Modelling and Simulation in Matlab/Simulink

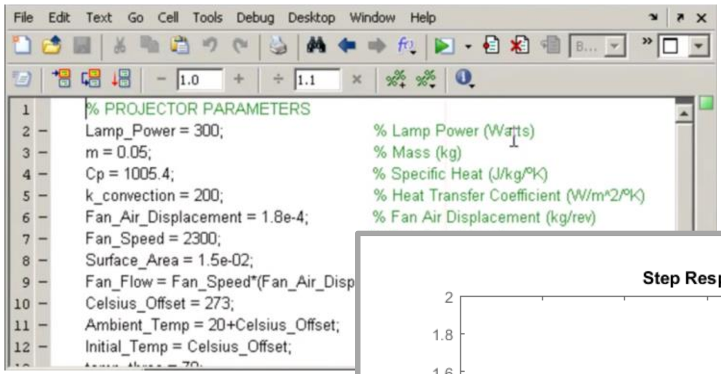
Mirco Rampazzo

mirco.rampazzo@dei.unipd.it

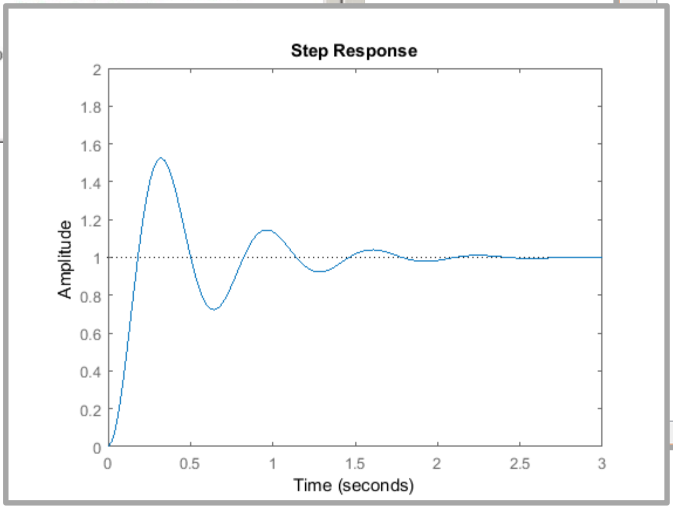
Lesson materials can be found at:

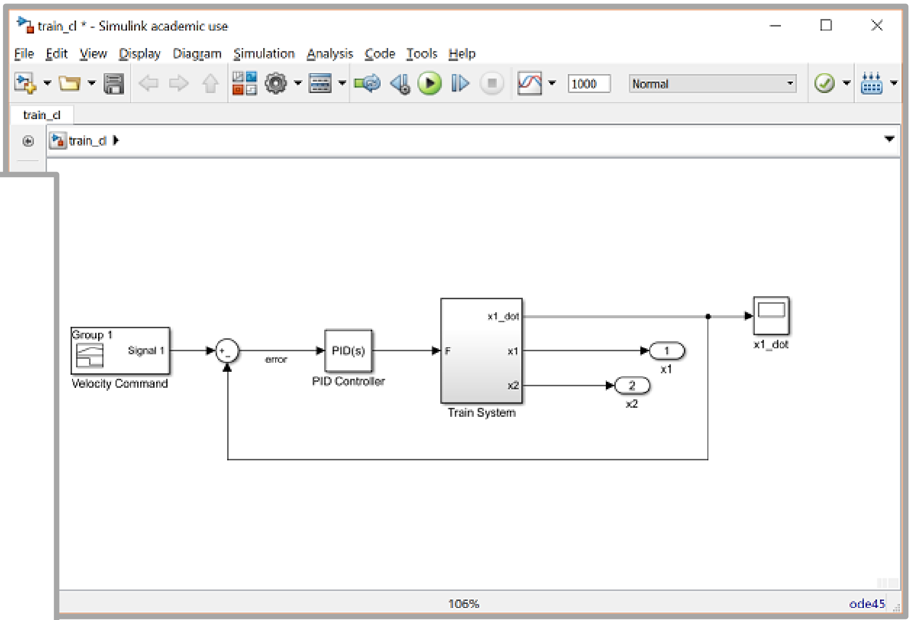
http://automatica.dei.unipd.it/people/rampazzo/teaching.html

Modelling and Simulation in Matlab/Simulink

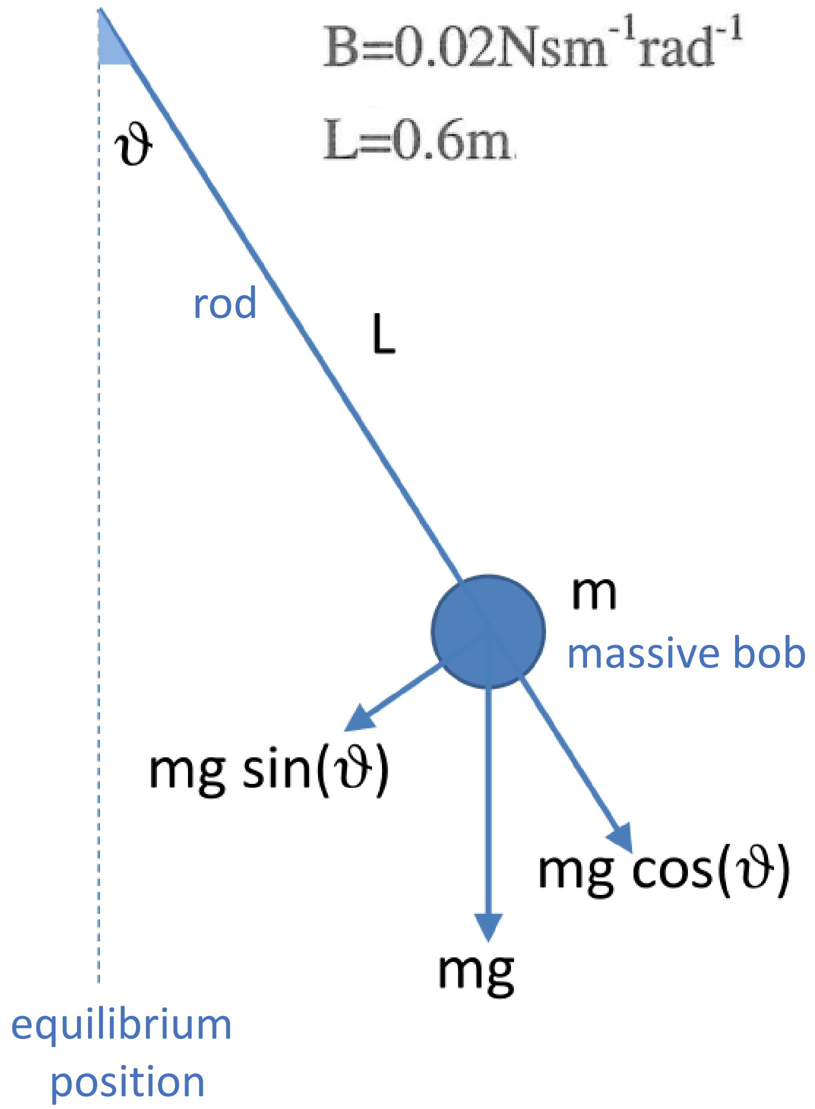


```
% PROJECTOR PARAMETERS
Lamp_Power = 300;                   % Lamp Power (Watts)
m = 0.05;                           % Mass (kg)
Cp = 1005.4;                        % Specific Heat (J/kg/°K)
k_convection = 200;                 % Heat Transfer Coefficient (W/m^2/°K)
Fan_Air_Displacement = 1.8e-4;      % Fan Air Displacement (kg/rev)
Fan_Speed = 2300;
Surface_Area = 1.5e-02;
Fan_Flow = Fan_Speed*(Fan_Air_Disp
Celsius_Offset = 273;
Ambient_Temp = 20+Celsius_Offset;
Initial_Temp = Celsius_Offset;
```





Simple Pendulum



W=mg=2N

B=0.02Nsm^{-1}rad^{-1}

L=0.6m

- Second law of motion:

$$mL^2\ddot{\theta}(t) = -mgL\sin\theta(t) - BL^2\dot{\theta}(t)$$

- State-space Model:

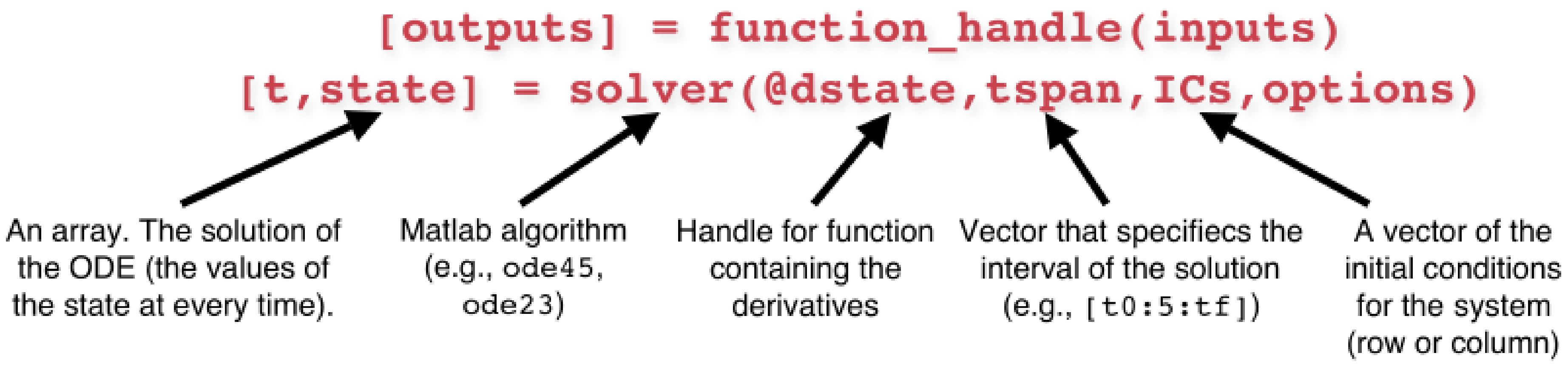
$$\begin{cases} x_1(t) = \theta(t) \\ x_2(t) = \dot{\theta}(t) \end{cases}$$

- ODEs Initial Value Problem (Cauchy Problem):

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = -\frac{g}{L}\sin x_1(t) - \frac{B}{m}x_2(t) \\ x_1(0)=1\text{rad} \\ x_2(0)=0\text{rads}^{-1} \end{cases} \qquad \mathbf{x}(t)=[x_1(t)\ x_2(t)]^T$$

Solving ODEs in Matlab

- ✓ An ODE is an equation that contains one independent variable (e.g. time) and one or more derivatives with respect to that independent variable
- ✓ In the time domain, ODEs are initial-value problems, so all the conditions are specified at the initial time t = 0
- ✓ Matlab has several different functions (built-ins) for the numerical solution of ODEs.
- ✓ These solvers can be used with the following syntax:



```
[outputs] = function_handle(inputs)
[t,state] = solver(@dstate,tspan,ICs,options)
```

An array. The solution of the ODE (the values of the state at every time).

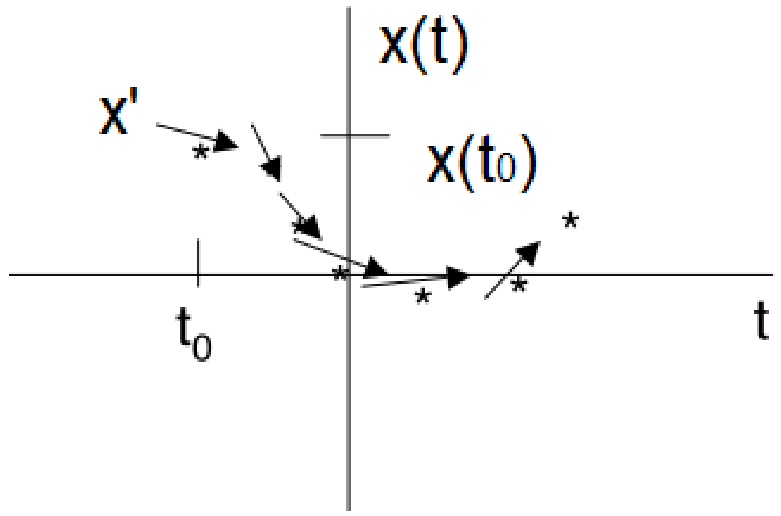
Matlab algorithm (e.g., `ode45`, `ode23`)

Handle for function containing the derivatives

Vector that specifiecs the interval of the solution (e.g., `[t0:5:tf]`)

A vector of the initial conditions for the system (row or column)

What are we doing when numerically solving ODE's?



- ✓ we know t_0 and $x(t_0)$
- ✓ we know the slope of x(t): dx/dt = f(t,x)
- ○ we don't know x(t) for any values of t other than t_0

❑ Integrators compute nearby value of x(t) using what we already know and repeat:

Euler's Method:

$$x(t_1) = x(t_0)+f(t_0,x(t_0))(t_1-t_0)$$
$$x(t_2) = x(t_1)+f(t_1,x(t_1))(t_2-t_1)$$
$$\ldots$$

❑ Higher order numerical methods reduce error at the cost of speed:

- Euler's Method - 1st order expansion
- Midpoint method - 2nd order expansion
- Runge-Kutta - 4th order expansion

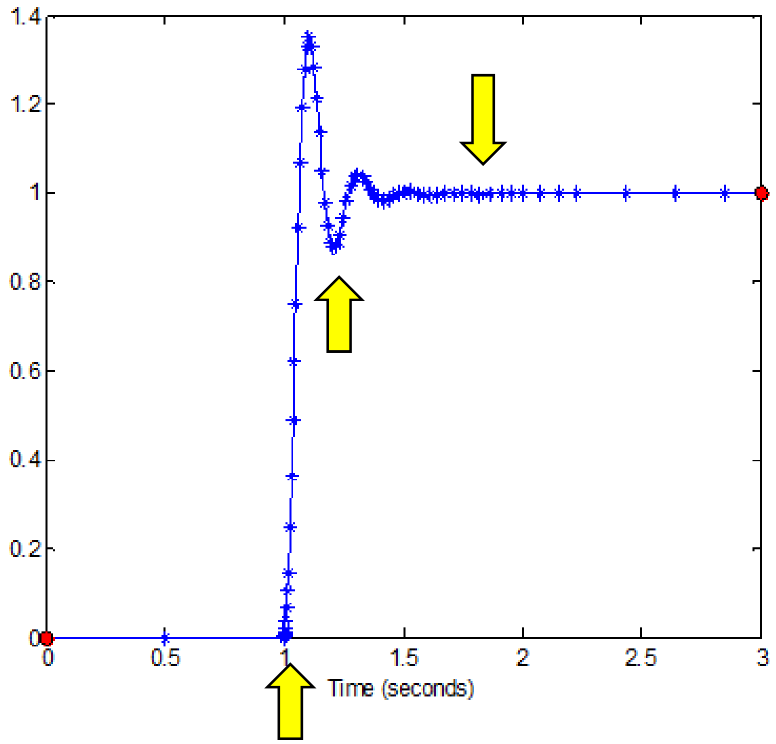
Choose a Solver

- There are solvers for continuous-time models and discrete-time models:
    - even for the discrete-time solvers it is possible to select between fixed step methods (the step will be equal to the sampling period or a fraction if it) or variable step methods (the step will be equal to one or more sampling periods)
    - it is possible to simulate a discrete-time model with a continuous-time solver, but the reverse is not allowed

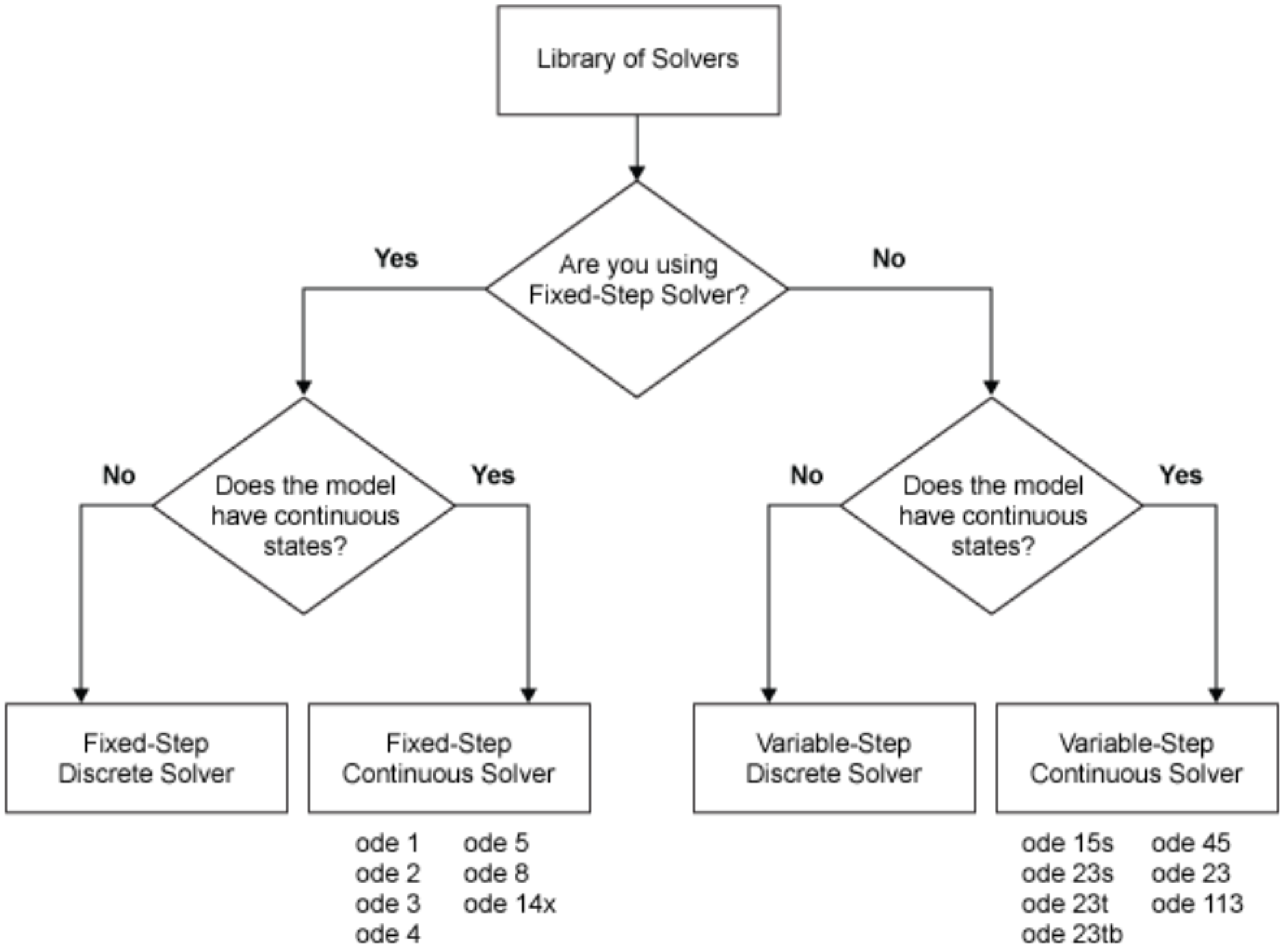
Choose a Solver

- Some available solvers:
  - ode45 with variable-step can be applied in many situations, and for this reason is the default solver in Simulink. It may not be so effective in simulating stiff models, i.e. models in which some components with very fast and very slow dynamics (i.e. component reacting promptly or very slowly to an excitation) coexist
  - ode1s with variable-step is the first method to try when it is believed that the model is stiff
  - discrete is the solver used for discrete-time models

- Example: variable-step



Choose a Solver

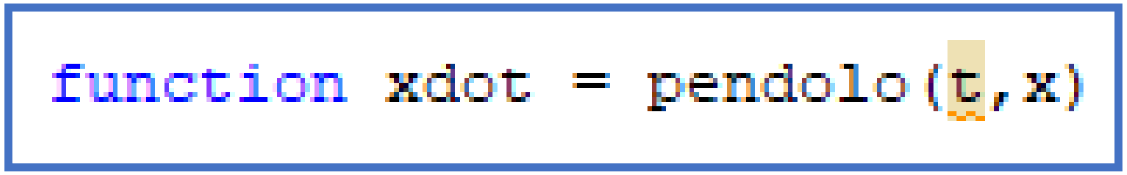

Library of Solvers
Are you using Fixed-Step Solver?
Yes
No
Does the model have continuous states?
No
Yes
Does the model have continuous states?
No
Yes
Fixed-Step Discrete Solver
Fixed-Step Continuous Solver
Variable-Step Discrete Solver
Variable-Step Continuous Solver
ode 1
ode 2
ode 3
ode 4
ode 5
ode 8
ode 14x
ode 15s
ode 23s
ode 23t
ode 23tb
ode 45
ode 23
ode 113

Choose a Solver

Solver	Accuracy	Description
`ode45`	Medium	This should be the first solver you try
`ode23`	Low	Less accurate than `ode45`
`ode113`	Low to high	For computationally intensive problems
`ode15s`	Low to medium	Use if `ode45` fails because the problem is stiff*

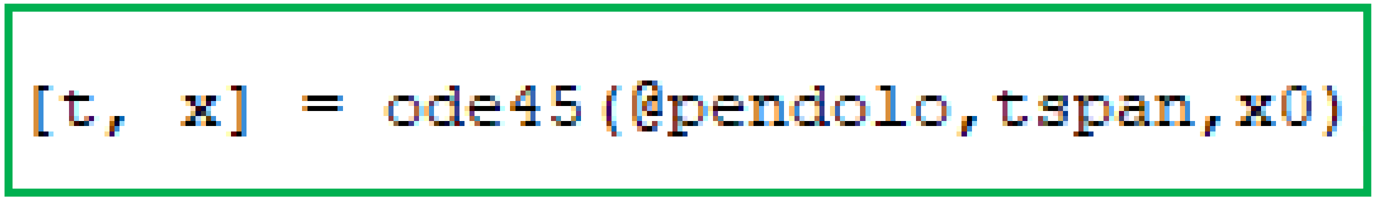
*An ordinary differential equation problem is stiff if the solution being sought is varying slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results.

Defining an ODE function in an M-file



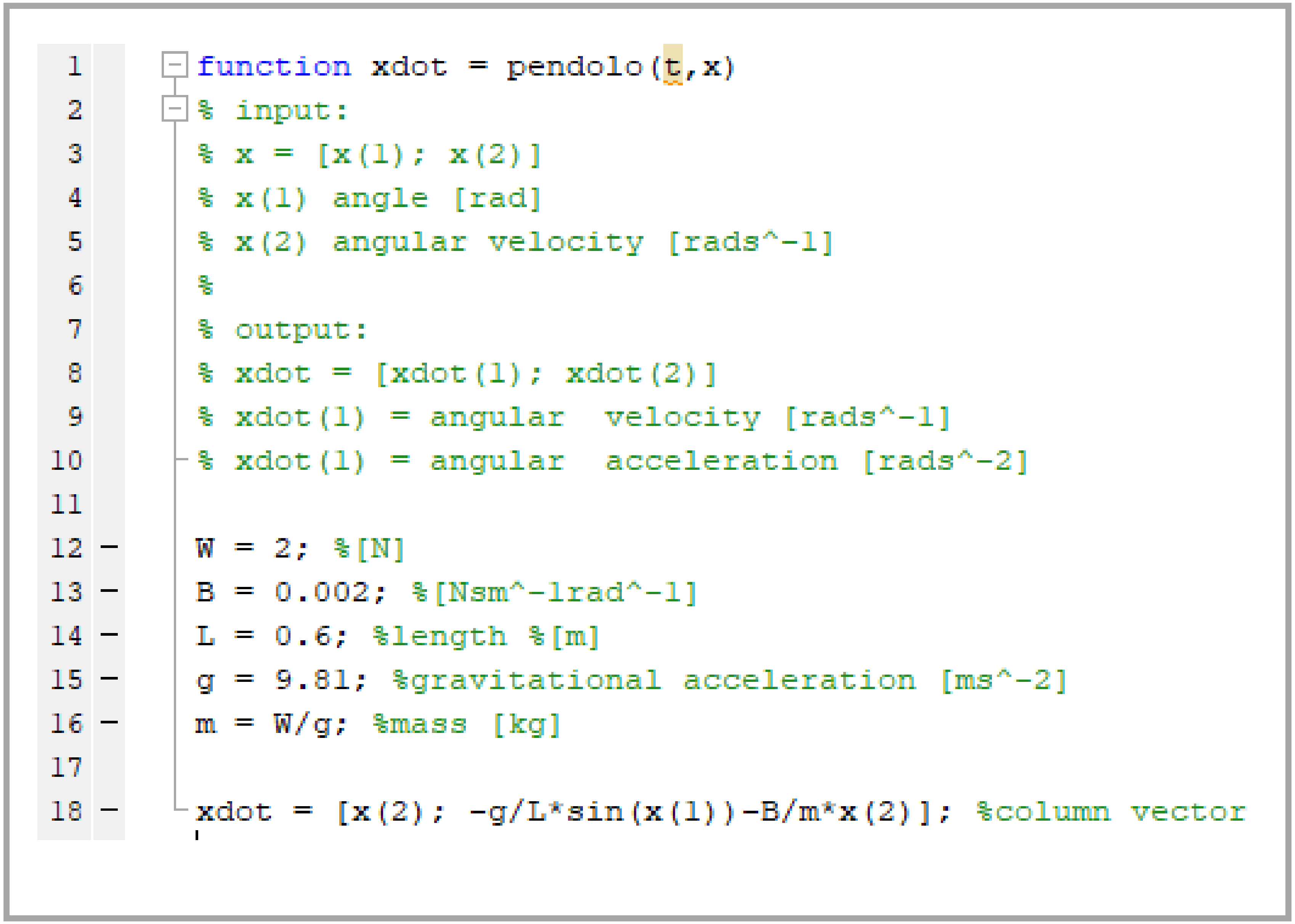
```
function xdot = pendolo(t,x)
```

integrates the system of differential equation



```
[t, x] = ode45(@pendolo,tspan,x0)
```

Defining an ODE function in an M-file

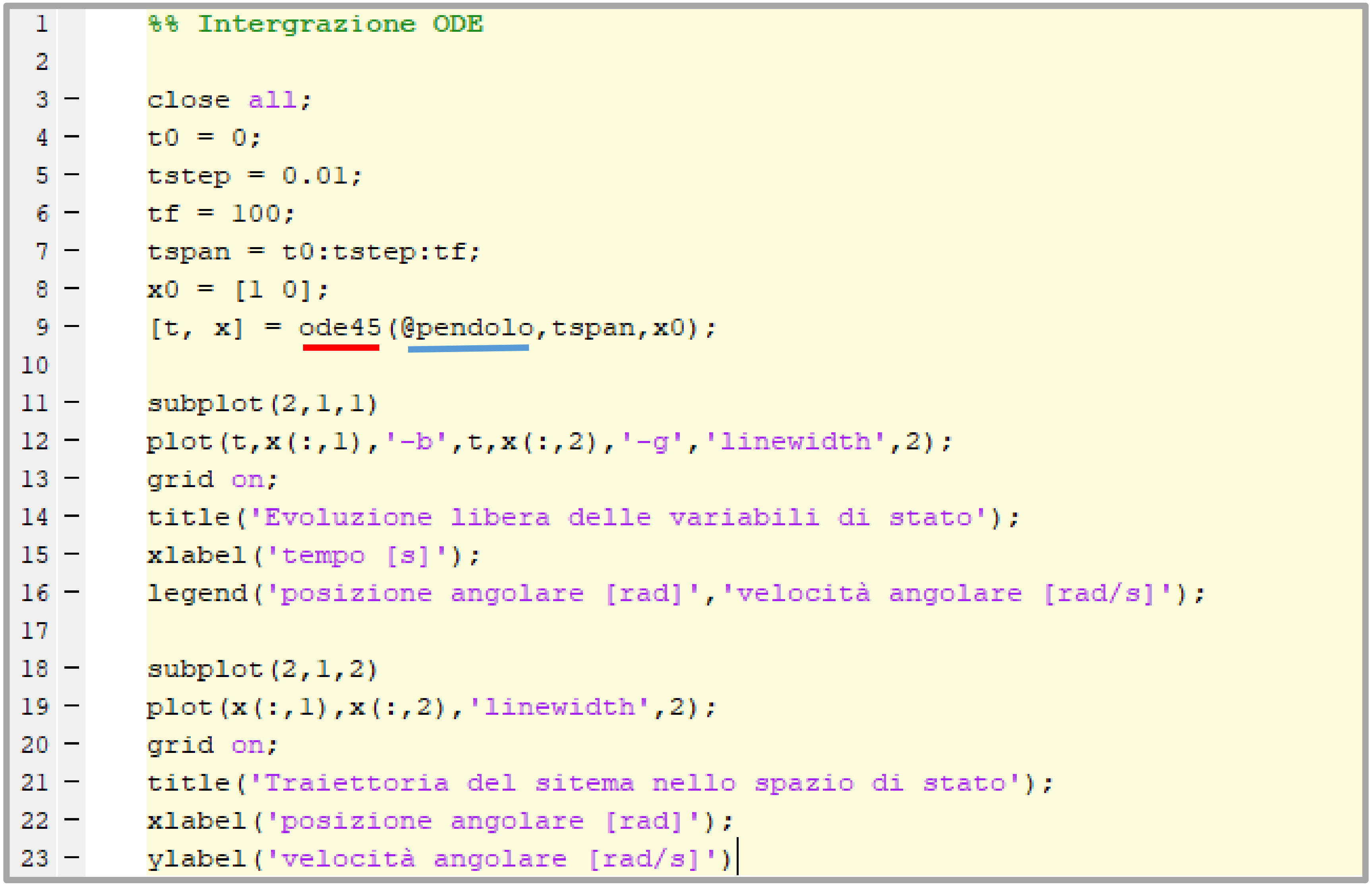


```
function xdot = pendolo(t,x)
% input:
% x = [x(1); x(2)]
% x(1) angle [rad]
% x(2) angular velocity [rads^-1]
%
% output:
% xdot = [xdot(1); xdot(2)]
% xdot(1) = angular  velocity [rads^-1]
% xdot(1) = angular  acceleration [rads^-2]

W = 2; %[N]
B = 0.002; %[Nsm^-1rad^-1]
L = 0.6; %length %[m]
g = 9.81; %gravitational acceleration [ms^-2]
m = W/g; %mass [kg]

xdot = [x(2); -g/L*sin(x(1))-B/m*x(2)]; %column vector
```

Integrate the System of ODE



```
%% Intergrazione ODE

close all;
t0 = 0;
tstep = 0.01;
tf = 100;
tspan = t0:tstep:tf;
x0 = [1 0];
[t, x] = ode45(@pendolo,tspan,x0);

subplot(2,1,1)
plot(t,x(:,1),'-b',t,x(:,2),'-g','linewidth',2);
grid on;
title('Evoluzione libera delle variabili di stato');
xlabel('tempo [s]');
legend('posizione angolare [rad]','velocità angolare [rad/s]');

subplot(2,1,2)
plot(x(:,1),x(:,2),'linewidth',2);
grid on;
title('Traiettoria del sitema nello spazio di stato');
xlabel('posizione angolare [rad]');
ylabel('velocità angolare [rad/s]')
```