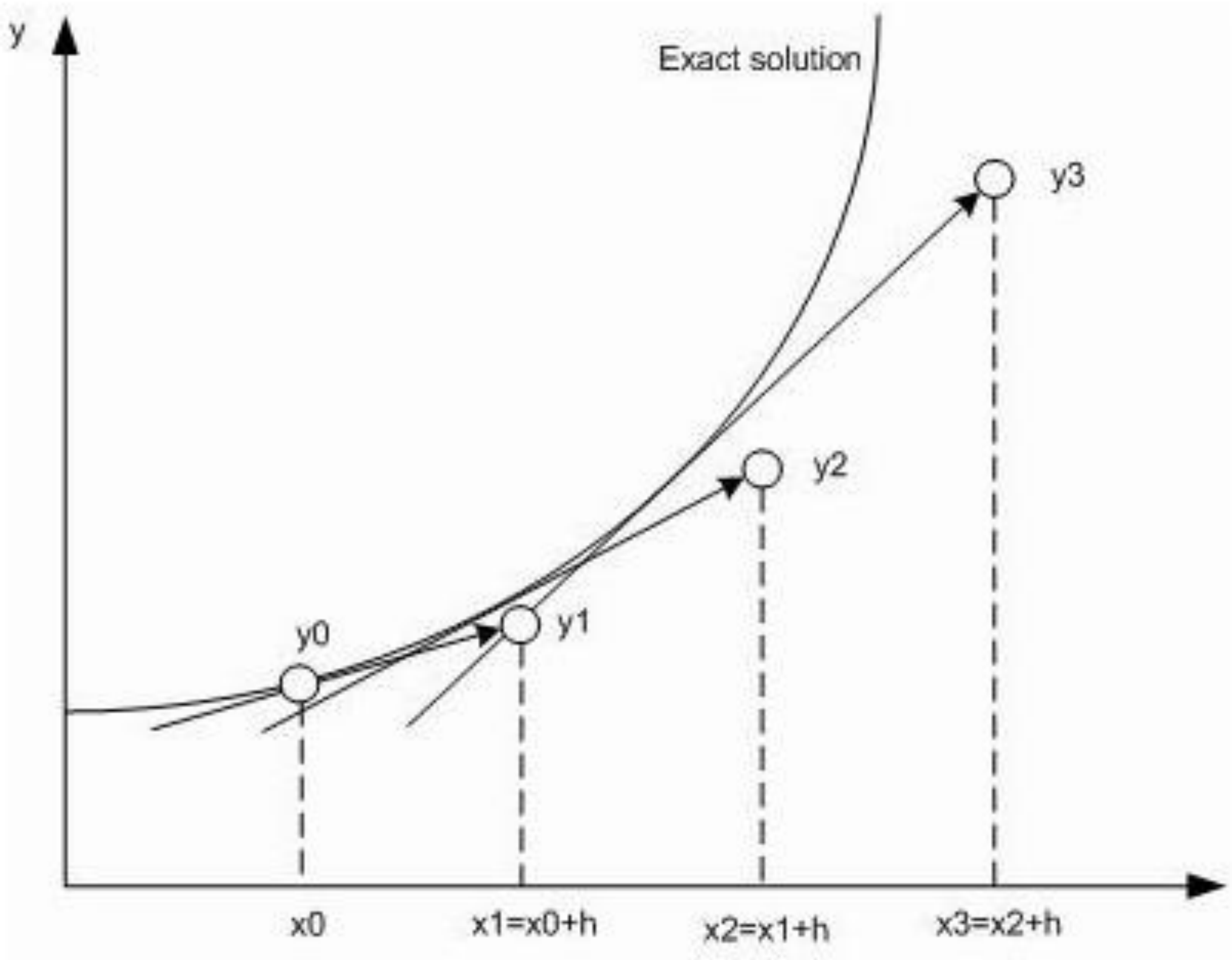


Image source: http://alexkr.com/source-code/127/intuitive-understanding-of-ode-solvers/

Backward Euler

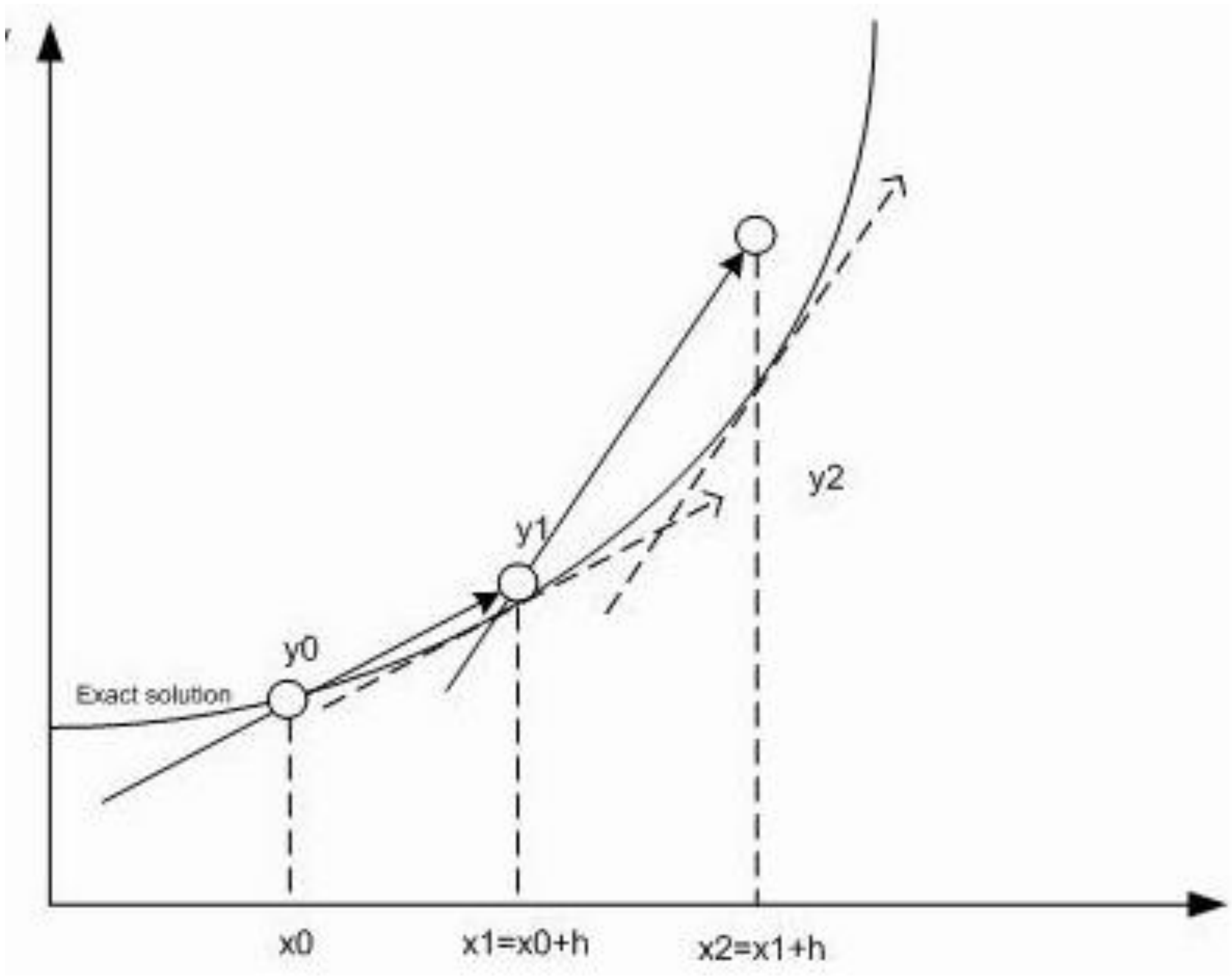


Image source: http://alexkr.com/source-code/127/intuitive-understanding-of-ode-solvers/

Euler methods in assignment

- You are given a parameter h in each of the functions as well as the current x, y, vx, vy values as pointers
  - h represents the timestep - δ in the previous slide
- Update position and velocity through the pointers you are given, function does not return anything

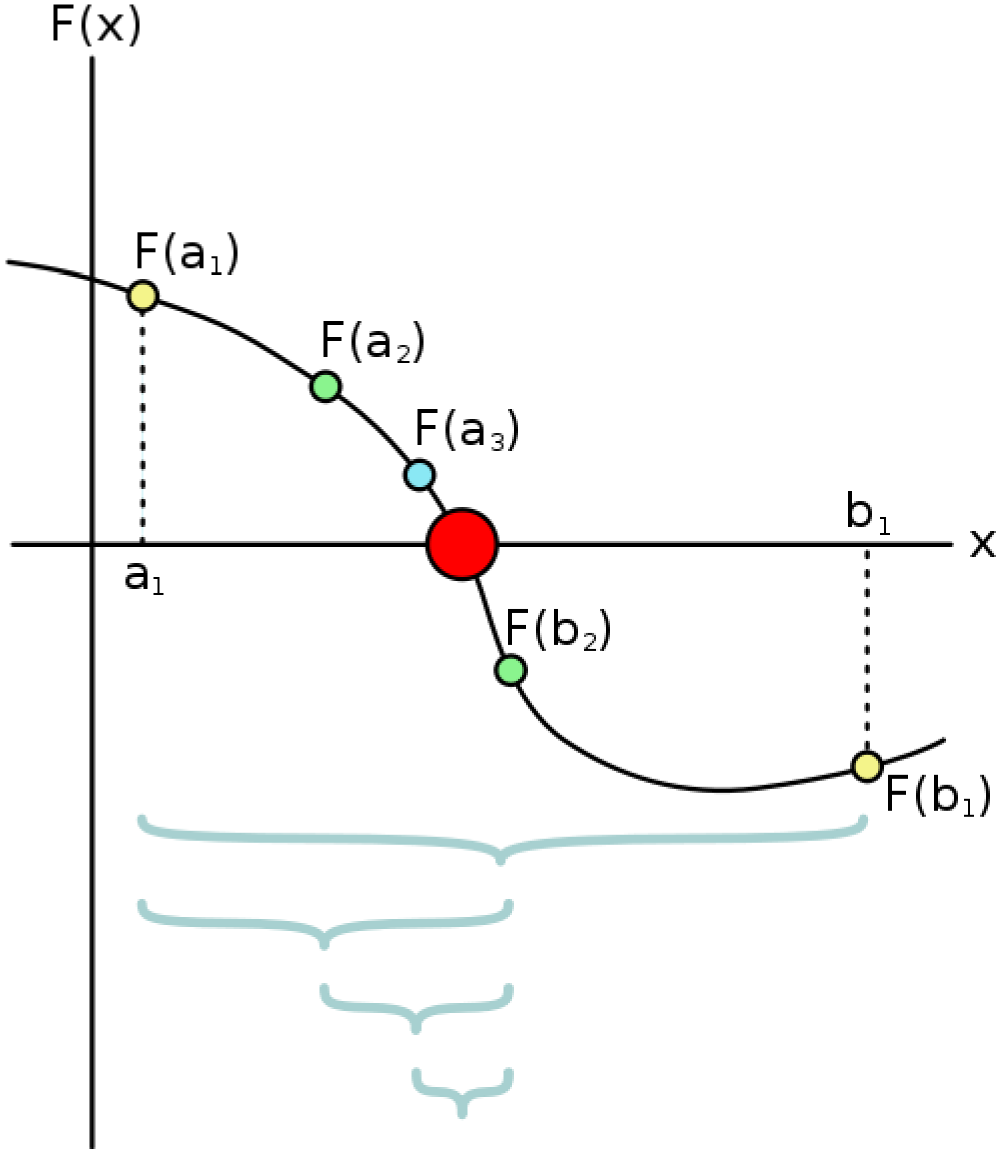
Root Finding

- This is exactly what it sounds like
- Some analytic functions do not have closed form solutions
- We have some numerical methods to find roots for these functions

The bisection method

- If we have a continuous function $f(x)$ and x_1, x_2 such that $f(\mathrm{x}_1)$ and $f(x_2)$ have opposite signs, the intermediate value theorem tells us that there is a root between x_1 and x_2. Assume $x_1 < x_2$ and start with the range $[x_1, x_2]$.
- Guess the root: $x_0 = \frac{x_1+x_2}{2}$
- If $f(x_0)$ has the same sign as $f(x_1)$, change our range to $[x_0, x_2]$
- Otherwise change our range to $[x_1, x_0]$
- Repeat until the length of the range is smaller than some given parameter
- In the assignment you are given x_1, x_2 and a parameter PRECISION. The function is passed as a function pointer, don't be scared of this
  - Calling the function is easy this way, $f(a)$ is simply $f(a)$ in the code

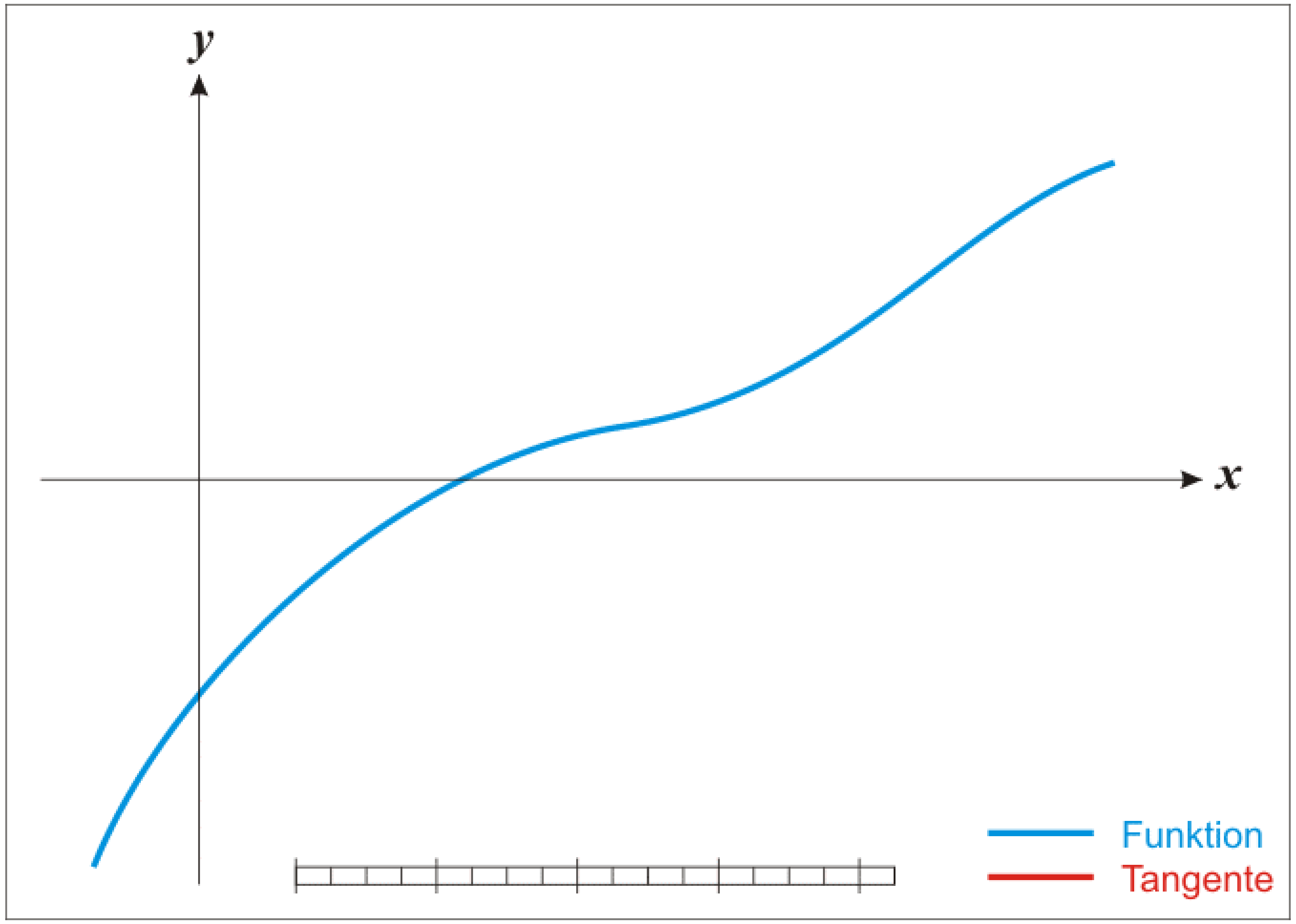
Picture example (photo from Wikipedia)



Newton-Raphson Method

- General idea is to continue tracing tangent lines until we get to our root
- Start with an initial guess x_1
- Linearize the function around x_1 by taking the first two terms of its Taylor expansion
  - $f(x) \approx f(x_1) + f'(x_1)(x - x_1)$
  - Set $f(x) = 0$ and solve the above equation to find
  - $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$
- If x_2 is close enough to zero (TOLERANCE in the assignment) then we found the root!
  - Evaluate the derivative the same way you evaluated functions in the bisection method

Newton-Raphson (animation)



Source: Wikipedia

Discrete and Fast Fourier Transforms

- This was covered very thoroughly in class, see lecture slides (they will be posted soon) for details
- We provide you a complex number class
  - You will probably find the getRootOfUnity function useful
  - Returns the primitive n^{th} root of unity of z in $z^n = 1$
- What to do with the roots?
  - See lecture slide
- FFT pseudocode is given on lecture slides – should be easy to follow