

Lecture 27: Introduction to ODE Solvers [draft]

5. Numerical Solution of Differential Equations.

The next segment of the course focuses on the numerical approximation of solutions to differential equations. The techniques of interpolation, approximation, and quadrature studied in previous sections are basic tools in your numerical tool chest, ones often used as building blocks to solve more demanding problems. Differential equations play a more central role; their (approximate) solution is required in nearly every corner of physics, chemistry, biology, engineering, finance, and beyond. For many practical problems (involving nonlinearities), there is no way to write down a closed-form solution to differential equations in terms of familiar functions such as polynomials, trigonometric functions, and exponentials. Thus the numerical solution of differential equations is an enormous field, with a great deal of effort in recent decades focused especially on partial differential equations (PDEs). In this course, we only have time to address the numerical solution of ordinary differential equations (ODEs) in detail, though we will briefly address PDEs at the end of this section (and on Problem Set 6).

The subject began in earnest with Leonhard Euler in mid 1700s, with especially important contributions following between 1880 and 1905. The primary motivating application was celestial mechanics, where the approximate integration of Newton's differential equations of motion was needed to predict the orbits of planets and comets. Indeed celestial mechanics (more generally, Hamiltonian systems) motivates modern innovations in so-called geometric/symplectic integrators.

5.0. Introduction to ordinary differential equation (ODE) initial value problems (IVPs).

Before computing numerical solutions to ODEs, it is important to understand the variety of problems that arise, and the theoretical conditions under which a solution exists.

5.0.1. Scalar equations.

A standard scalar initial value problem takes the form

$$\begin{aligned} \text{Given: } & x'(t) = f(t, x), \text{ with } x(t_0) = x_0, \\ \text{Determine: } & x(t) \text{ for all } t \geq t_0. \end{aligned}$$

That is, we are given a formula for the derivative of some unknown function $x(t)$, together with a single value of the function at some *initial time*, t_0 . The goal is to use this information to determine $x(t)$ at all points t beyond the initial time.

Some examples. Differential equations are an inherently graphical subject, so we should look at a few sample problems, together with plots of their solutions.

First we consider the simple, essential model problem

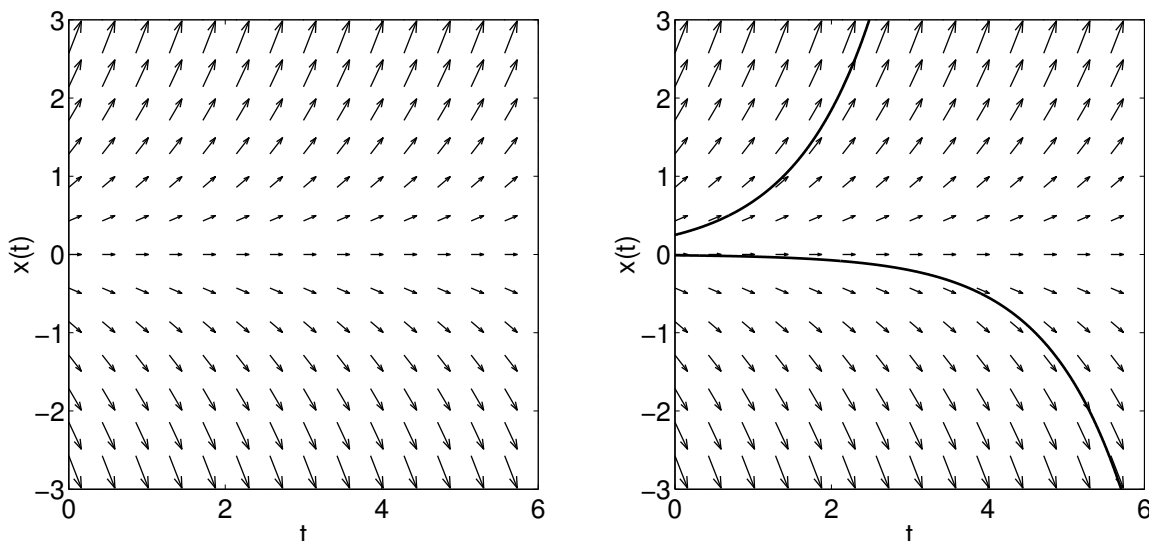
$$x'(t) = \lambda x(t),$$

with exact solution $x(t) = \alpha e^{\lambda t}$, where the constant α is derived from the initial data (t_0, x_0) . If $\lambda > 0$, the solution grows exponentially with t ; $\lambda < 0$ yields exponential decay. Because this linear equation is easy to solve, it provides a good test case for numerical algorithms. Moreover, it is the prototypical linear ODE; from it, we gain insight into the local behavior of nonlinear ODEs.

Applications typically give equations whose solutions cannot be expressed as simply as the solution of this linear model problem. Among the tools that improve our understanding of more

difficult problems is the *direction field* of the function $f(t, x)$, a key technique from the sub-discipline of the *qualitative analysis* of ODEs.[†] To draw a plot of the direction field, let the horizontal axis represent t , and the vertical axis represent x . Then divide the (t, x) plane with regular grid points, $\{(t_j, x_k)\}$. Centered at each grid point, draw a line segment whose slope is $f(t_j, x_k)$. To get a rough impression of the solution of the differential equation $x'(t) = f(t, x)$ with $x(t_0) = x_0$, begin at the point (t_0, x_0) , and follow the direction of the slope lines.

The plot below shows the direction field for $f(t, x) = x$ on the left; on the right, we superimpose solutions of $x'(t) = x$ for several values of $x(0) = x_0$. (The increasing solution has $x(0) = 1/4$, while the decreasing solution has $x(0) = -1/100$.) Notice how the solutions follow the lines in the direction field.



It is a simple matter to compute direction fields in MATLAB using the `quiver` command. For example, the code below produced the plot on the left above.

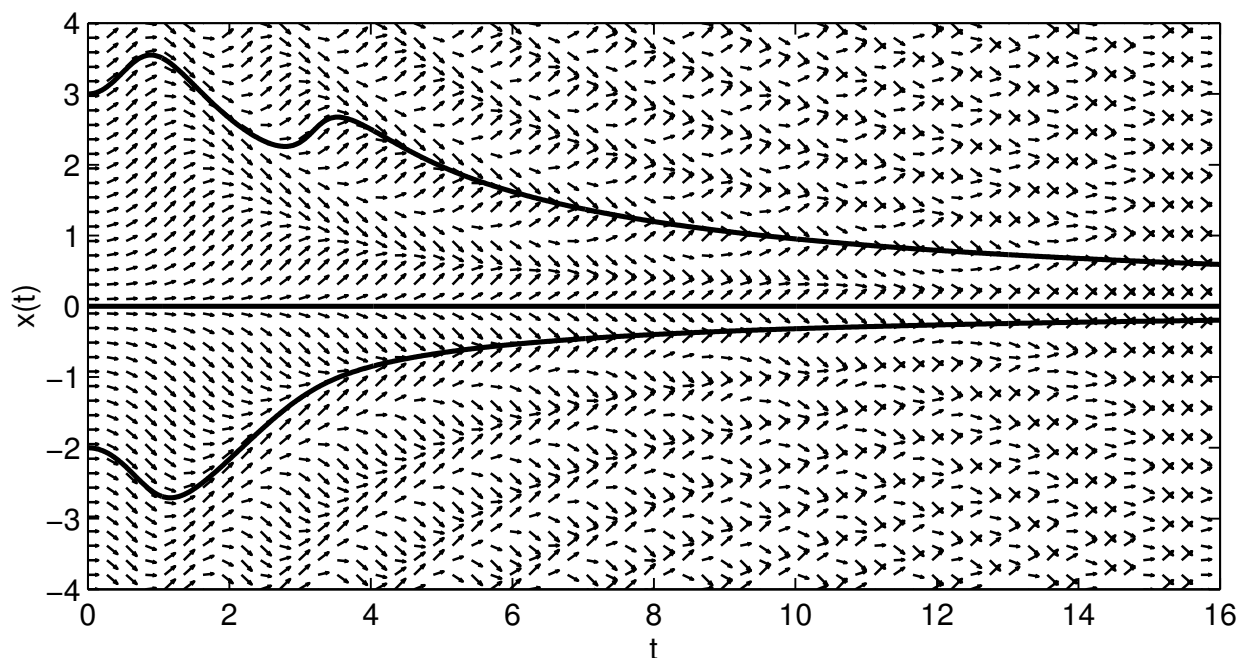
```
f = inline('x','t','x'); % x' = f(t,x)
x = linspace(-3,3,15); t = linspace(0,6,15); % grid of points at which to plot the slope
[T,X] = meshgrid(t,x); % turn grid vectors into matrices
PX = zeros(length(x),length(t));
for j=1:length(x) % compute the slopes
    for k=1:length(t)
        PX(j,k) = f(t(k),x(j));
    end
end
figure(1), clf
quiver(T,X,ones(size(T)),PX), hold on % produce a "quiver" plot
axis equal, axis([min(t) max(t) min(x) max(x)])
set(gca,'fontsize',20)
xlabel('t','fontsize',20)
ylabel('x(t)','fontsize',20)
```

[†]For an elementary introduction to this field, see J. H. Hubbard and B. H. West, *Differential Equations: A Dynamical Systems Approach, Part I*, Springer-Verlag, 1991.

Next consider an equation that lacks an elementary solution that can be expressed in closed form,

$$x'(t) = \sin(xt).$$

The direction field for $\sin(xt)$ is shown below. Though we don't have access to the exact solution, it is a simple matter to compute accurate approximations. Several such solutions (for $x(0) = 3$, $x(0) = 0$, and $x(0) = -2$ are superimposed on the direction field. These were computed using Euler's method, which we will discuss momentarily. (Those areas, mainly in the right side of the direction field, where it appears that down and up arrows cross, are asymptotes of the solution: between the up and down arrow is a point where the slope $f(t, x)$ is zero.)



5.0.2. Systems of equations.

In most applications we do not have a simple scalar equation, but rather a system of equations describing the coupled dynamics of several variables. Such situations give rise to vector-valued functions $\mathbf{x}(t) \in \mathbb{R}^n$. In particular, the initial value problem becomes

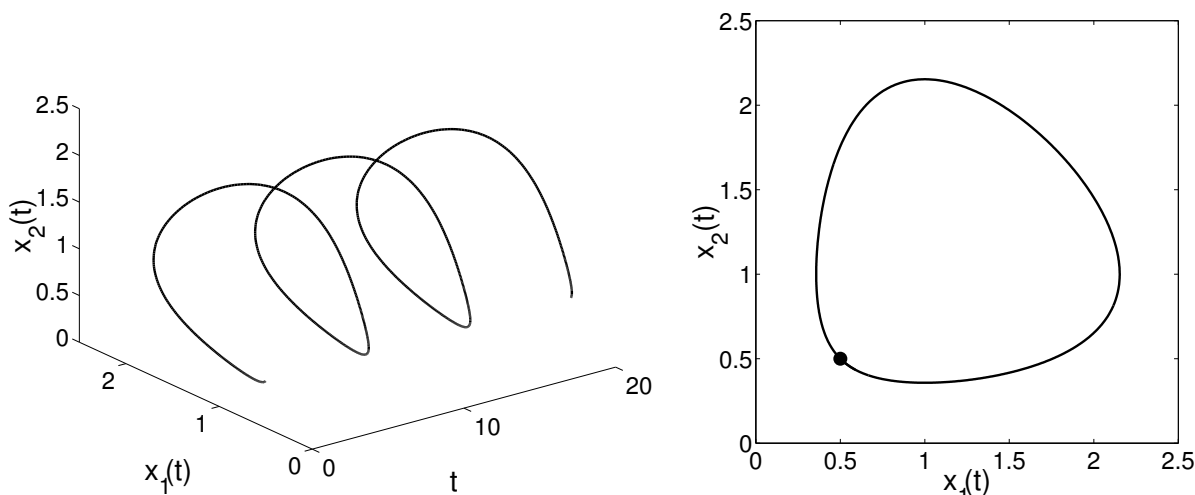
$$\begin{aligned} \text{Given: } & \mathbf{x}'(t) = \mathbf{f}(t, \mathbf{x}), \text{ with } \mathbf{x}(t_0) = \mathbf{x}_0, \\ \text{Determine: } & \mathbf{x}(t) \text{ for all } t \geq t_0. \end{aligned}$$

All the techniques for solving scalar initial value problems described in this course can be applied to systems of this type.

An example. Many nonlinear systems of ordinary differential equations give rise to fascinating behavior, widely studied in the field of *dynamical systems*. Many interesting systems are *autonomous* differential equations, meaning that the time variable t does not explicitly appear in the function \mathbf{f} . For example, the Lotka–Volterra predator–prey equations are given by

$$\mathbf{x}' = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} x_1 - x_1x_2 \\ -x_2 + x_1x_2 \end{bmatrix} = \mathbf{f}(t, \mathbf{x}).$$

Here x_1 represents the population of prey (e.g., gazelles), while x_2 denotes the population of predators (e.g., lions). This system exhibits cyclic behavior, with the populations oscillating regularly in time. The plots below illustrate one such periodic solution based on the initial condition $x_1(0) = x_2(0) = 1/2$. The left plot shows how x_1 and x_2 evolve as a function of time. The *phase space* is shown on the right. For this plot, we stare down the time axis, seeing how x_1 and x_2 relate independent of time. The black dot denotes the system's position at $t = 0$.



5.0.3. Higher-order ODEs.

Many important ODEs arise from Newton's Second Law, $\mathbf{F}(t) = m\mathbf{a}(t)$. Noting the acceleration $\mathbf{a}(t)$ is the second derivative of position, we arrive at

$$\mathbf{x}''(t) = m^{-1}\mathbf{F}(t).$$

Thus, we are often interested in systems of higher-order ODEs.

To keep the notation simple, consider the scalar second-order problem

$$\text{Given: } x''(t) = f(t, x, x'), \text{ with } x(t_0) = x_0, x'(t_0) = y_0$$

$$\text{Determine: } x(t) \text{ for all } t \geq t_0.$$

Note, in particular, that the initial conditions $x(t_0)$ and $x'(t_0)$ must both be supplied.[‡]

This second order equation (and higher-order ODE's as well) can always be written as a first order system of equations. Define $x_1(t) = x(t)$, and let $x_2(t) = x'(t)$. Then

$$\begin{aligned} x_1'(t) &= x'(t) = x_2(t) \\ x_2'(t) &= x''(t) = f(t, x, x') = f(t, x_1, x_2). \end{aligned}$$

Writing this in vector form, $\mathbf{x} = [x_1 \ x_2]^T$, and the differential equation becomes[§]

$$\mathbf{x}'(t) = \begin{bmatrix} x_1'(t) \\ x_2'(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ f(t, x_1, x_2) \end{bmatrix} = \mathbf{f}(t, \mathbf{x}).$$

[‡]Problems where the initial data is given at two different t points are called *boundary value problems*. We will study them in Section 5.4.

[§]Note that fonts are important here: $x(t)$ is a scalar quantity, while $\mathbf{x}(t)$ is a vector.

The initial value is given by

$$\mathbf{x}_0 = \begin{bmatrix} x_1(t_0) \\ x_2(t_0) \end{bmatrix} = \begin{bmatrix} x(t_0) \\ x'(t_0) \end{bmatrix}.$$

An example. The most famous second order differential equation,

$$x''(t) = -x(t),$$

has the solution $x(t) = \alpha \cos(t) + \beta \sin(t)$, for constants α and β depending upon the initial values. This gives rise to the system

$$\mathbf{x}' = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} x_2 \\ -x_1 \end{bmatrix}.$$

When we combine Newton's inverse-square description of gravitational force with his Second Law, we obtain the system of second order ODEs

$$\mathbf{x}''(t) = \frac{-\mathbf{x}(t)}{\|\mathbf{x}(t)\|_2^3},$$

where $\mathbf{x} \in \mathbb{R}^3$ is a vector in Euclidean space, and the 2-norm denotes the usual (Euclidean) length of a vector,

$$\|\mathbf{x}(t)\|_2 = (x_1(t)^2 + x_2(t)^2 + x_3(t)^2)^{1/2}.$$

Since $\mathbf{x}(t) \in \mathbb{R}^3$, this second order equation reduces to a system of *six* first order equations.

5.0.4. Picard's Theorem: Existence and Uniqueness of Solutions.

Before constructing numerical solutions to these differential equations, it is important to understand when solutions exist at all. Picard's theorem establishes existence and uniqueness. For a proof, see Süli and Mayers, Section 12.1.

Theorem (Picard's Theorem). Let $f(t, x)$ be a continuous function on the rectangle

$$D = \{(t, x) : t \in [t_0, t_{\text{final}}], x \in [x_0 - c, x_0 + c]\}$$

for some fixed $c > 0$. Furthermore, suppose $|f(t, x_0)| \leq K$ for all $t \in [t_0, t_{\text{final}}]$, and suppose there exists some *Lipschitz constant* $L > 0$ such that

$$|f(t, u) - f(t, v)| \leq L|u - v|$$

for all $u, v \in [x_0 - c, x_0 + c]$ and all $t \in [t_0, t_{\text{final}}]$. Finally, suppose that

$$c \geq \frac{K}{L} \left(e^{L(t_{\text{final}} - t_0)} - 1 \right).$$

(That is, the box D must be sufficiently large to compensate for large values of K and L .) Then there *exists a unique* $x \in C^1[t_0, t_{\text{final}}]$ such that $x(t_0) = x_0$, $x'(t) = f(t, x)$ for all $t \in [t_0, t_{\text{final}}]$, and $|x(t) - x_0| \leq c$ for all $t \in [t_0, t_{\text{final}}]$.

In simpler words, these hypotheses ensure the existence of a unique C^1 solution to the initial value problem, and this solution stays within the rectangle D for all $t \in [t_0, t_{\text{final}}]$.

5.1. One-step methods.

Finally, we are prepared to discuss some numerical methods to approximate the solution to all these ODEs. To simplify the notation, we present our methods in the context of the scalar equation

$$x'(t) = f(t, x)$$

with the initial condition $x(t_0) = x_0$. All the algorithms generalize trivially to systems: simply replace scalars with vectors.

When computing approximate solutions to the initial value problem, we will not obtain the solution for every value of $t > t_0$, but only on a discrete grid.[¶] In particular, we will generate approximate solutions at some regular grid of time steps

$$t_k = t_0 + kh$$

for some constant *step-size* h . (Actually, the methods we consider in this subsection actually allow h to change with each step size, so one actually has $t_k = t_{k-1} + h_k$. For simplicity of notation, we will assume for now that the step-size is fixed.)

The approximation to x at the time t_k is denoted by x_k , so hopefully

$$x_k \approx x(t_k).$$

Of course, the initial data is exact:

$$x_0 = x(t_0).$$

5.1.1. Euler's method.

We need some approximation that will advance from the exact point on the solution curve, (t_0, x_0) to time t_1 . Recall from introductory calculus that

$$x'(t) = \lim_{h \rightarrow 0} \frac{x(t+h) - x(t)}{h}.$$

This definition of the derivative inspires our first method. Apply it at time t_0 with our small but finite time step h to obtain

$$x'(t_0) \approx \frac{x(t_0+h) - x(t_0)}{h}.$$

Since $x'(t_0) = f(t_0, x(t_0)) = f(t_0, x_0)$, we have access to the quantity on the left hand side of this approximation. The only quantity we don't know is $x(t_0+h) = x(t_1)$. Rearranging the above to put $x(t_1)$ on the left hand side, we obtain

$$x(t_1) \approx x(t_0) + hx'(t_0) = x_0 + hf(t_0, x_0).$$

This approximation is precisely the one suggested by the direction field discussion in §5.0.1. There, to progress from the starting point (t_0, x_0) , we followed the line of slope $f(t_0, x_0)$ some distance,

[¶]The field of *asymptotic analysis* delivers approximations in terms of elementary functions that can be highly accurate; these are typically derived in a non-numerical fashion, and often have the virtue of accurately identifying leading order behavior of complicated solutions. For a beautiful introduction to this important area of applied mathematics, see Carl M. Bender and Seven A. Orszag, *Advanced Mathematical Methods for Scientists and Engineers*; McGraw-Hill, 1978; Springer, 1999.

which in the present context is our step size, h . To progress from the new point, (t_1, x_1) , we follow a new slope, $f(t_1, x_1)$, giving the iteration

$$x_2 = x_1 + hf(t_1, x_1).$$

There is an important distinction here. Ideally, we would have derived our value of $x_2 \approx x(t_2)$ from the formula

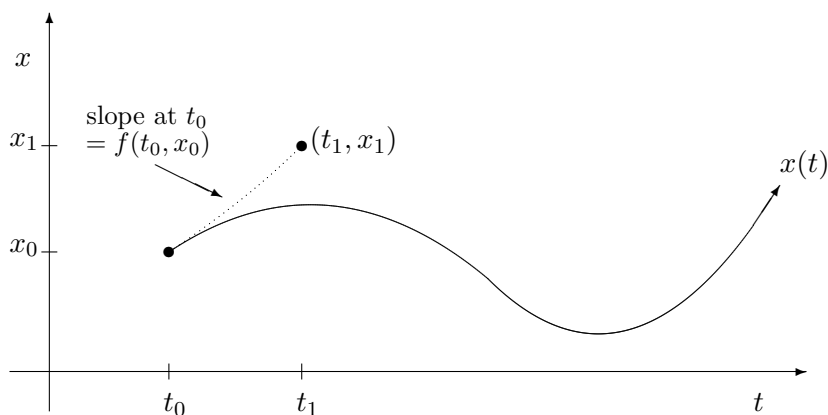
$$x(t_2) \approx x(t_1) + hf(t_1, x(t_1)).$$

However, an error was made in the computation of $x_1 \approx x(t_1)$; we do not have access to the exact value $x(t_1)$. Thus, compute x_2 from x_1 , the quantity we have access to. This might seem like a minor distinction, but in general the difference between the approximation x_k and the true solution $x(t_k)$ is vital. At each step, a *local error* is made due to the approximation of the derivative by a line. These local errors accumulate, giving *global error*. Is a small local error enough to ensure small global error? This question is the subject of the next two lectures.

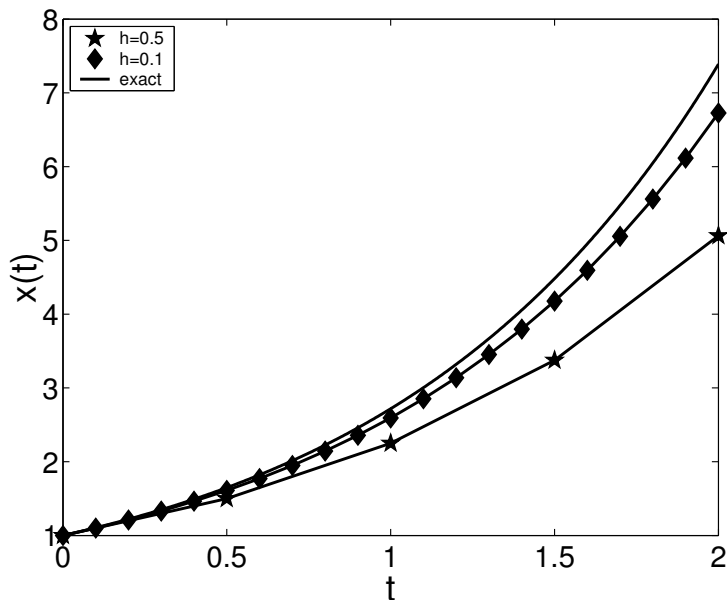
Given the approximation x_2 , repeat the same procedure to obtain x_3 , and so on. Formally,

$$\text{Euler's Method:} \quad x_{k+1} = x_k + hf(t_k, x_k).$$

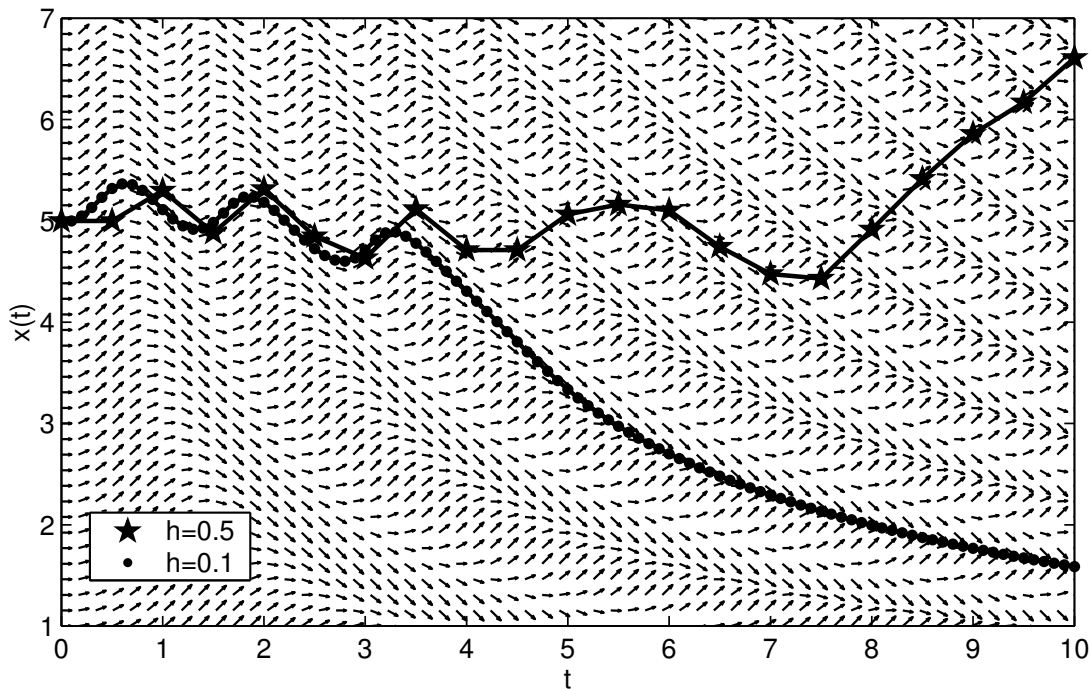
The first step of Euler's method is illustrated in the following schematic.



Examples of Euler's method. Consider the performance of Euler's method on the two examples for §5.0.1. First, we examine the equation, $x'(t) = x(t)$, with initial condition $x(0) = 1$. We apply two step sizes: $h = 0.5$ and $h = 0.1$. Naturally, we expect that decreasing h will deliver improved local accuracy. But with $h = 0.1$, we require five times as many approximations as with $h = 0.5$. How do the errors made at these many steps accumulate? We see in the plot below that both approximations underestimate the true solution, but that indeed, the smaller step size yields the better approximation.



Next, consider our second example, $x'(t) = \sin(tx)$, this time with $x(0) = 5$. Since we do not know the exact solution, we can only compare approximate answers, here obtained with $h = 0.5$ and $h = 0.1$. For $t > 4$, the solutions completely differ from one another! Again, the smaller step size is the more accurate solution. In the plot below, the direction field is shown together with the approximate solutions. Note that $f(t, x) = \sin(tx)$ varies with x , so when the $h = 0.5$ solution diverges from the $h = 0.1$ solution, very different values of f are used to generate iterates. The $h = 0.5$ solution ‘jumps’ over the correct asymptote, and provides a very misleading answer.



For a final example of Euler's method, consider the equation

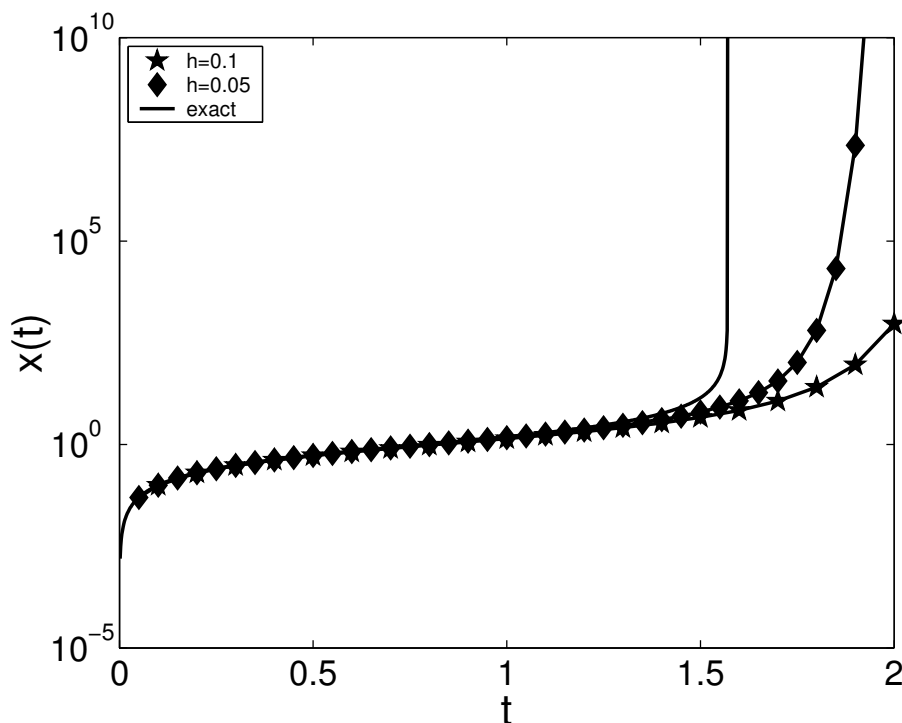
$$x'(t) = 1 + x(t)^2$$

with $x(0) = 0$.^{||} This equation looks innocuous enough; indeed, you might notice that the exact solution is $x(t) = \tan(t)$. The true solution blows up *in finite time*, $x(t) \rightarrow \infty$ as $t \rightarrow \pi/2$. (Such blow-up behavior is common in ODEs and PDEs where the formula for the derivative of x involves higher powers of x .) It is reasonable to seek an approximate solution to the differential equation for $t \in [0, \pi/2)$, but beyond $t = \pi/2$, the equation does not have a solution, and any answer produced by our numerical method is, essentially, garbage.

For any finite x , $f(t, x) = 1 + x^2$ will always be finite. Thus Euler's method,

$$\begin{aligned} x_{k+1} &= x_k + hf(t_k, x_k) \\ &= h + x_k(1 + hx_k) \end{aligned}$$

will always produce some finite quantity; it will never give the infinite answer at $t = \pi/2$. Still, as we see in the plots below, Euler's method captures the qualitative behavior well, with the iterates growing very large soon after $t = \pi/2$. (Notice that the vertical axis is logarithmic, so by $t = 2$, the approximation with time step $h = 0.05$ exceeds 10^{10} .)



Below, we present MATLAB code to implement Euler's method, with sample code illustrating how to call the routine for the second example, $x'(t) = \sin(tx)$.

^{||}This example is given in Kincaid and Cheney, page 525.

```

function [t,x] = euler(xprime, tspan, x0, h)

% function [t,x] = euler(xprime, [t0 tfinal], x0, h)
% Approximate the solution to the ODE:  $x'(t) = xprime(x,t)$ 
% from  $t=t_0$  to  $t=tfinal$  with initial condition  $x(t_0)=x_0$ .
% xprime should be a function that can be called like: xprime(t,x).
% h is the step size: reduce h to improve the accuracy.
% The ODE can be a scalar or vector equation.

t0 = tspan(1); tfinal = tspan(end);

% set up the t values at which we will approximate the solution
t=[t0:h:tfinal];

% include tfinal even if h does not evenly divide tfinal-t0
if t(end)~=tfinal, t = [t tfinal]; end

% execute Euler's method
x = [x0 zeros(length(x0),length(t)-1)];
for j=1:length(t)-1
    x(:,j+1) = x(:,j) + h*feval(xprime,t(j),x(:,j));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Sample code to call euler.m for the equation  $x'(t) = \sin(t*x)$ .
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xprime = inline('sin(x*t)', 't', 'x');      %  $x'(t) = \sin(t*x)$ 
tspan = [0 10];                          % integrate from  $t=0$  to  $t=10$ 
x0 = 5;                                    % start with  $x(0) = 5$ 
h = 0.1;                                   % time step
[t,x] = euler(xprime, tspan, x0, h);      % call Euler
figure(1), clf
plot(t,x,'b.', 'markersize', 15)         % plot output

```