An Ode to MATLAB's ode45

Background

```
>> help ode45

 ode45  Solve non-stiff differential equations, medium order method.
    [TOUT,YOUT] = ode45(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates
    the system of differential equations y' = f(t,y) from time T0 to TFINAL
    with initial conditions Y0. ODEFUN is a function handle. For a scalar T
    and a vector Y, ODEFUN(T,Y) must return a column vector corresponding
    to f(t,y). Each row in the solution array YOUT corresponds to a time
    returned in the column vector TOUT.  To obtain solutions at specific
    times T0,T1,...,TFINAL (all increasing or all decreasing), use TSPAN =
    [T0 T1 ... TFINAL].
```

This[1] is the first paragraph of the `MATLAB` help for the built-in function `ode45`. Let's parse this "help".

- What does `ode45` do? The built-in function `ode45` solves non-stiff, first order, differential equations, using a medium order method.
- What is the *application programming interface*[2] (API) for `ode45`? In other words, what does a call to `ode45` look like?

  `[TOUT,YOUT] = ode45(ODEFUN,TSPAN,Y0)`

- What are the input arguments?
  - `ODEFUN` is a function handle. For a scalar `T` and a vector `Y`, `ODEFUN(T,Y)` must return a column vector corresponding to $f(t, y)$. Thus, the *application programming interface* for `ODEFUN` would look like

    `[dYdt] = ODEFUN(T,Y)`

    Note that the user-supplied `ODEFUN` must take exactly two arguments: a scalar `T` and a vector `Y`. Why a "vector" `Y`? Because `ode45` will solve a single first-order differential equation, or a system of first-order differential equations. If you are solving a system of m first-order differential equations then input argument `Y` is an $(m \times 1)$ matrix

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{1}$$

[1]For those of you who were wondering, the "ode" is a real word in the English language. *"Ode is a type of lyrical stanza. A classic ode is structured in three major parts: the strophe, the antistrophe, and the epode. Different forms such as the homostrophic ode and the irregular ode also exist. It is an elaborately structured poem praising or glorifying an event or individual, describing nature intellectually as well as emotionally."* [`http://en.wikipedia.org/wiki/Ode`, visited 8 March 2013.]

[2]See, for example, `http://en.wikipedia.org/wiki/Application_programming_interface`.

The user-supplied function `ODEFUN` must return a vector of first derivatives. If you are solving a single differential equation, `ODEFUN` returns a scalar. If you are solving a system of m differential equations, `ODEFUN` must return an $(m \times 1)$ matrix

$$\textbf{dYdt} = \begin{bmatrix} \frac{dy_1}{dt} \\ \frac{dy_2}{dt} \\ \vdots \\ \frac{dy_m}{dt} \end{bmatrix} \tag{2}$$

Note that your user-defined input function `ODEFUN` does not have to be called `ODEFUN`, since you are passing in a function handle.

- `TSPAN` is a row matrix of times. When `TSPAN` is equal to a (1×2) matrix `[T0 TFINAL]`, `ode45` integrates the system of differential equations $dy/dt = f(t, y)$ from time `T0` to `TFINAL`.

  To obtain solutions at specific times `T0,T1,...,TFINAL` (all increasing or all decreasing), use `TSPAN` equal to `[T0 T1 ... TFINAL]`.
- `Y0` is the initial condition. If you are solving a single differential equation, `Y0` is a scalar. If you are solving a system of m differential equations, `Y0` is a $(1 \times m)$ matrix of initial conditions.

$$\texttt{Y0} = \begin{bmatrix} y_1(\texttt{T0}) & y_2(\texttt{T0}) & \dots & y_m(\texttt{T0}) \end{bmatrix} \tag{3}$$

- What does `ode45` return? Each row in the solution array `YOUT` corresponds to a time returned in the column vector `TOUT`. If you are solving a single differential equation, `TOUT` and `YOUT` will both be $(n \times 1)$ matrices such that `YOUT(j)` $= y($`TOUT(j)`$)$.

  If you are solving a system of m differential equations, then `TOUT` is still a $(n \times 1)$ matrix of times, but `YOUT` is a $(n \times m)$ matrix of computed y-values.

$$\texttt{TOUT} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix} \qquad \texttt{YOUT} = \begin{bmatrix} y_1(t_1) & y_2(t_1) & \dots & y_m(t_1) \\ y_1(t_2) & y_2(t_2) & \dots & y_m(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ y_1(t_n) & y_2(t_n) & \dots & y_m(t_n) \end{bmatrix} \tag{4}$$

  The number of rows in the `TOUT` and `YOUT` matrices, n, depends on how you defined `TSPAN`. If `TSPAN` is a (1×2) matrix, then `MATLAB` chooses n internally as it solves the problem. In this case the values in `TOUT` will NOT be evenly spaced in time. If `TSPAN` is a matrix `[T0 T1 ... TFINAL]`, then n is the number of columns in the `TSPAN` matrix.

Example 1

We want to solve

$$\frac{dy}{dt} = y \cos \sqrt{t} \tag{5}$$

with initial condition $y(0) = -1$. We are interested in the solution over the range $0 \leqslant t \leqslant 10$. We can solve this with one line of `MATLAB` code using the command line.

```
>> [T,Y] = ode45(@(t,y) y*cos(sqrt(t)), [0,10], -1);
```

After executing this command, `T` and `Y` are both (49×1) matrices. Since the `TSPAN` argument gives only beginning and an end, `[0,10]`, `MATLAB` automatically determines the number of steps internally. In this case 48 step were used; this yields 49 values including the initial condition. Note that the steps are not uniformly spaced: `MATLAB`'s `ode45` uses an adaptive step-size scheme. The solution is plotted in Figure 1.

Example 2

We want to solve

$$\frac{dy}{dt} = y \cos \sqrt{t} \tag{6}$$

with initial condition $y(3) = 2$. We are interested in the solution over the range $3 \leqslant t \leqslant 10$. Furthermore, we want numeric values for $y(t)$ where $t = 3.0, 3.1, 3.2, \ldots, 10.0$. We can solve this with two lines of `MATLAB` code using the command line.

```
>> tspan = linspace(3,10,71);
>> [T,Y] = ode45(@(t,y) y*cos(sqrt(t)), tspan, 2);
```

After executing this command, `T` and `Y` are both (71×1) matrices. In this case, the `TSPAN` argument contains more than two values, so `ode45` returns the solution at these points only. Internally, `MATLAB`'s `ode45` uses an adaptive step-size scheme and the computational step-size may be much smaller than the steps given in `TSPAN`. The solution is plotted in Figure 2.

Example 3

We want to solve the system of two first-order differential equations

$$\frac{dx}{dt} = y \cos \sqrt{t} \tag{7}$$

$$\frac{dy}{dt} = x \sin \sqrt{t} \tag{8}$$

with initial conditions $x(0) = 1$ and $y(0) = -1$. We are interested in the solution over the range $0 \leqslant t \leqslant 10$. We can solve this with one line of `MATLAB` code using the command line, but it is more informative to use a separate function.

```
function [ dYdt ] = Example3(t,Y)
    x = Y(1);
    y = Y(2);

    dxdt = y*cos(sqrt(t));
    dydt = x*sin(sqrt(t));

    dYdt = [dxdt; dydt];
end
```

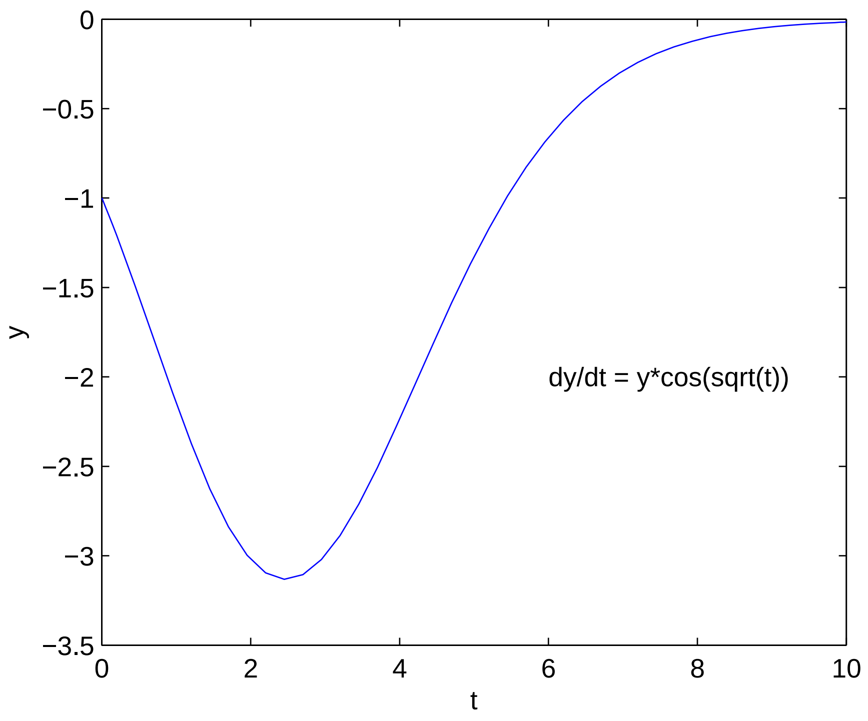

Figure 1: Solution to Example 1.

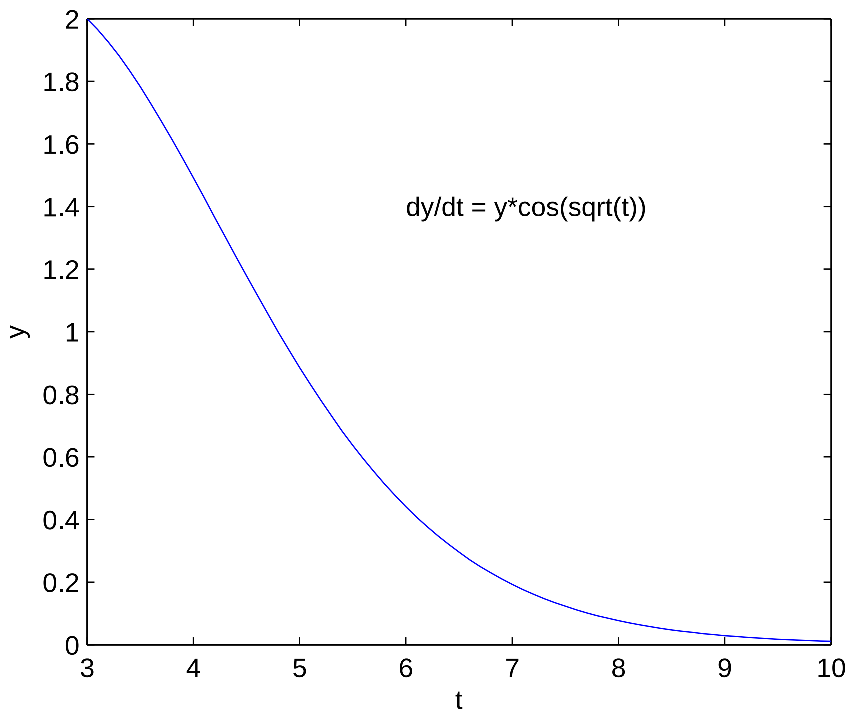

Figure 2: Solution to Example 2.

To actually solve the differential equation, we type the following on the command line.

```
>> [T,Y] = ode45(@Example3, [0,10], [1,-1]);
```

After executing this command, `T` is a (49×1) matrix, but `Y` is a (49×2). The first column of `Y` are the computed x values, the second column of `Y` are the computed y values. The solution is plotted in Figure 3.

It is also interesting to interpret the solution to (7) and (8) as a pair of parametric equations, and plot x versus y; see Figure 4.

Example 4

We want to solve a third order differential equation

$$\frac{d^3y}{dx^3} = \cos\left(x \cdot \sin(y)\right) \tag{9}$$

with initial conditions $y(0) = 1$, $dy/dx(0) = -1/2$, and $d^2y/dx^2(0) = -1$. We are interested in the solution over the range $0 \leqslant t \leqslant 5$. Our first necessary step is to convert the third-order differential equation into a system of three first-order differential equations.

We define two intermediate variables u and v as

$$u = \frac{dy}{dx} \tag{10}$$

$$v = \frac{d^2y}{dx^2} \tag{11}$$

and rewrite (9) as

$$\frac{dy}{dx} = u \tag{12}$$

$$\frac{du}{dx} = v \tag{13}$$

$$\frac{dv}{dx} = \cos\left(x \cdot \sin(y)\right) \tag{14}$$

$$\tag{15}$$

with associated boundary conditions $y(0) = 1$, $u(0) = -1/2$, and $v(0) = -1$

Our associated `ODEFUN` could look like

```
function [dYdx] = Example4(x,Y)
    y = Y(1);
    u = Y(2);
    v = Y(3);

    dvdx = cos(x*sin(y));
    dudx = v;
    dydx = u;

    dYdx = [dydx; dudx; dvdx];
end
```

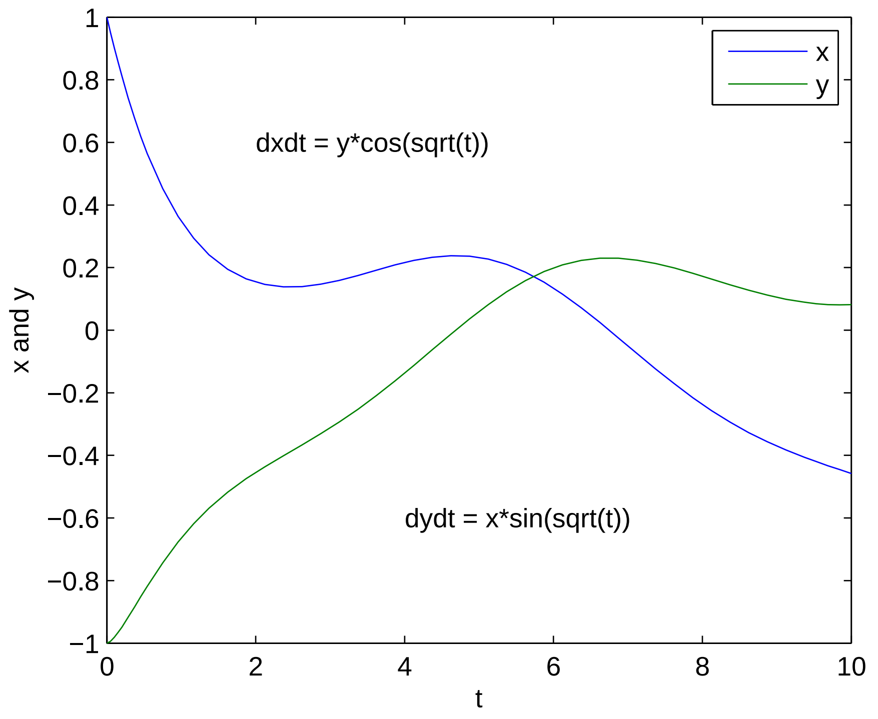

Figure 3: Solution to Example 3.

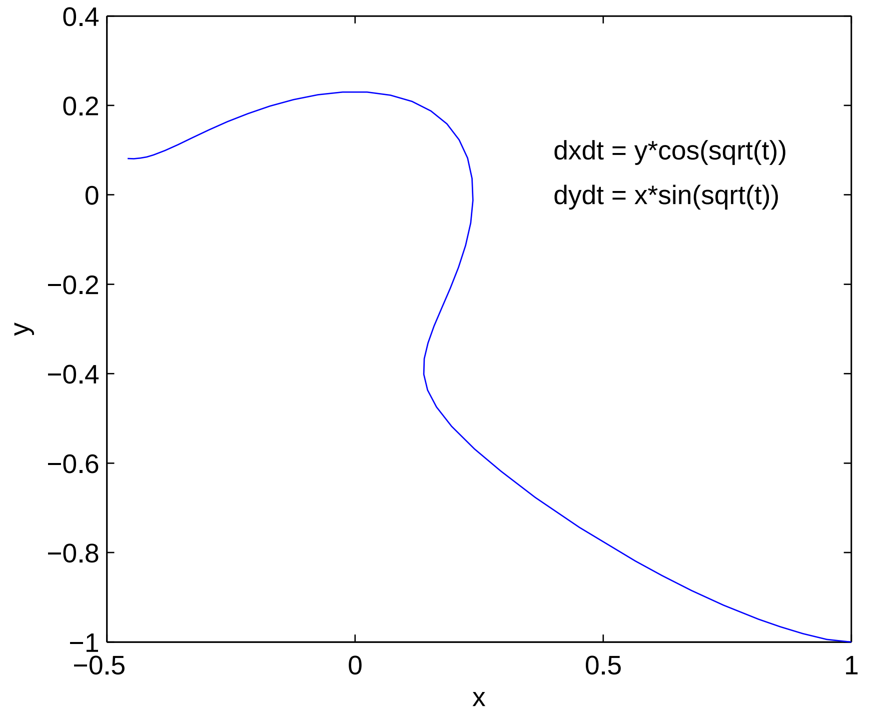

Figure 4: Solution to Example 3 represented as a parametric equation.

To actually solve the differential equation, we type the following on the command line.

```
>> [x,Y] = ode45(@Example4, [0,5], [1,-0.5,-1]);
```

After executing this command, `x` is a (45×1) matrix, but `Y` is a (45×3). The first column of `Y` are the computed y values, the second column of `Y` are the computed $dy/dx = u$ values, and the third column of `Y` are the computed $d^2y/dx^2 = v$ values. The solution is plotted in Figure 5.

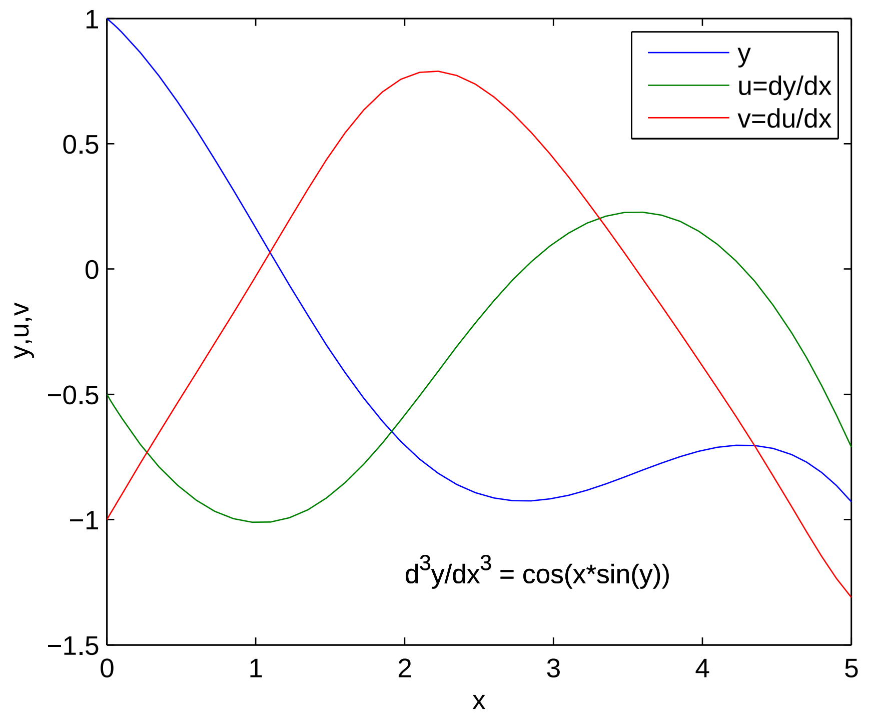

Figure 5: Solution to Example 4.