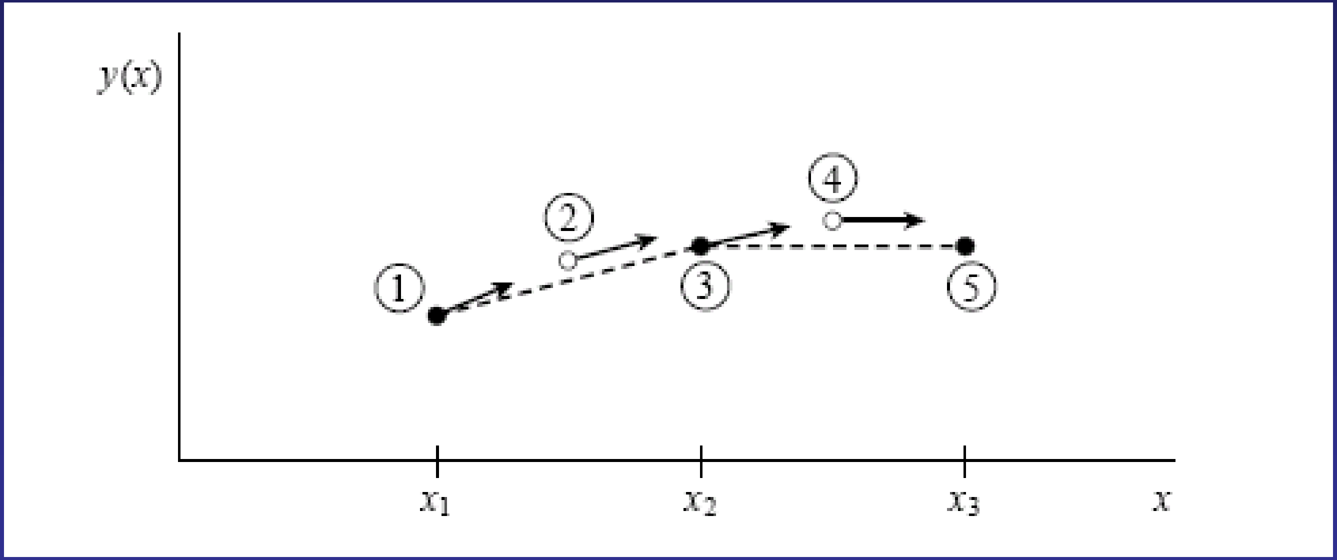
Algorithms and ODE



How to we solve problems with a computer?

We need an algorithm!

Algorithm language – “An arithmetic language presenting numerical procedures to a computer…” American Heritage Dictionary

Revisit the task of recovering the motion of a dynamical system from its equation of motion

Consider the simplest 1st order system:

$$\boxed{b\dot{x} + kx = 0}$$

What does this system corresponds to?

The solution of this system can of course be obtained analytically but also simply numerically by a single integration

Simple integration -- Quad

quad(func, a, b)

$$y = \int_a^b f(x)dx$$

As an example:

Define m-file:

```
function y=fun1(x)
y=x.*x;
```

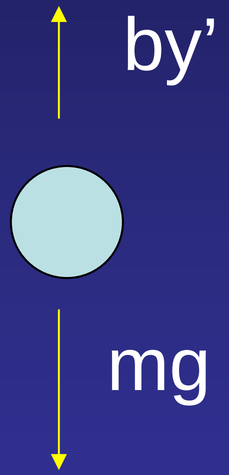
Now we are ready to use quad:

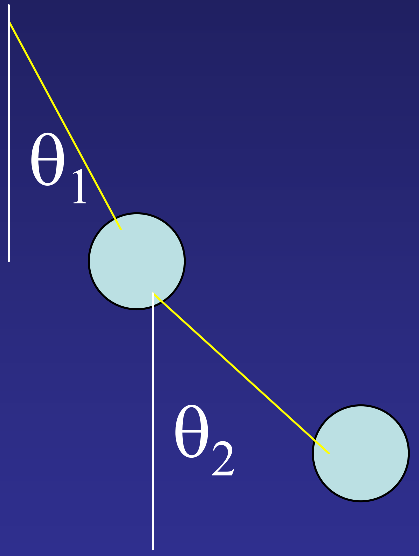
```
>> quad(@fun1, 0, 1)

ans =
    0.3333
```

Limitation of Simple Integration: Quad

Simple integration is very limited and does not solve a large class of dynamic problems. As examples:

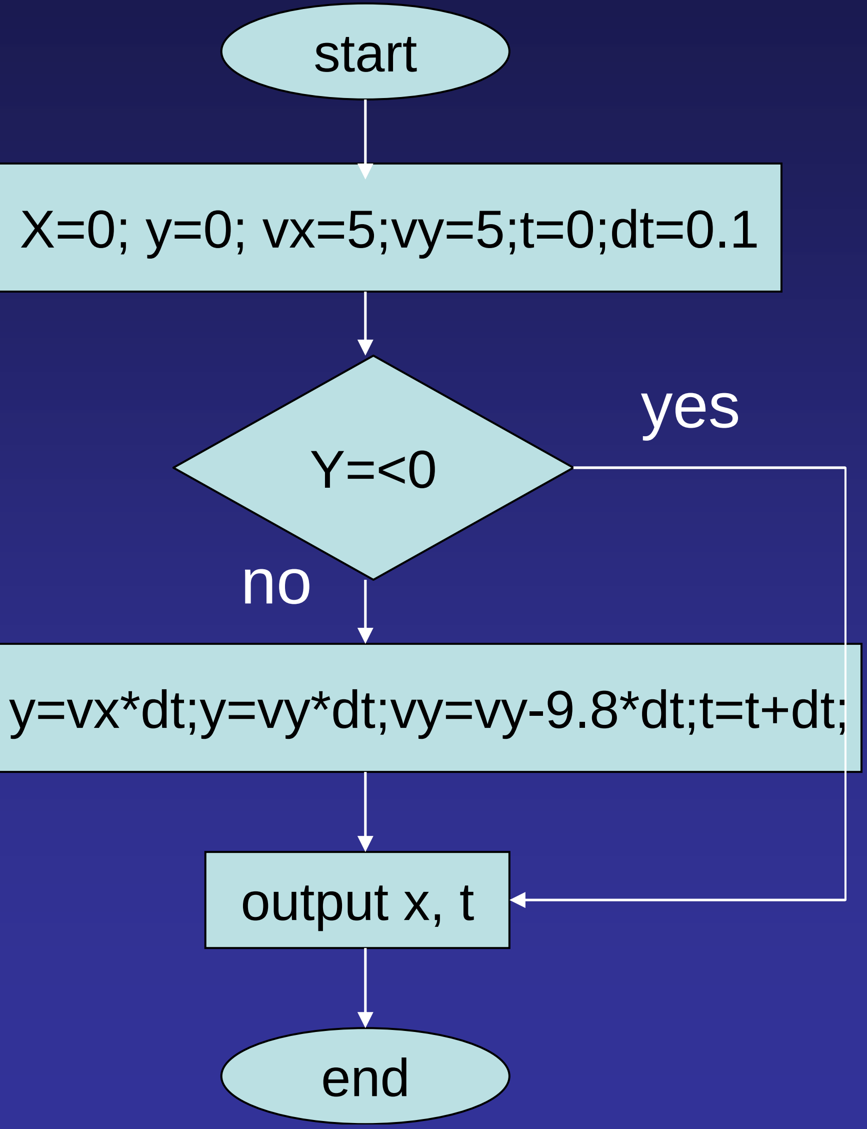


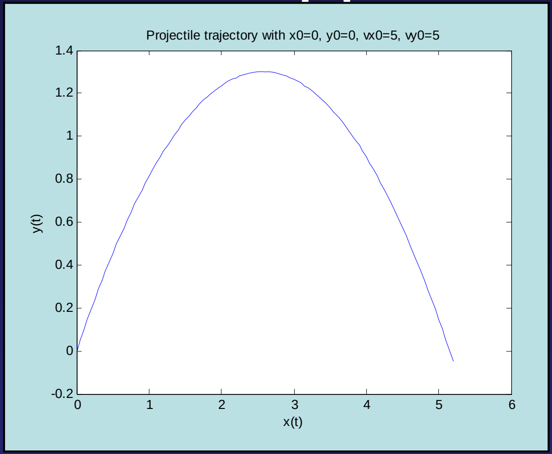


Falling ball – 2nd order

Coupled multiple degree of freedom system

How did we solve this class of problems? We use a very simple straight forward approach of doing numerical integration:


start
X=0; y=0; vx=5;vy=5;t=0;dt=0.1
Y=<0
yes
no
y=vx*dt;y=vy*dt;vy=vy-9.8*dt;t=t+dt;
output x, t
end



Actually, this simple approach Has a name – it is called Euler Method

In general, you should NEVER ever use Euler Method. People uses it only if they don't know any better.

The General Numerical Problem of Solving Ordinary Differential Equations (ODEs)

$$y^{(n)} = f(y^{(n-1)}, \cdots, y', y, t)$$

Note that y does not have to be a scaler but can be a vector as in the case for multiple degrees of freedom systems

$$y = (y_1, y_2, \cdots y_m)$$

Converting higher order differential equation to a system of first order differential equation

Consider probably the most important case:

$$y'' = f(t)y' + g(t)y + h(t)$$

This can be readily converted to a system of first order differential equations

$$y_2 = y' \quad y_1 = y$$

$$y_1' = y_2$$
$$y_2' = f(t)y_2 + g(t)y_1 + h(t)$$

General equivalence between higher order differential equation and a system of first order equations

$$y^{(n)} = f(y^{(n-1)},\cdots,y',y,t)$$

$$y_1 = y;\ y_2 = y';\cdots,\ y_{n-1} = y^{(n-2)};\ y_n = y^{(n-1)}$$

$$y_n' = f(y_n,\cdots,y_2,y_1,t)$$

The problem of solving all higher order ordinary differential equation is thus reduced to solving a system of linear differential equations

Solving linear first order differential equation by Euler Method

In general, the system of equations look like:

$$y_i'(t) = f_i(y_1, y_2, \cdots, y_n, t) \quad i = 1 \cdots n$$

Euler Method says:

$$y_i(j\Delta t) = y_i((j-1)\Delta t) + f_i(y_1((j-1)\Delta t), \cdots, y_n((j-1)\Delta t), (j-1)\Delta t)\Delta t$$
$$i = 1 \cdots n$$

This equation can be solved if we have the initial conditions:

$$y_1(0) = y_{10},\ y_2(0) = y_{20}, \cdots, y_n(0) = y_{n0}$$

What is the accuracy of the Euler Method?

Euler method is equivalent to taking the 1st order Taylor series expansion for $y_i(t)$; it is unsymmetric and uses only the derivative information at the start of the time step

$$y_i(j\Delta t) = y_i((j-1)\Delta t) + f_i(y_1((j-1)\Delta t), \cdots, y_n((j-1)\Delta t), (j-1)\Delta t)\Delta t + O(\Delta t^2)$$
$$i = 1 \cdots n$$

Correction is only one less order then the correction term. How do we get better accuracy? We call the differences between Reimann sum and Simpson Rule ….

A working ODE solver – Runge-Kutta Method

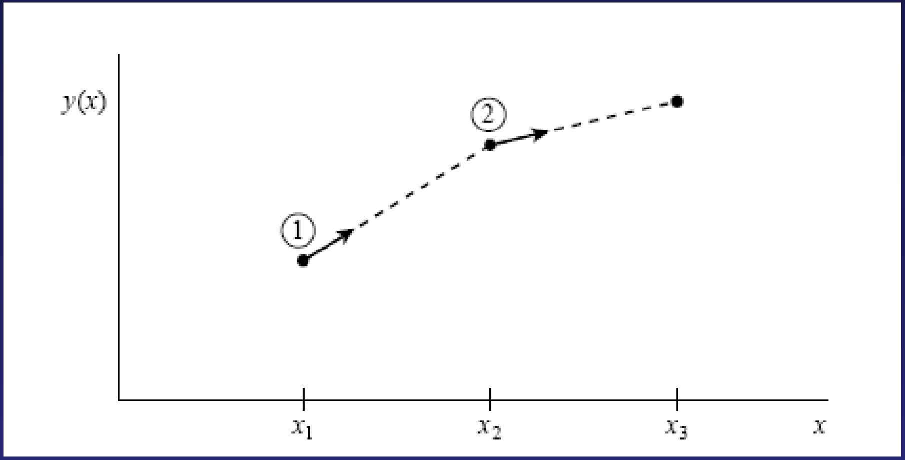
Estimate where the mid-point for y_i is first. Then, Evaluate the slope at the mid-point to estimate the next value of $y_{i.}$

$$
\begin{aligned}
&k1_i = f_i(y_1((j-1)\Delta t), \cdots, y_n((j-1)\Delta t), (j-1)\Delta t)\Delta t \\
&k2_i = f_i(y_1((j-1)\Delta t) + k1_1/2, \cdots, y_n((j-1)\Delta t + k1_n/2), (j-1/2)\Delta t)\Delta t \\
&y_i(j\Delta t) = y_i((j-1)\Delta t) + k2_i \\
&\quad i = 1 \cdots n
\end{aligned}
$$

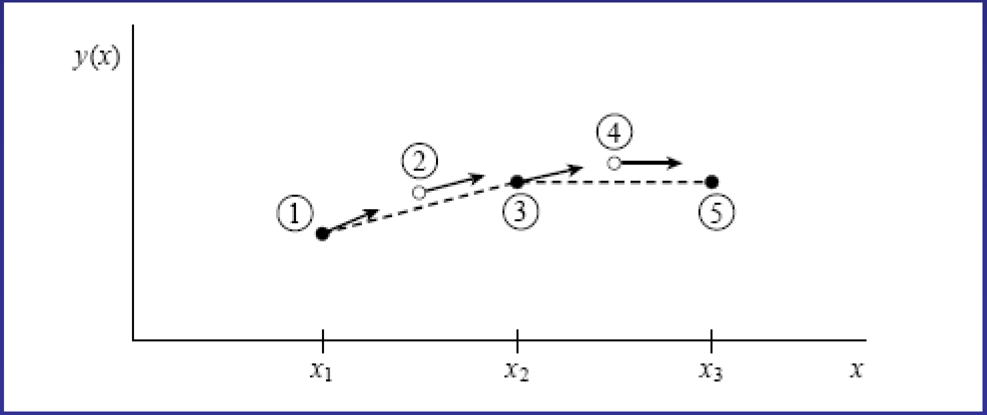
Because of symmetry, this method is good to $O(\Delta t^3)$

This is called the 2nd order Runge-Kutta method.

Schematically, the differences between Euler and 2nd order Runge-Kutta are fairly clear



Euler Method



2nd order Runge-Kutta

From Numerical Recipe in C

Higher Order Runge-Kutta Method

Just like Simpson method can be extended to higher order estimate, Runge-Kutta also has straightforward Higher order analog. The most commonly used one is the 4th order Runge-Kutta method

$$k1_i = f_i(y_1((j-1)\Delta t), \cdots, y_n((j-1)\Delta t), (j-1)\Delta t)\Delta t$$

$$k2_i = f_i(y_1((j-1)\Delta t) + k1_1/2, \cdots, y_n((j-1)\Delta t + k1_n/2), (j-1/2)\Delta t)\Delta t$$

$$k3_i = f_i(y_1((j-1)\Delta t) + k2_1/2, \cdots, y_n((j-1)\Delta t + k2_n/2), (j-1/2)\Delta t)\Delta t$$

$$k4_i = f_i(y_1((j-1)\Delta t) + k3_1 \cdots, y_n((j-1)\Delta t + k3_n), j\Delta t)\Delta t$$

$$y_i(j\Delta t) = y_i((j-1)\Delta t) + k1_i/6 + k2_i/3 + k3_i/3 + k4_i/6 + O(\Delta t^5)$$

$$i = 1 \cdots n$$

Runge-Kutta methods are implemented in MATLAB as ODE23 and ODE45 functions

Using MATLAB to solve a system of differential equation

Consider solving the following system of ODE:

$$\begin{aligned} y'_1 &= y_2\, y_3 & y_1(0) &= 0 \\ y'_2 &= -y_1\, y_3 & y_2(0) &= 1 \\ y'_3 &= -0.51\, y_1\, y_2 & y_3(0) &= 1 \end{aligned}$$

Using MATLAB to solve a system of differential equations

(1) First define the system of ODEs as a function:

```
function dy = system(t,y)
dy = zeros(3,1);    % a column vector
dy(1) = y(2) * y(3);
dy(2) = -y(1) * y(3);
dy(3) = -0.51 * y(1) * y(2);
```

(2) Call ODE45 or ODE23 using the function handle

```
[T,Y] = ode45(@system,[0 12],[0 1 1]);
```

(3) Plot result

```
plot(T,Y(:,1),'-',T,Y(:,2),'-.',T,Y(:,3),'.')
```

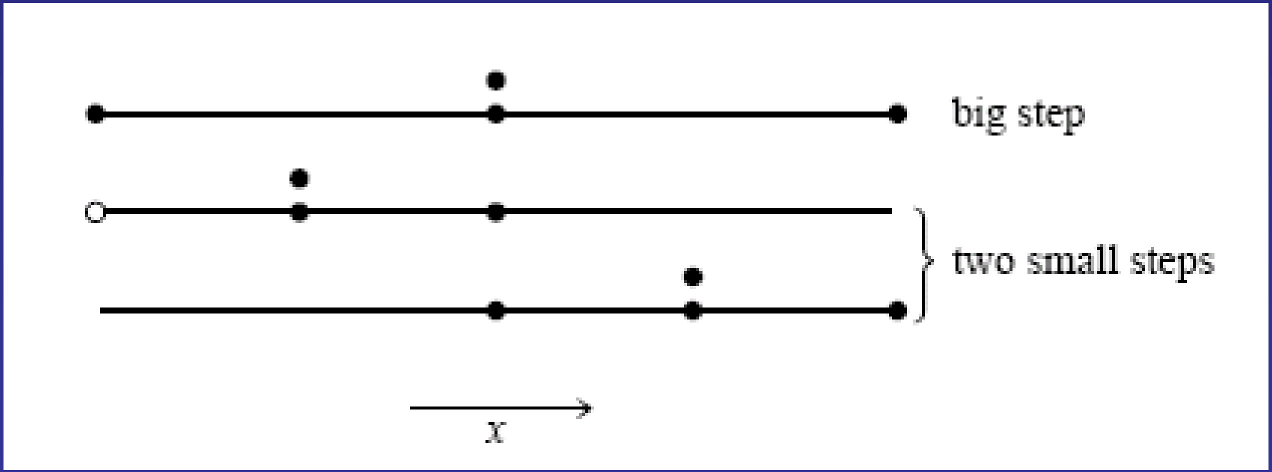
How should we choose the time step for Rugge-Kutta?

Consider implementing 4th order RK with taking a single step of 2 Δt (solution y_1) or two steps of Δt (solution y_2)

$$y(x+2\Delta t) = y_1 + (2\Delta t)^5 \phi + O(\Delta t^6)$$

$$y(x+2\Delta t) = y_2 + 2(\Delta t)^5 \phi + O(\Delta t^6)$$

Where $\phi \propto \frac{y^{(5)}(x)}{5!}$ from Taylor expansion



Numerical error and step size choice

Numerical error associated with step doubling is:

$$\boxed{\delta = y_2 - y_1}$$

So if we evaluate 4th order RK at each point twice, once by taking a full step and once by taking a half step, we have an estimate of the error associated with step size choice.

If we know δ can we choose step size? Yes.

If we measure error δ_0 associated with step size Δt_0, we can estimate Δt if the target error is δ:

$$\boxed{\Delta t = \Delta t_0 \left(\frac{\delta}{\delta_0} \right)^{0.2}}$$

Accuracy control in MATLAB

How to control step size and accuracy of Runge-Kutta. MATLAB handles step size control for you mostly but sometimes you do need to set the desired accuracy (to make sure that step is small enough) so that the ODE45 or ODE23 outputs are stable and correct.

Use odeset:

```
>> options = odeset('RelTol',1e-4,'AbsTol',[1e-7 1e-7]);
```

RelTol is fractional tolerance (default: 1e-3)
AbsTol is absolute tolerance (default: 1e-6)

Both criteria must be met.

Consider a simple case of a driven pendulum

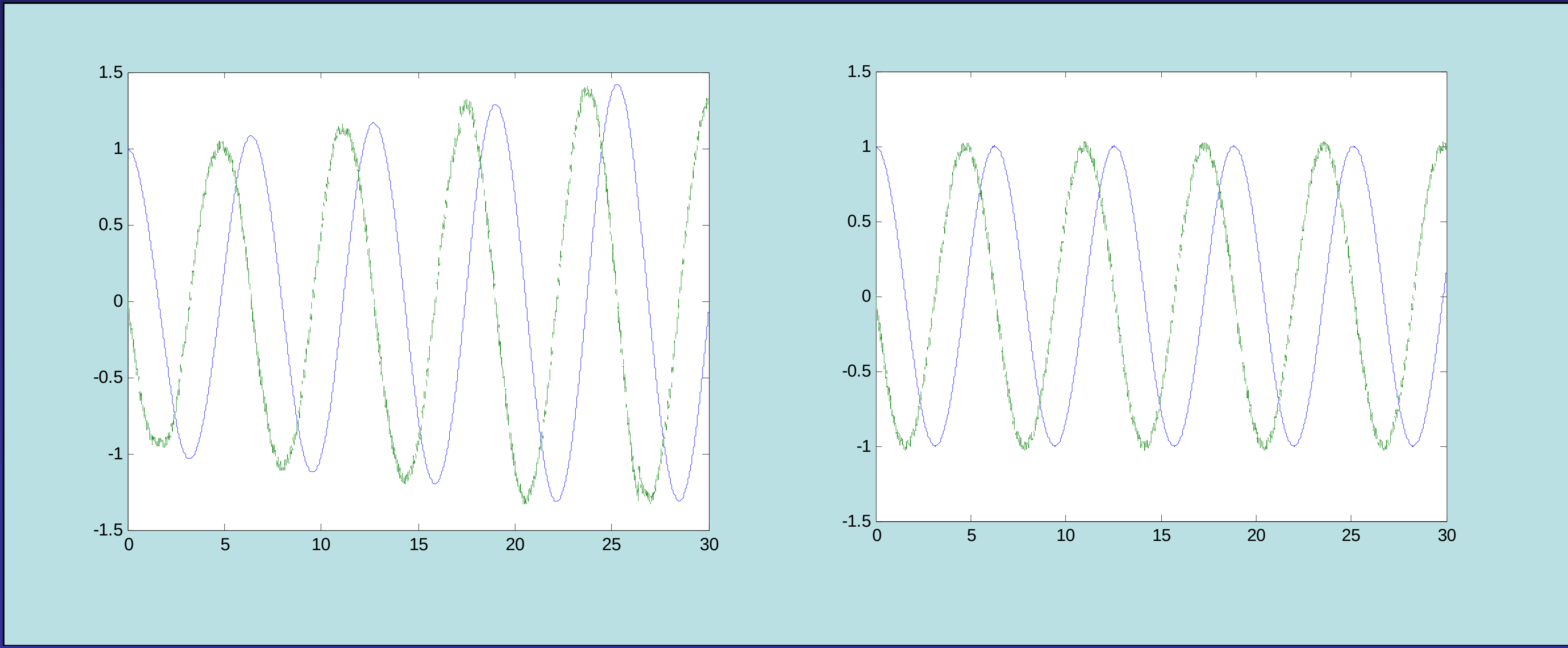
$$\ddot{\theta} + \frac{g}{l}\theta = F\cos(\omega t)$$

The corresponding m-file:

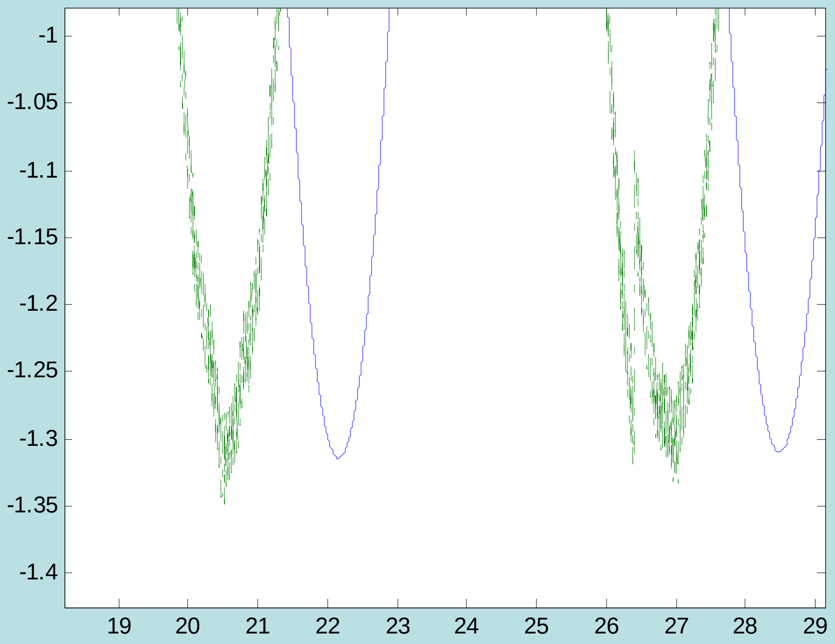
```
function dy = driven_pend(t,y)
dy = zeros(2,1);    % a column vector
g=10;
l=10;
f=1.5;
w=2*pi*100;
dy(1) = y(2);
dy(2) = -g/l*y(1) + f*cos(w*t);
```

Default Output of ODE23 for f=500

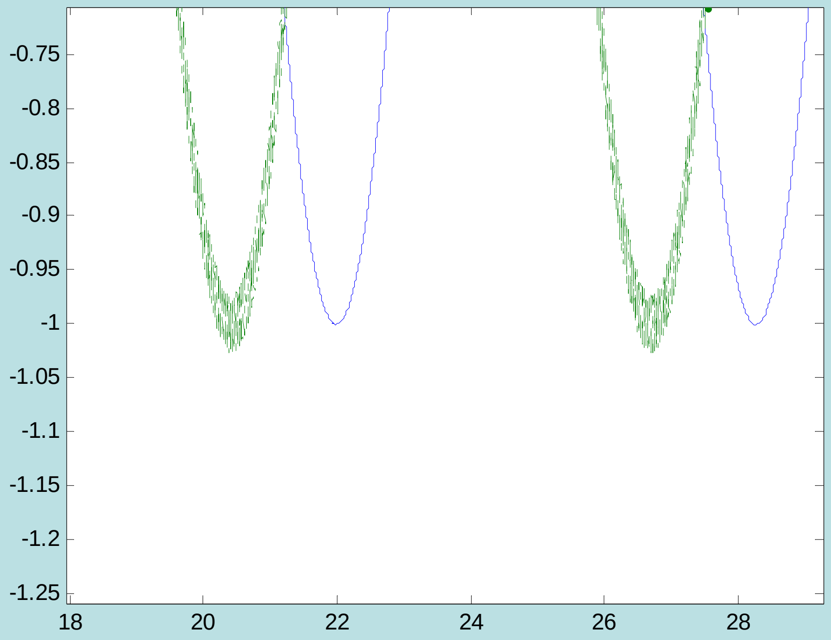
```
>>[T,Y] = ode45(@driven_pend,[0 30],[1 0]);
>> plot(T,Y(:,1),'-',T,Y(:,2),'-.')
>>options = odeset('RelTol',1e-5,'AbsTol',[1e-7 1e-7]);
>>[T,Y] = ode45(@driven_pend,[0 30],[1 0], options);
>> plot(T,Y(:,1),'-',T,Y(:,2),'-.')
```



Artifacts in velocity profile without precision control



-1
-1.05
-1.1
-1.15
-1.2
-1.25
-1.3
-1.35
-1.4
19
20
21
22
23
24
25
26
27
28
29



-0.75
-0.8
-0.85
-0.9
-0.95
-1
-1.05
-1.1
-1.15
-1.2
-1.25
18
20
22
24
26
28