Initial Value Problems

Paul Heckbert

Computer Science Department
Carnegie Mellon University

Generic First Order ODE

given

$y' = f(t, y)$

$y(t_0) = y_0$

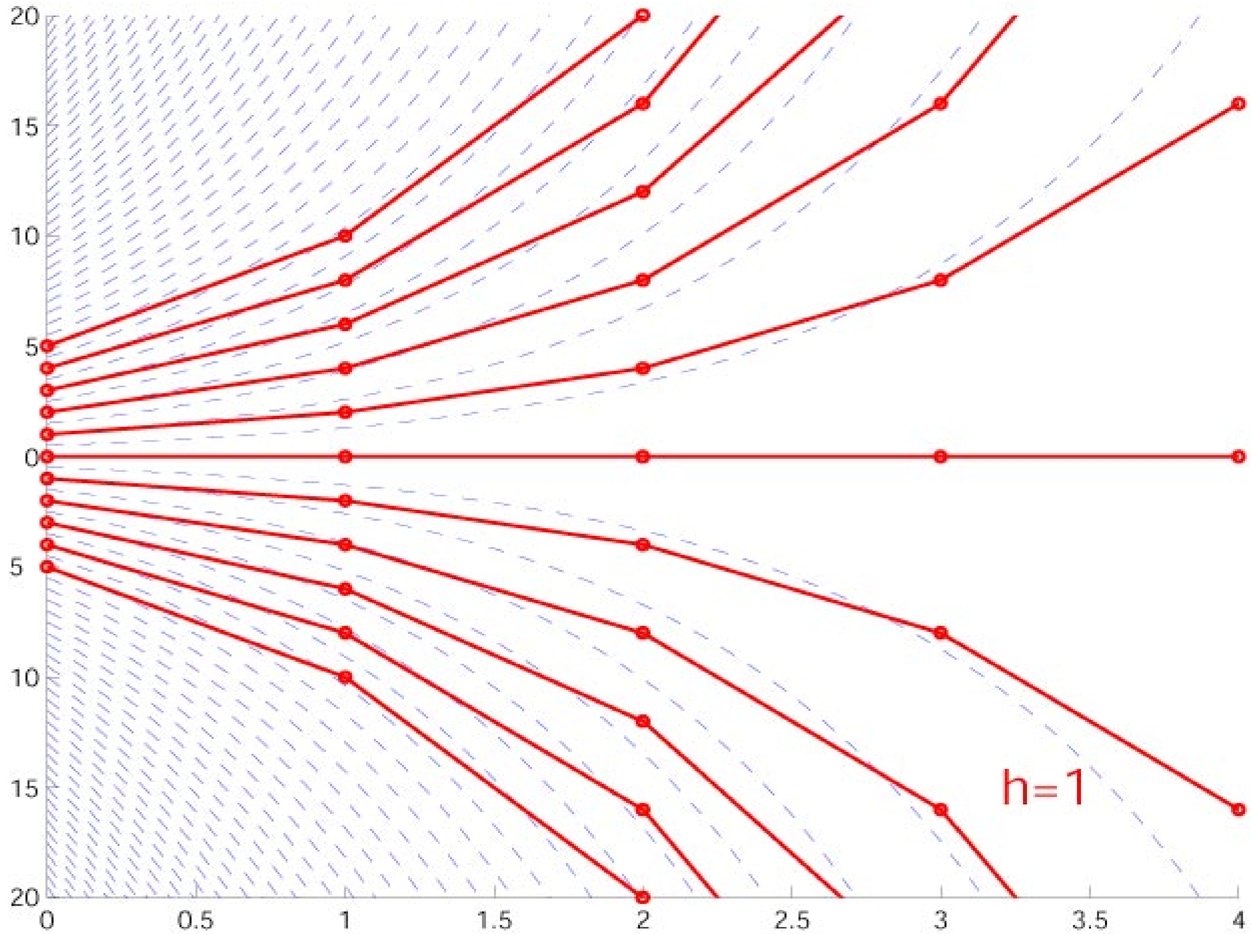
solve for $y(t)$ for $t > t_0$

First ODE:
$y' = y$

- ODE is unstable
- (solution is $y(t)=ce^t$)

- we show solutions with Euler's method

$y' = y$



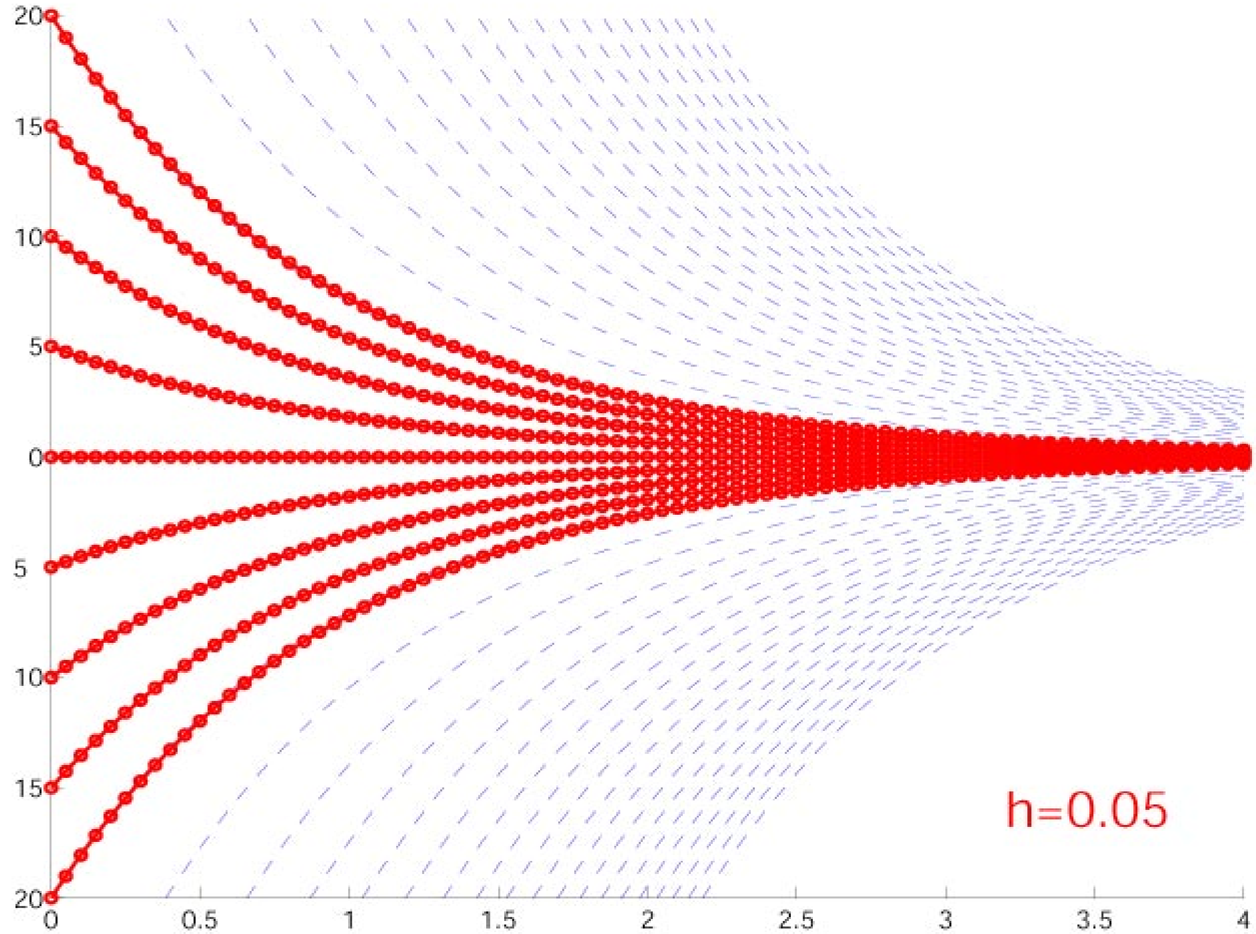
Second ODE:
$y' = -y$

- ODE is stable
- (solution is $y(t) = ce^{-t}$)

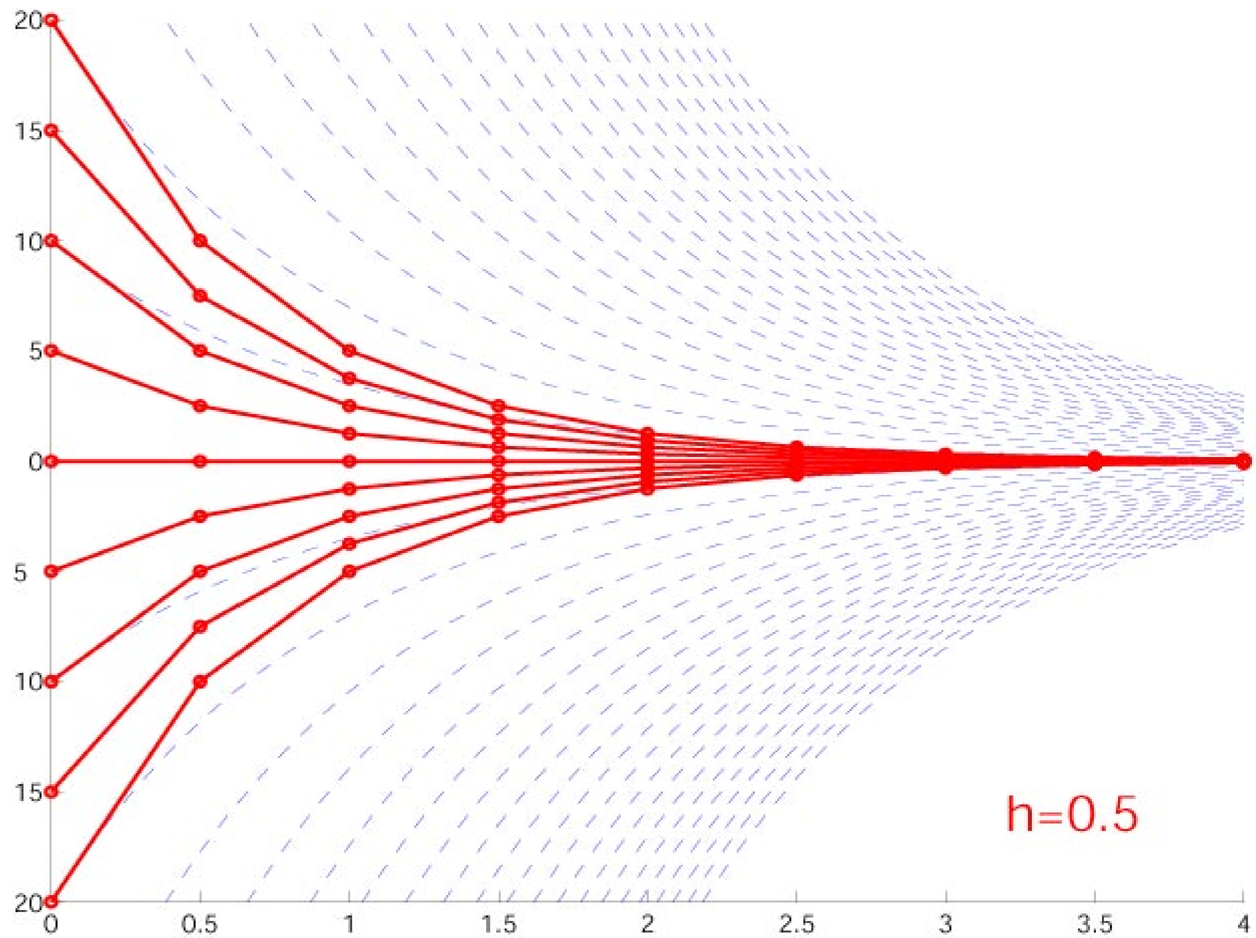
- if h too large, numerical solution is unstable

- we show solutions with Euler's method in red

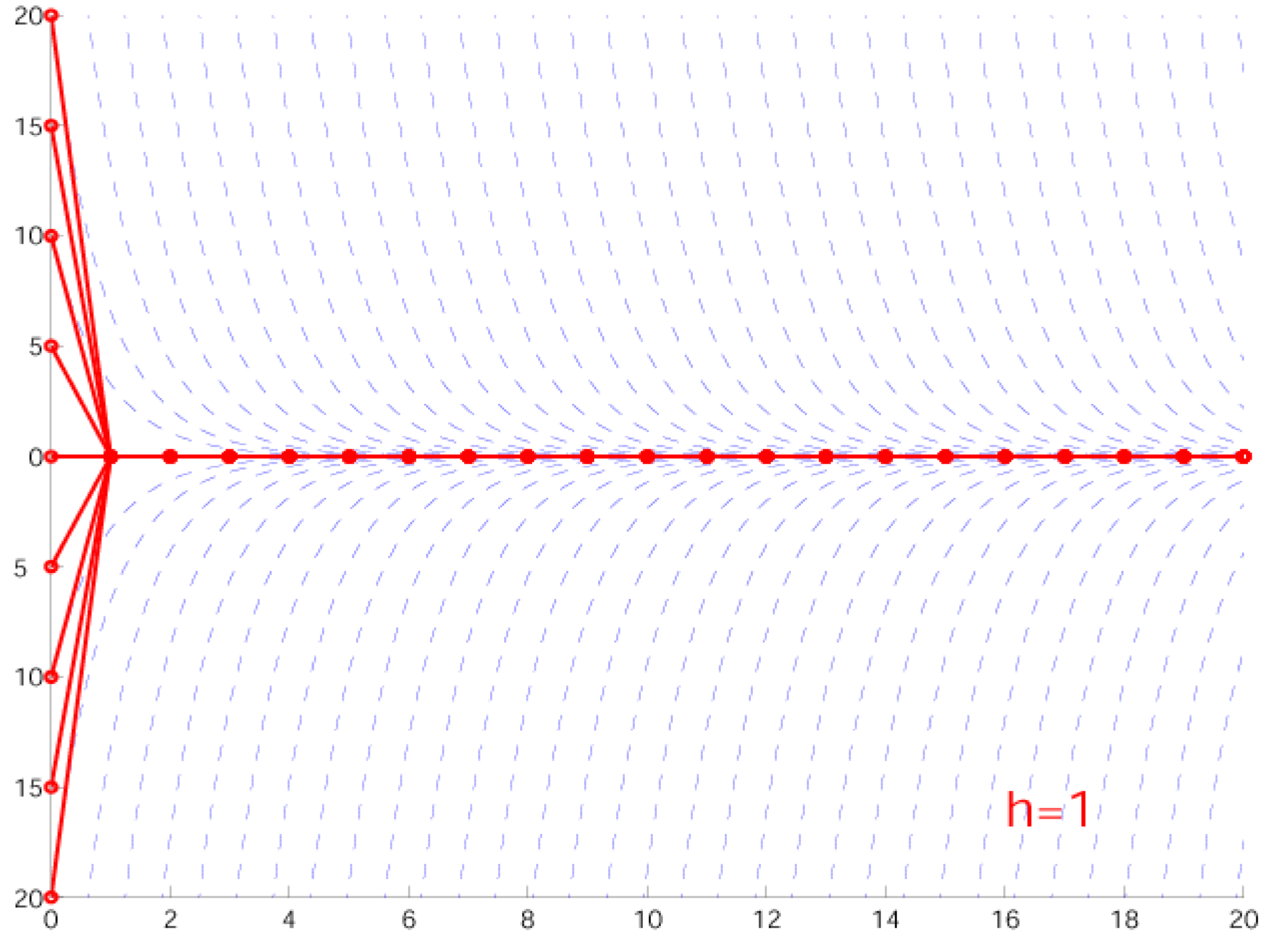
$y' = -y$, stable but slow solution



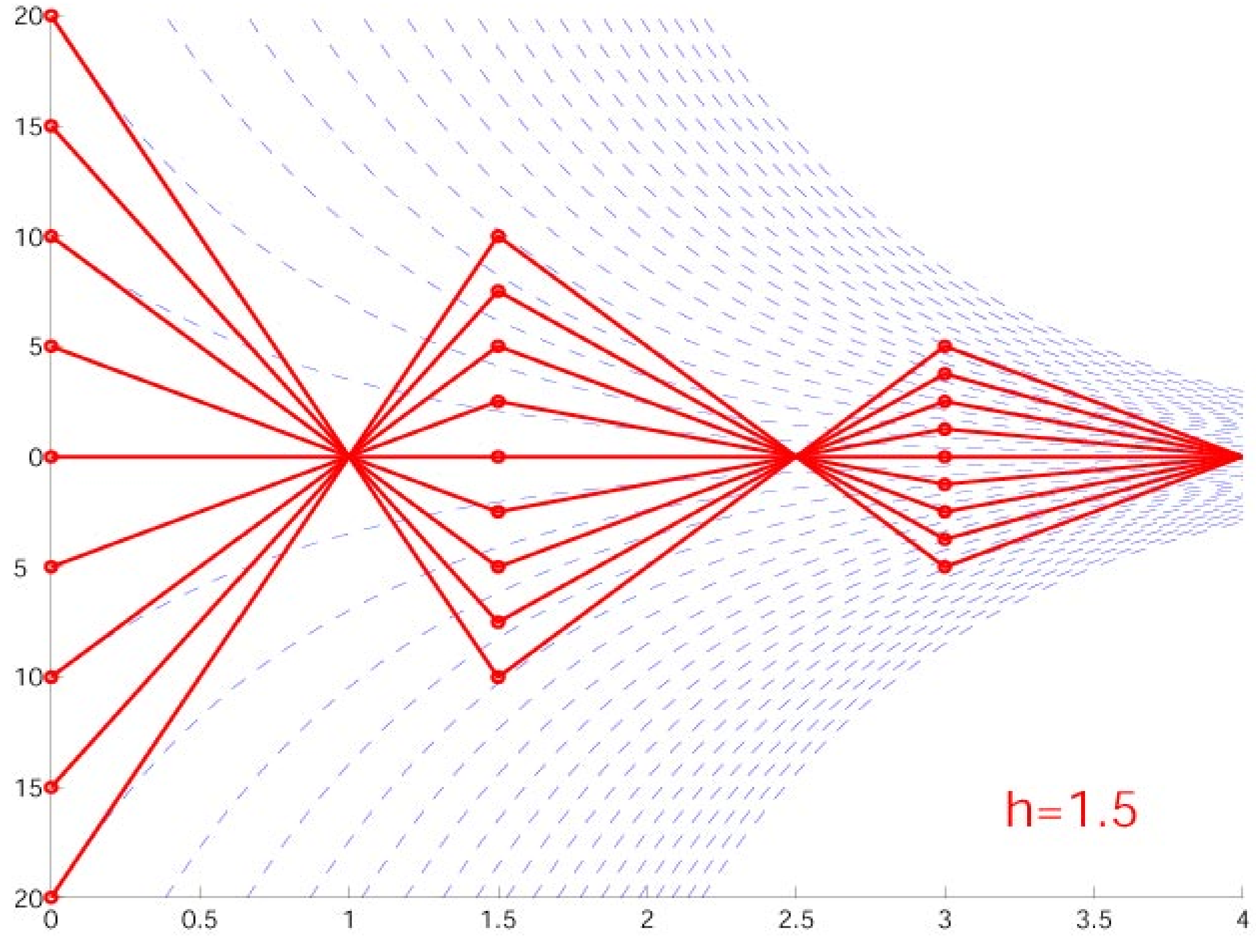
$y'=-y$, stable, a bit inaccurate soln.



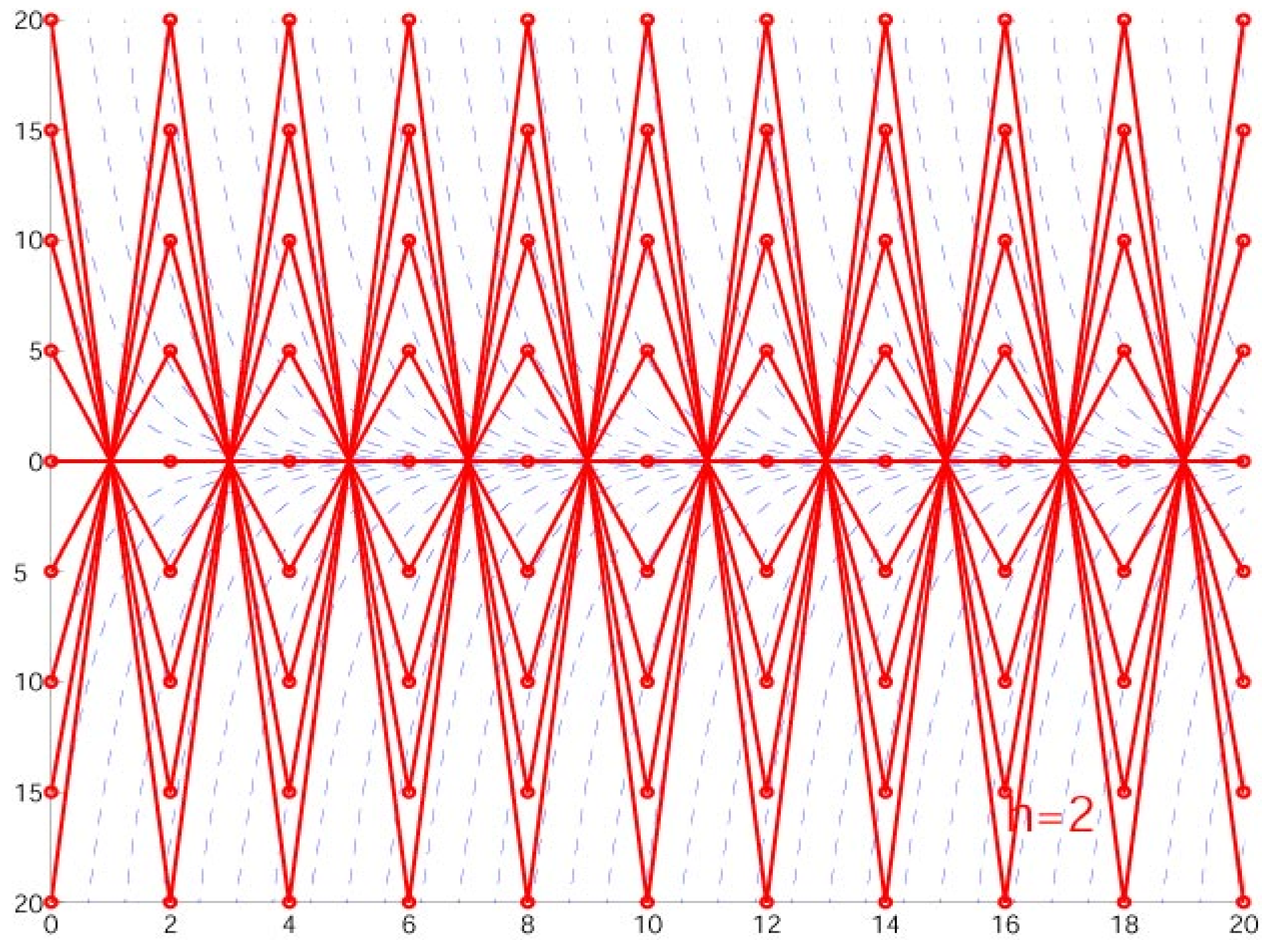
$y' = -y$, stable, rather inaccurate soln.



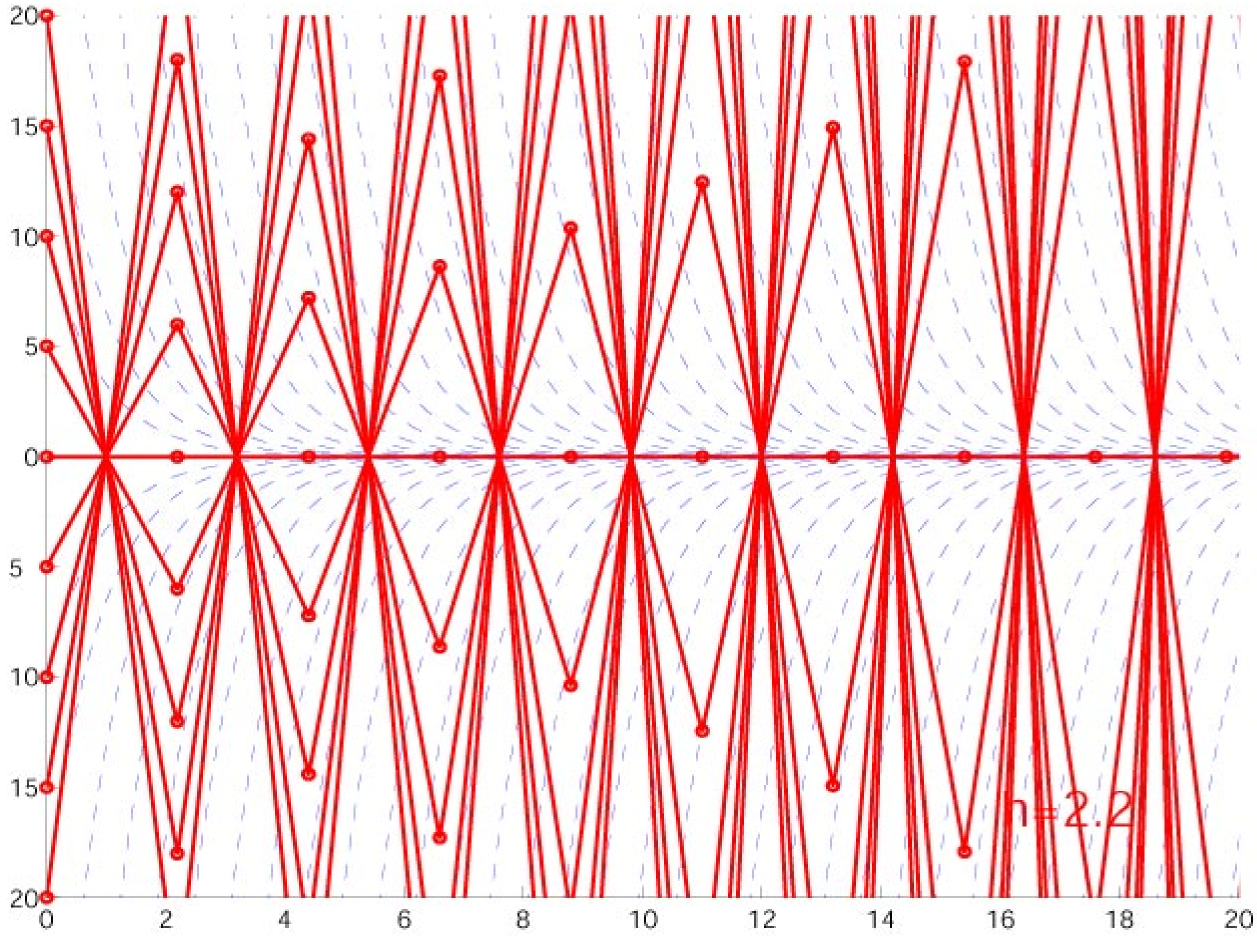
$y' = -y$, stable but poor solution



$y' = -y$, oscillating solution



$y' = -y$, unstable solution



Jacobian of ODE

- ODE: $y' = f(t,y)$, where y is n-dimensional
- Jacobian of f is $J_{ij} = \frac{\partial f_i}{\partial y_j}$ a square matrix
- if ODE homogeneous and linear then J is constant and $y' = Jy$
- but in general J varies with t and y

Stability of ODE depends on Jacobian

At a given (t,y) find $J(t,y)$ and its eigenvalues

find $r_{\max} = \max_i \{ \mathrm{Re}[\lambda_i(J)] \}$

if $r_{\max}<0$, ODE stable, locally

$r_{\max}=0$, ODE neutrally stable, locally

$r_{\max}>0$, ODE unstable, locally

Stability of Numerical Solution

- **Stability of numerical solution is related to, but not the same as stability of ODE!**

- *Amplification factor* of a numerical solution is the factor by which global error grows or shrinks each iteration.

Stability of Euler's Method

- Amplification factor of Euler's method is $I+hJ$
- Note that it depends on h and, in general, on t & y.
- Stability of Euler's method is determined by eigenvalues of $I+hJ$
- spectral radius $\rho(I+hJ)= \max_i | \lambda_i(I+hJ) |$
- if $\rho(I+hJ)<1$ then Euler's method stable
  - if all eigenvalues of hJ lie inside unit circle centered at –1, E.M. is stable
  - scalar case: $0<|hJ|<2$ iff stable, so choose $h < 2/|J|$
- What if one eigenvalue of J is much larger than the others?

Stiff ODE

- An ODE is s*tiff* if its eigenvalues have greatly differing magnitudes.

- With a stiff ODE, one eigenvalue can force use of small *h* when using Euler’s method

Implicit Methods

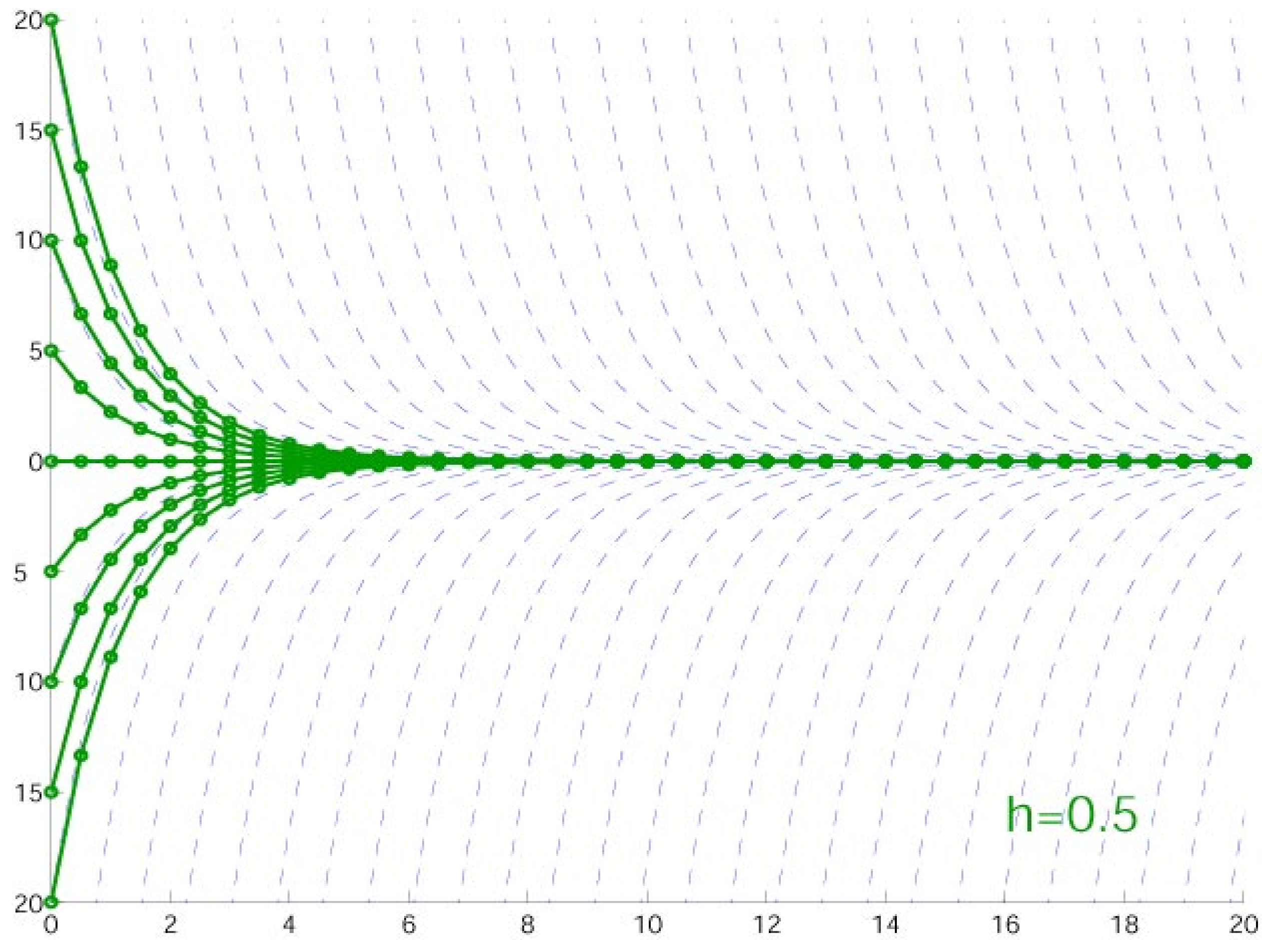
- use information from future time t_{k+1} to take a step from t_k
- Euler method: $y_{k+1} = y_k + f(t_k, y_k) h_k$
- backward Euler method: $y_{k+1} = y_k + f(t_{k+1}, y_{k+1}) h_k$

- example:
- $y' = Ay \quad f(t,y) = Ay$
- $y_{k+1} = y_k + A y_{k+1} h_k$
- $(I - h_k A) y_{k+1} = y_k$ -- *solve this system each iteration*

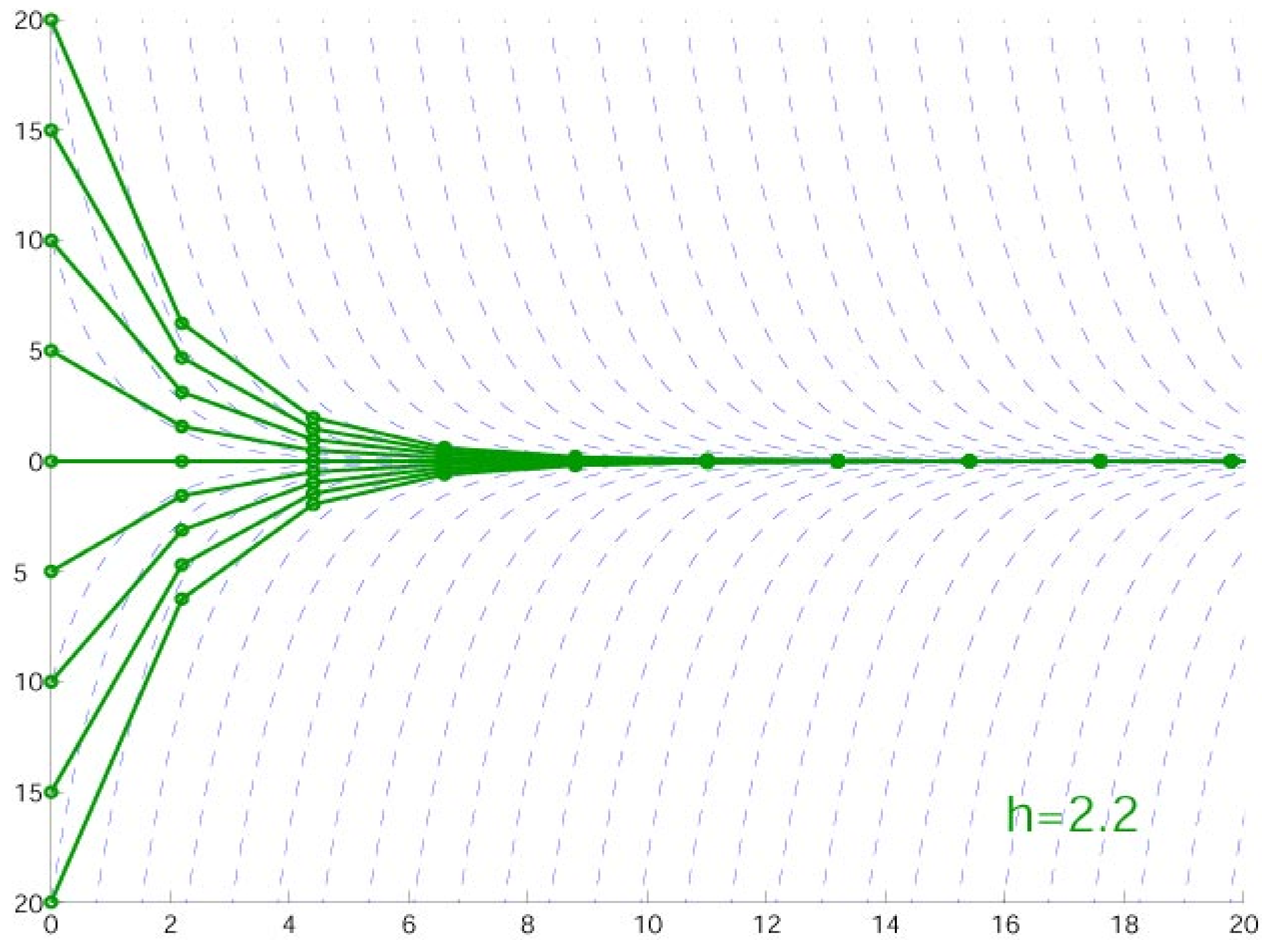
Stability of Backward Euler's Method

- Amplification factor of B.E.M. is $(I-hJ)^{-1}$
- B.E.M. is stable independent of *h* (*unconditionally stable*) as long as $r_{max}<0$, i.e. as long as ODE is stable
- Implicit methods such as this permit bigger steps to be taken (larger *h*)

$y' = -y$, B.E.M. with large step



$y' = -y$, B.E.M. with very large step

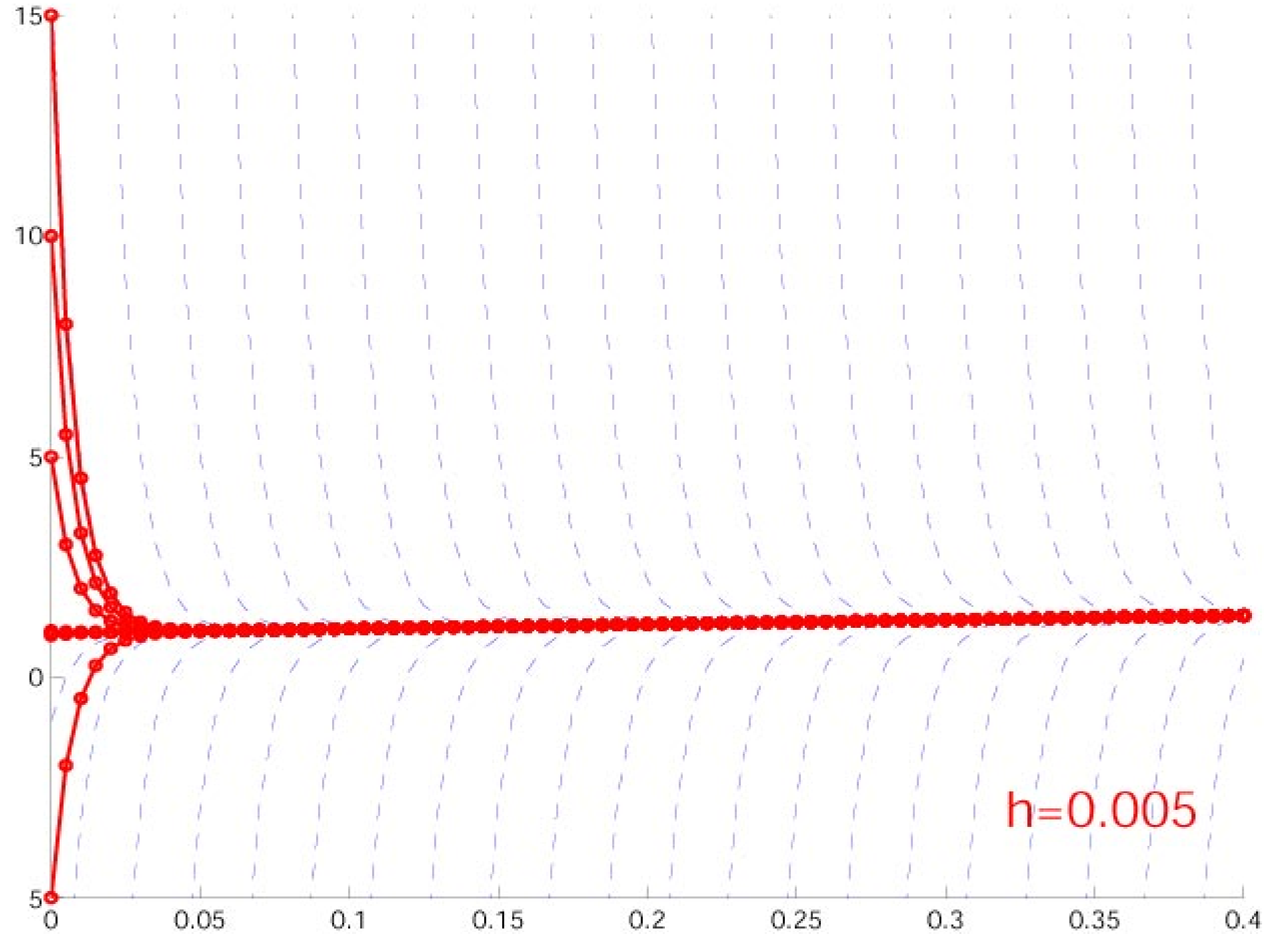


Third ODE: $y' = -100y + 100t + 101$

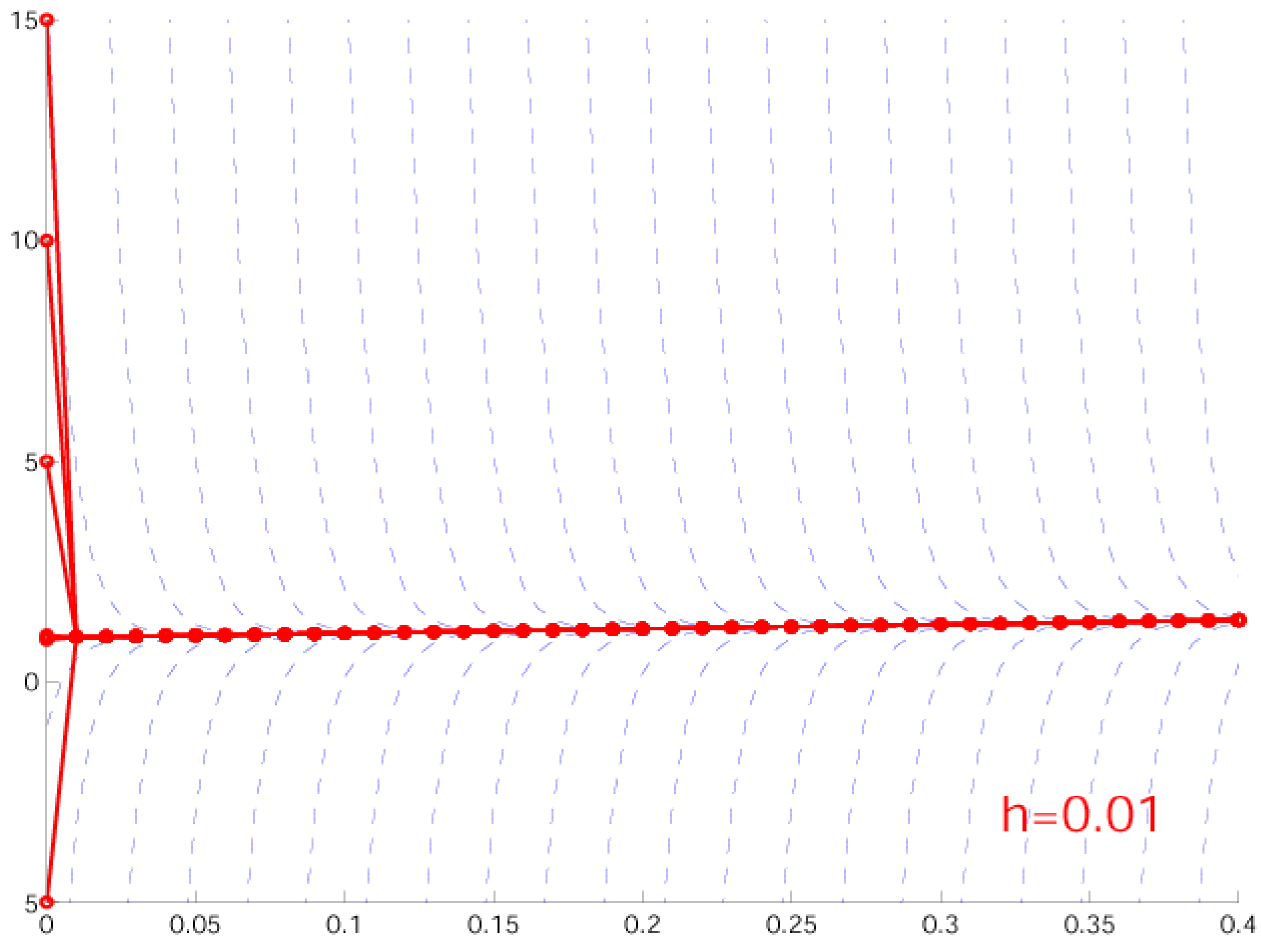
- ODE is stable, very stiff
- (solution is $y(t) = 1 + t + ce^{-100t}$)

- we show solutions:
  - Euler's method in red
  - Backward Euler in green

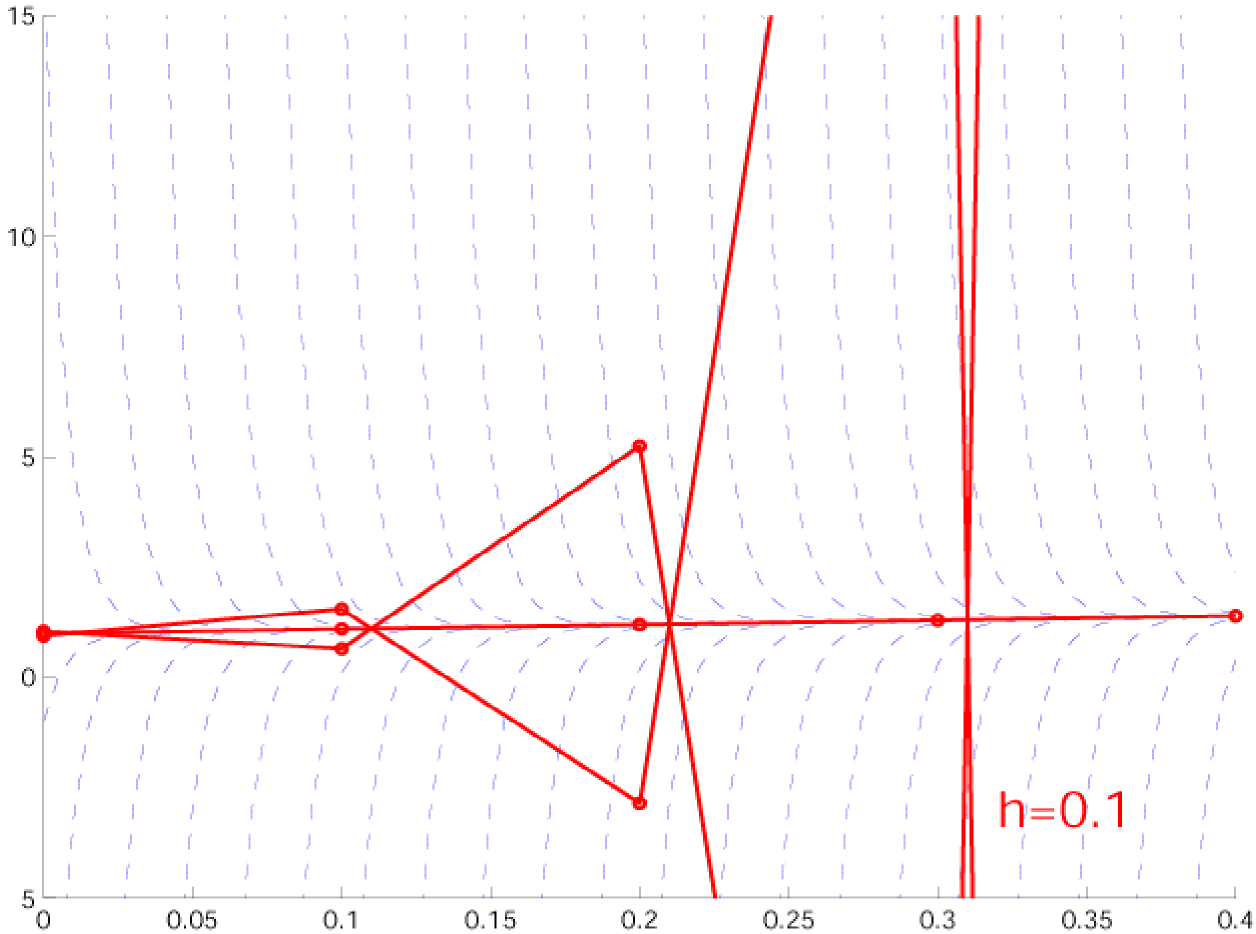
Euler's method requires tiny step



Euler’s method, with larger step

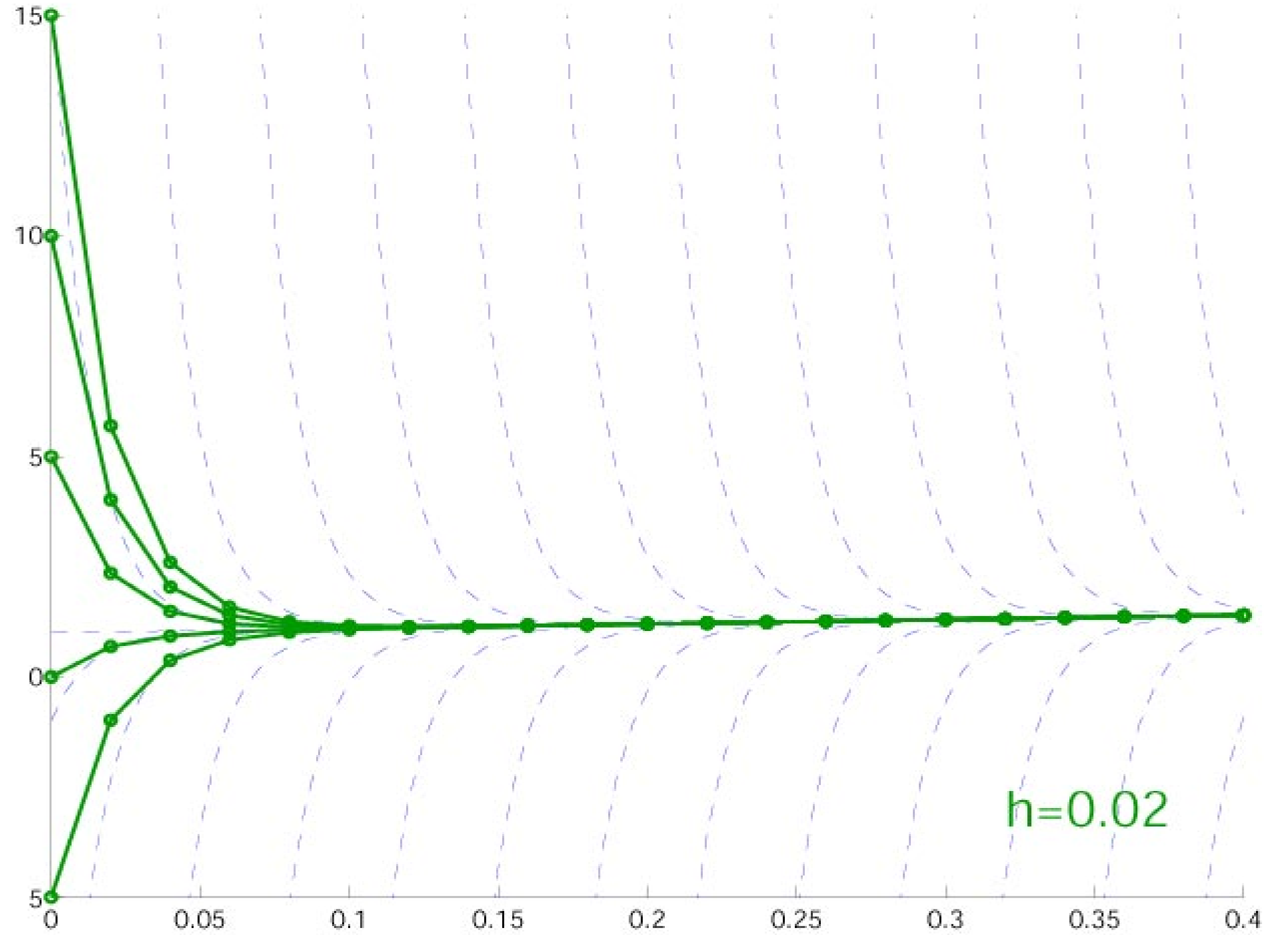


Euler's method with too large a step

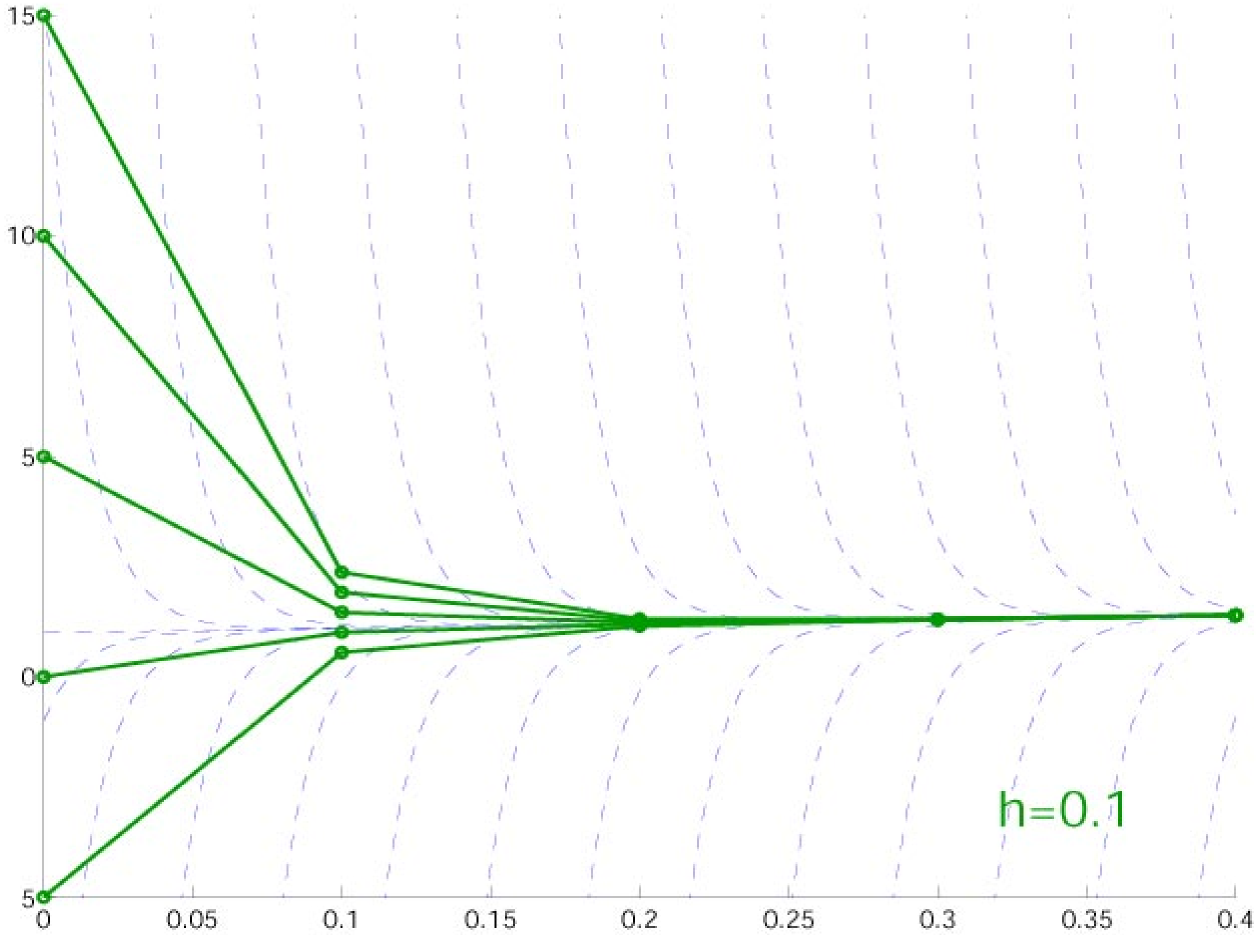


three solutions started at y_0=.99, 1, 1.01

Large steps OK with Backward Euler's method



Very large steps OK, too



Popular IVP Solution Methods

- Euler's method, 1st order
- backward Euler's method, 1st order
- trapezoid method (a.k.a. 2nd order Adams-Moulton)
- 4th order Runge-Kutta

- If a method is p^{th} order accurate then its global error is $O(h^p)$

Matlab code used to make E.M. plots

- ```
  function [tv,yv] = euler(funcname,h,t0,tmax,y0)
  % use Euler's method to solve y'=func(t,y)
  % return tvec and yvec sampled at t=(t0:h:tmax) as col. vectors
  % funcname is a string containing the name of func
  % apparently func has to be in this file??

  % Paul Heckbert           30 Oct 2000

  y = y0;
  tv = [t0];
  yv = [y0];
  for t = t0:h:tmax
      f = eval([funcname '(t,y)']);
      y = y+f*h;
      tv = [tv; t+h];
      yv = [yv; y];
  end

  function f = func1(t,y)   % Heath fig 9.6
  f = y;
  return;

  function f = func2(t,y)   % Heath fig 9.7
  f = -y;
  return;

  function f = func3(t,y)   % Heath example 9.11
  f = -100*y+100*t+101;
  return;
  ```
  ```

Matlab code used to make E.M. plots

- ```
  function e3(h,file)
  figure(4);
  clf;
  hold on;
  tmax = .4;
  axis([0 tmax -5 15]);
  % axis([0 .05 .95 2]);

  % first draw "exact" solution in blue
  y0v = 2.^(0:4:80);
  for y0 = [y0v -y0v]
      [tv,yv] = euler('func3', .005, 0, tmax, y0);
      plot(tv,yv,'b--');
  end

  % then draw approximate solution in red
  for y0 = [(.95:.05:1.05) -5 5 10 15]
      [tv,yv] = euler('func3', h, 0, tmax, y0);
      plot(tv,yv,'ro-', 'LineWidth',2, 'MarkerSize',4);
  end
  text(.32,-3, sprintf('h=%g', h), 'FontSize',20, 'Color','r');

  eval(['print -depsc2 ' file]);
  ```

- run, e.g. `e3(.1, 'a.eps')`
  ```