Today’s goals

1. Qualitative methods in ODEs
2. Euler methods and their graphical interpretation
3. Constants of the motion

Physics has sometimes been called the *science of differential equations*.

Most of the questions physics seeks to answer are posed in the form of *differential equations*.

1. *Ordinary differential equations*, ODEs (this section of the course)
2. *Partial differential equations*, PDEs to be covered later in the course.

We begin our study of *ODEs* by looking at first order *ODEs, initial value problems*. These have the form

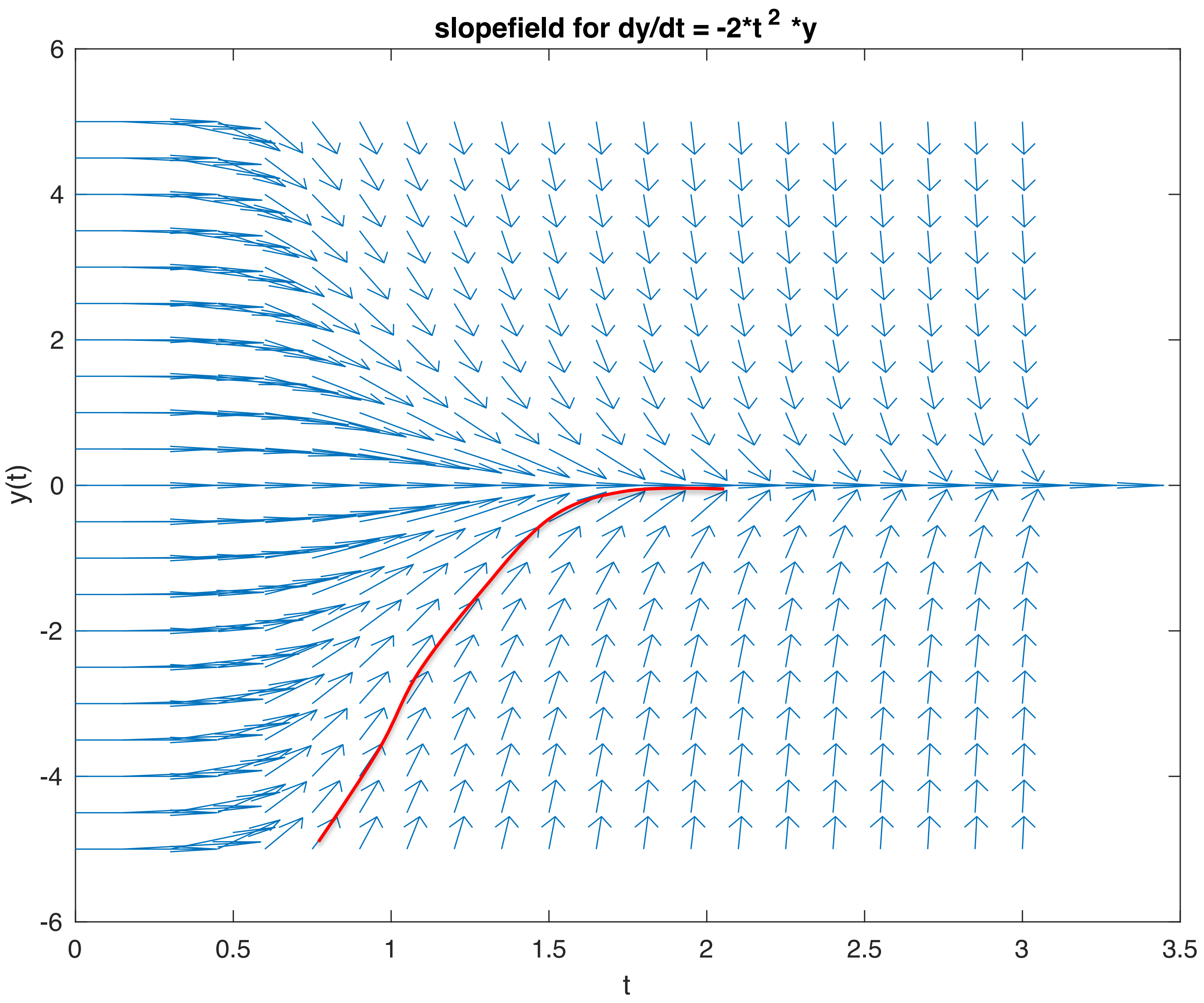
$$\dot{y} \equiv \frac{dy}{dt} = f(y,t); \quad y(t_o) = y_o \qquad (1)$$

The *dot-notation* for the *time derivative* is common in physics*. Spatial derivatives* are sometimes expressed using the ' notation, i.e. *$y' = dy/dx$.* The auxiliary condition, *$y(t_o) = y_o$* is called the *initial condition*.

Equation (1) says that the *derivative* of the dependent variable, *y,* is given by the function, *$f(y,t)$.*

But of course, we are not interested in the *derivative of y, but in y.* Thus we must extract *y* from this derivative equation. Because the integral is sometimes the *anti-derivative*, extracting *y* is sometimes referred to as finding *the first integral*

Let's unpack more what Eq. (1) is saying. Do question 1 on the worksheet and S T O P



slopefield for dy/dt = -2*t ^2 *y
6
4
2
0
-2
-4
-6
y(t)
0
0.5
1
1.5
2
2.5
3
3.5
t

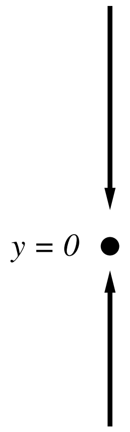
```
function  dirfield(tmin,tmax,ymin,ymax,n)
 % function plots direction field.
% Inputs are the min and max values for meshgrid, and n, the
% number of points between the min and max values

delt = (tmax-tmin)/n; % t-step size
dely = (ymax-ymin)/n; % y-step size
[t,y] = meshgrid(tmin:delt:tmax,ymin:dely:ymax);
dy = function goes here;
dt=ones(size(dy));
dtu = dt./sqrt(dt.^2 + dy.^2);
dyu= dy./sqrt(dt.^2 + dy.^2);
quiver(t,y,dtu,dyu)
xlabel('t')
ylabel('y(t)')
title('slopefield for dy/dt = -2*t^2 *y')

end
```

Note the point *y = 0* is a special point. Once the derivative reaches 0, the value of y(t) never changes. This kind of point is called an *equilibrium point* and they come in 3 types.

1. *Stable equilibrium* points are equilibrium points in which solutions near the point tend to head *towards t*he equilibrium point.
2. *Unstable equilibrium* points are equilibrium points in which solutions near the point tend to *head away* from the equilibrium points.
3. *Saddle points* are equilibrium points in which solutions *head to the* equilibrium point from *one dire*ction, and *away from* the equilibrium point in the *other direction.*

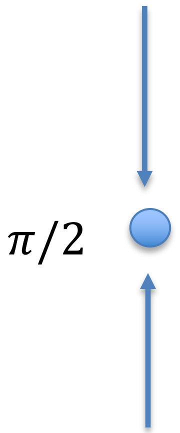
Eq. (1) is 0 when either *y* or *t* are 0. The case of *t = 0* is unrealistic since it means we haven't yet *started* the system. So let's focus on the equilibrium point at *(t, y = 0)*

$$\begin{aligned} y = 0.01 \quad &\Rightarrow \quad \dot{y} = -2(t^2)(0.01) < 0 \\ y = -0.01 \quad &\Rightarrow \quad \dot{y} = -2(t^2)(-0.01) > 0 \end{aligned}$$



Phase line

Do questions (2) and (3) on the worksheet and S T O P


$\pi/2$

(3) The differential equation is of the form

dy/dt = f(t) g(y).

This type of ODE is call *separable,* and it is straightforward to solve as

$$\begin{aligned}\frac{dy}{dt} &= f(t)g(y)\\ \frac{dy}{g(y)} &= f(t)dt\\ \int\frac{dy}{g(y)} &= \int f(t)dt\end{aligned}$$

Most ODES cannot be integrated and so we must resort to approximation techniques.

In question (1), you exploited the fact that geometrically, the first derivative is the slope at a point. We will use this to develop our first algorithm for solving ODEs numerically

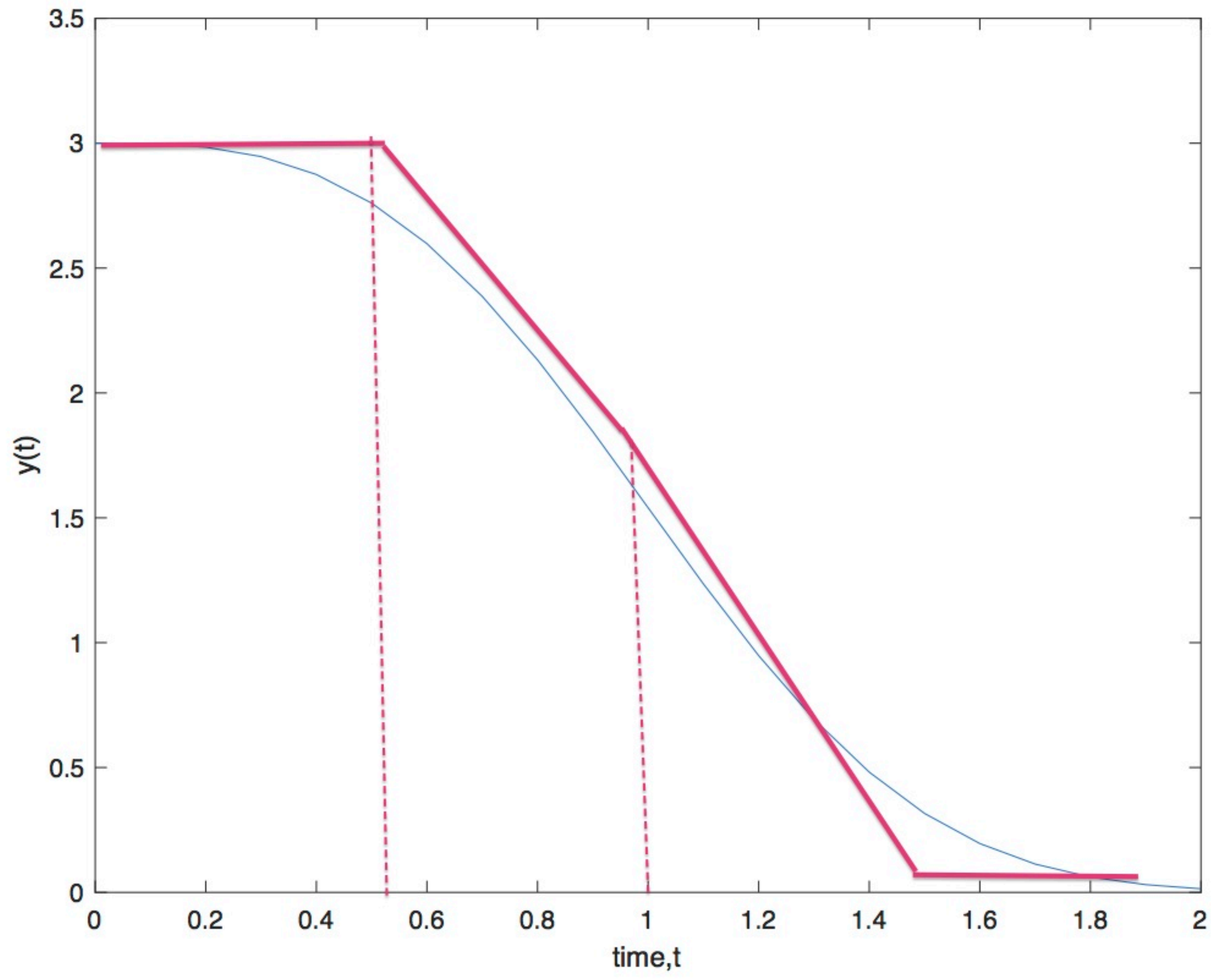
Do question (4—6) on the worksheet and S T O P

(4)

$$\begin{aligned}\frac{dy}{dy} &\approx \frac{\Delta y}{\Delta t} = f(t,y)\\ y_f - y_i &= f(t_i, y_i)\Delta t\\ y_f &= y_i + f(t_i, y_i)\Delta t\end{aligned}$$

i	t_i	y_i	$y_{i+1} = y_i - (\Delta t \cdot 2t^2 y)$
0	0	3	3
1	0.5	3	2.25
2	1	2.25	0
3	1.5	0	0

(4)



(6) The deviation starts almost immediately, although it is not significant at first. The deviation occurs because the slope between points t_i and $t_i + \Delta t$ is the slope at t_i . A better approximation to the ‘real’ solution could be found by decreasing the size of Δt.

A MatLab code that implements Euler’s method is

```
function myodesolver(to,yo,h,tf)
        nsteps = round((tf - to)/h);
        t = zeros(nsteps,1);
        y = zeros(nsteps,1);
        t(1) = to;
        y(1) = yo;
        for n = 1:nsteps
                y(n+1) = myeuler(t(n),y(n),h)
                t(n+1+) = t(n) + h
        end
        plot(t,y,’.k’,’MarkerSize’,18)
end
function yprime=myeuler(tn,yn,h)
        yprime = (yn-2*yn*tn^2*h). % function goes here
end
```

Euler's method basically approximates derivatives at a point by slopes of a secant line between two nearby points. Another view of Euler's method is as a Taylor expansion.

Recall that we have, $\frac{dy}{dt} = f(t, y)$ and we wish to solve for ***y(t)*** given an initial condition

If we Taylor expand the unknown ***y(t)*** about a point ***t*** we have,

$$y(t) = y(t_o) + (t - t_o)\frac{dy(t_o, y)}{dt} + \text{higher order terms}$$

But,

$$\frac{dy(t_o, y)}{dt} = f(t_o, y)$$

So $y(t) \approx y(t_o) + (t - t_o)f(t_o y)$ which is the same equation we obtained by approximating slopes at a point by slopes of secant lines.

What is the error in each step in the Euler's method?

We define the local truncation error, the error at each step as follows.

The solution given by one step of Euler's methods is:

$$y_1 = y_o + hf(t_o, y_o) \quad h = \Delta t$$

The exact solution we don't know, but we can Taylor expand *y(t)*, about the point, t_o to find that

$$y(t_o + h) = y(t_o) + hy'(t_o) + \frac{1}{2}h^2 y''(t_o) + \mathcal{O}(h^3)$$

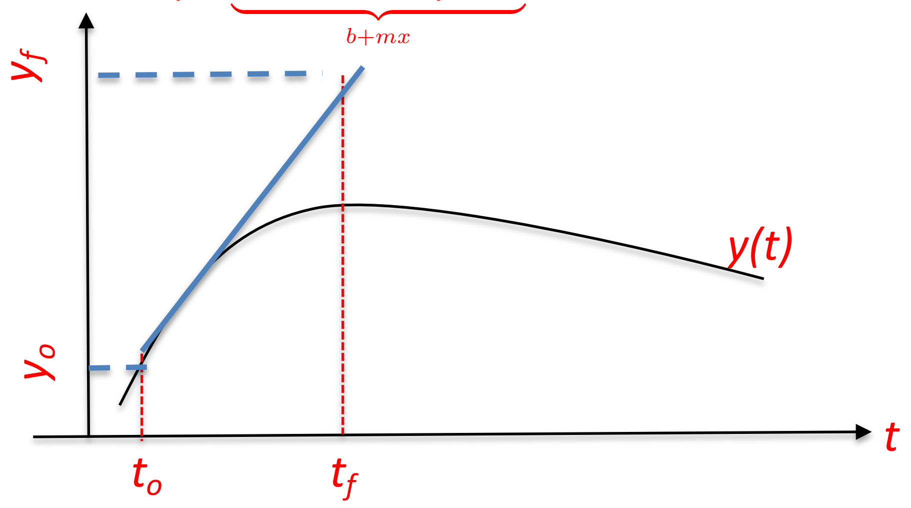
Then the local truncation error is $y - y_1$ or

$$\text{LTE} = y(t_o + h) - y_1 = \frac{1}{2}h^2 y''(t_o) + \mathcal{O}(h^3)$$

Careful. This is **not** the error between the ``real" solution and the Euler solution. It is the error caused by truncating the Taylor series after the linear term.

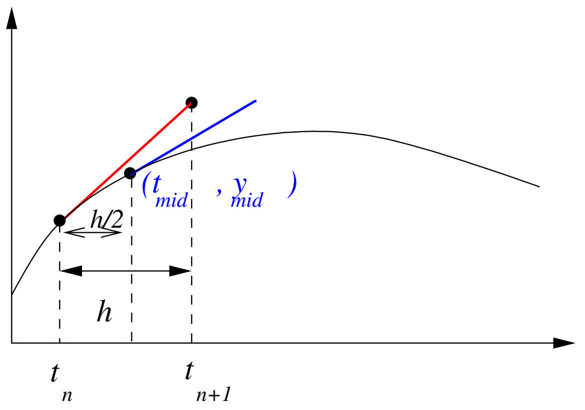
Euler's method gives the solution one step at a time.

$$y_f \approx \underbrace{y_o + f(t_o, y_o)\,(t_f - t_o)}_{b+mx}$$



As the figure shows, Euler's method uses a line of constant slope that passes through the point t_o to make the prediction at what happens at point t_f. Lots of ways to think about doing this, we'll introduce one called the *mid-point method.*

In this method we approximate y_f by using the midpoint between the interval (as found using Euler's method) instead of y_o.



Mid-point method

1. First, find the value of *y* halfway across the interval using the derivative to find its value. That is,

$$y_{mid} = y_o + f(t_o, y_o) \cdot \frac{\Delta t}{2}$$

2. Use this value to find the value of *y* at ***t+ Δt.*** That is,

$$y_{n+1} = y_n + f(t_{mid}, y_{mid}) \cdot \Delta t$$

Improved Euler:

1. Evolve the system using the normal Euler method
2. Substitute the new y to find the derivative at the end of the interval.
3. Average of the derivative at the beginning and end of the intervals to step the system forward

$$y_{f,temp} = y_o + hf(t_o, y_o) \quad \text{so that}$$

$$f(t_o + h, y_f) = f(t_o + h, \underbrace{y_o + hf(t_o, y_o)}_{y_{f,temp}})$$

$$y_f = y_o + h\left[\frac{f(t_o, y_o) + f(t_o + h, y_o + hf(t_o, y_o))}{2}\right]$$

Do question (7) on the worksheet and S T O P

The mid point method used one intermediate function to estimate the derivative at the next time step.

But why stop at one? Runge-Kutta methods use more intermediate functions to better estimate the derivative at the next time step. The most popular is the 4th order Runge-Kutta method which has the form:

$$y(t_o + h) = y(t_o) + \frac{h}{6}\left(f_o + 2f_1 + 2f_2 + f3\right)$$

where

$$\begin{aligned} f_o &= f(t_o, y_o) \\ f_1 &= f\left(t_o + \frac{h}{2}, y_o + \frac{h}{2} f_o\right) \\ f_2 &= f\left(t_o + \frac{h}{2}, y_o + \frac{h}{2} f_1\right) \\ f_3 &= f\left(t_o + h, y_o + h f_2\right) \end{aligned}$$

Do question (8) on the worksheet and S T O P

ODEs—01

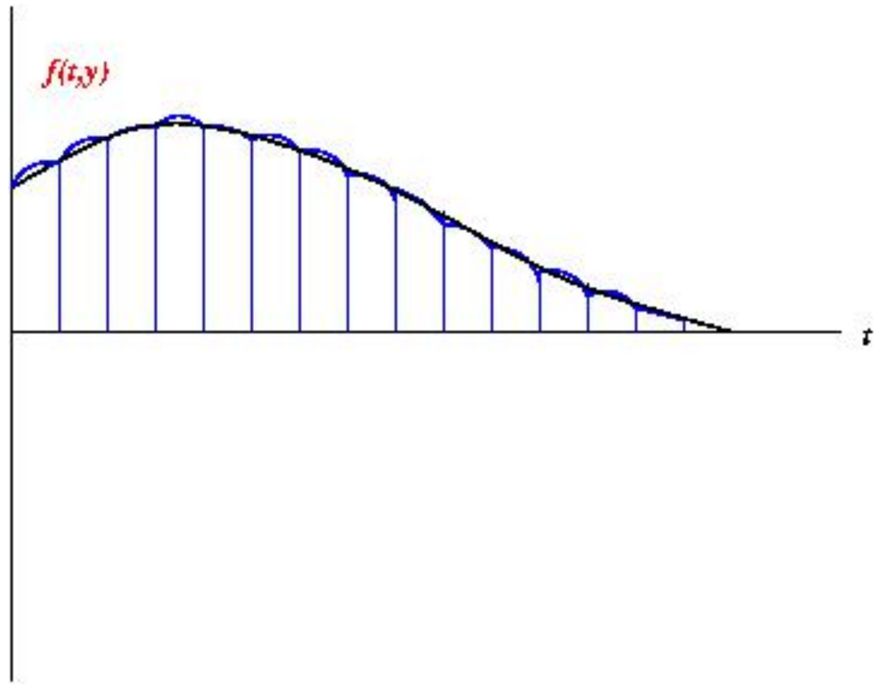
```
function myodesolver(to,yo,h,tf,alg)
%driver program to solve odes, alg = 0 euler,
% alg =1 modified euler, else RK4
        nsteps = round((tf - to)/h);
        t = zeros(nsteps,1);
        y = zeros(nsteps,1);
        t(1) = to;
        y(1) = yo;
        for n = 1:nsteps
           if alg == 0
                y(n+1) = myeuler(t(n),y(n),h);
           elseif alg == 1
                y(n+1) = myeuler_mod(t(n),y(n),h)
           else
                y(n+1) = myrk4(t(n),y(n),h);
           end
           t(n+1) = t(n) + h;
        end
        plot(t,y,'.k','MarkerSize',18)
end
```

```
function yprime=myrk4(tn,yn,h)
%myrk4 Runge-Kutta routine
% uses anonymous function
        RHS=@(t,y) (-2*y*t^2);
        fo=RHS(tn,yn);
        f1=RHS(tn+h/2,yn+h/2*fo);
        f2=RHS(tn+h/2,yn+h/2*f1);
        f3=RHS(tn+h,yn+h*f2);
        yprime=yn +h/6*(fo+2*f1+2*f1+f3);
end
```

Euler's method

Modified Euler

Runge-Kutta



Do question 9 on the worksheet