Unit 5:

Numerical ODEs

Most Ordinary Differential Equations (ODEs) are not solvable on paper. Here's how we deal with them using computers.

Let's start with an example.

Example: Population Modeling

Let $p(t)$ bet the population at time t.

$$p(2000) = 30,700,000$$

We will refer to $p(t)$ as a "state variable". What affects population?

Change in $p(t)$ in 1 year $=$

$=$

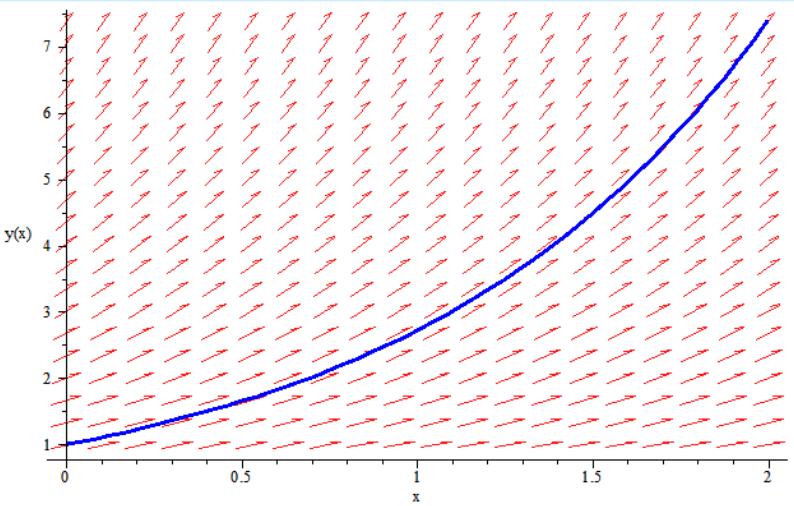
$=$

This is a differential equation (DE) because it involves a derivative. If we could find a function $p(t)$ that satisfies (1), then $p(t)$ would be called the "solution" of the DE.

eg.

Another example:

$$y'(x) = y$$
$$y(0) = 1$$
$$x \in [0, 2]$$



Initial Value Problems (IVP)

An Initial Value Problem consists of two parts:

Standard For for First-Order IVPs

Data for dyanmics

$m =$

$M =$

$f(t,z) =$

Data for initial state

$t_0 =$

$z^{(i)} =$

The IVP is

Example: Population Model

$m =$

$M =$

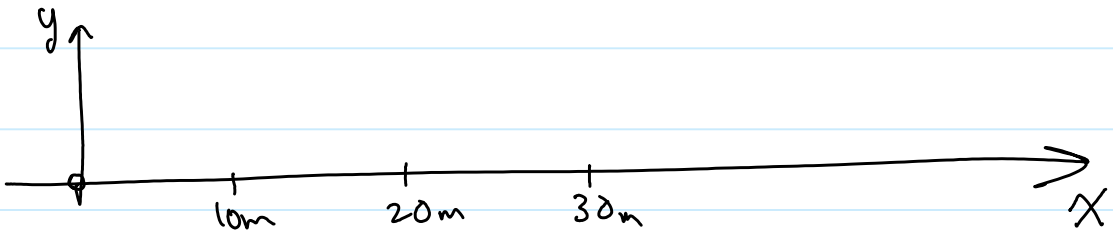
(1) $f(t,z) =$

(2) $z(2000) =$

L28: IVP part 2

Goal: Expand our formulation of IVPs to include "events".

Example: Novelty Golf Driving Range



Ball starts at $(x(0), y(0)) = (0, 0)$

Golfer hits the ball with initial velocity vector (v_x, v_y).

Dynamics Model

This is a 2nd-order differential equation (DE). Standard 1st-order form involves only 1st-order DEs.

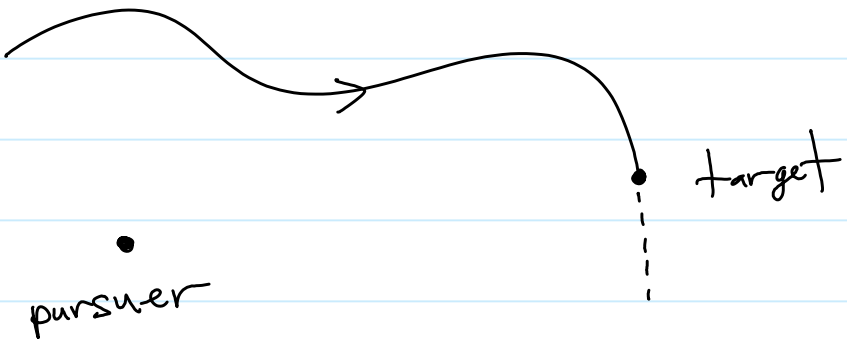
Let

Another example of a higher-order ODE converted into a system of first-order ODEs.

eg. $y'' + 3y' + 4y = \cos x$

Example: Pursuit

Think of a fox chasing a rabbit, or a heat-seaking missile fired at a jet.



The pursuer moves with speed S_p directly at the target at all times. In what direction does the pursuer move?

Pursuer

$(x_p(t), y_p(t), z_p(t))$

Target

$(x_T(t), y_T(t), z_T(t))$

As a unit vector…

We multiply it by S_p to get the velocity vector for the pursuer.

$$\therefore \quad \frac{dx_p(t)}{dt} =$$

$$\frac{dy_p(t)}{dt} =$$

$$\frac{dz_p(t)}{dt} =$$

In standard form, $m = 3$ $\quad M = I$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \qquad f(t, z) = \qquad z(0) =$$

Terminal Events

A terminal event occurs when something happens in the simulation that is not incorporated in the dynamics model. For example, a falling object hits the ground, or a prey species goes extinct.

Eg. **Golf Driving Range Model**

We want to terminate the computation when

a. the ball hits the ground

b. the ball hits the barrier

Eg. **Pursuit**

Terminate when

END.

Goal: To see how solutions to ODEs can be approximated numerically.

Numerical Solution of ODEs

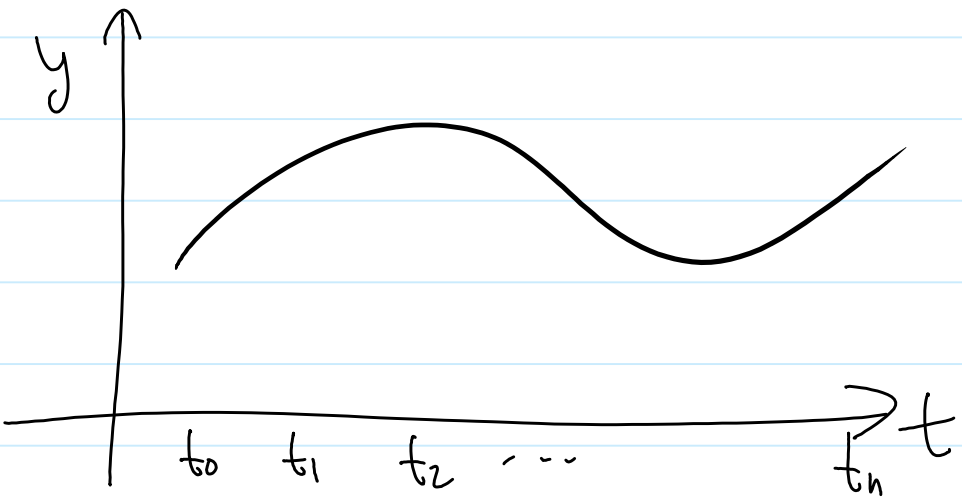
Almost all DEs are not solvable analytically. That is, rarely do we have a mathematical formula for the solution of a DE. In practice, solutions to DEs are usually approximated numerically.

Consider the IVP

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

2) $y(t_0) = y^{(i)}$

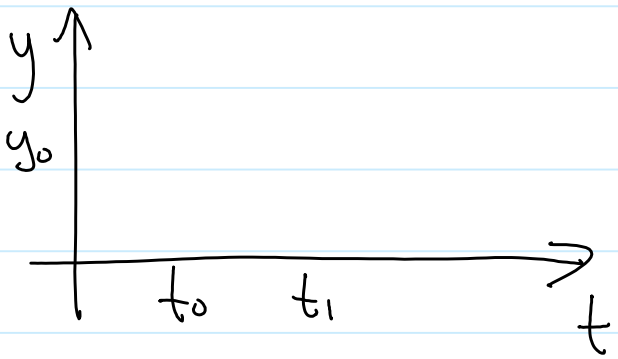
If $y(t)$ is the (hypothetical) true solution, we want to find a set of points such that



Euler's Method

Starting with the initial state, we can approximate the solution by

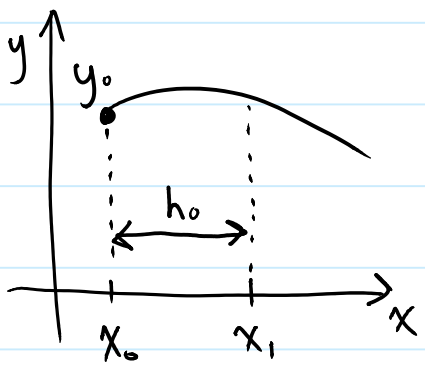
Let h_0 be the first time step, so that $t_1 - t_0 = h_0$.



The slope at (t_0, y_0) can be approximated by

And in general:

This is called **Euler's Method**



<u>Local and Global Error</u>

Let us denote the true solution curve through (x_n, y_n) as $\hat{y}_n(x)$.

Local Error: $l_{n+1} = \left| \hat{y}_n(x_{n+1}) - y_{n+1} \right|$

Global Error: $\varepsilon_{n+1} = \left| \hat{y}_0(x_{n+1}) - y_{n+1} \right|$

The local error of Euler's Method is

To integrate a solution through a fixed domain of length c (eg. tspan=[0 2], so $c=2$), the number of steps of length h would be

$N =$

For this reason, the global error for Euler's Method is

Example: Golf Driving Range

$$f(t,z) = \begin{bmatrix} V_x \\ z_3 \\ -g \end{bmatrix} \qquad z(0) = \begin{bmatrix} 0 \\ 0 \\ V_y \end{bmatrix}$$

Let $V_x = 30$, $V_y = 12$, $g = 9.81$

Euler step of size $h_0 = 0.1$ seconds

$z^{(1)} =$

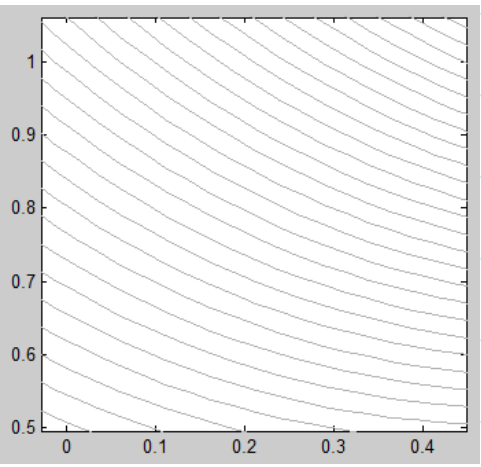
$z^{(2)} =$

$z^{(3)} =$

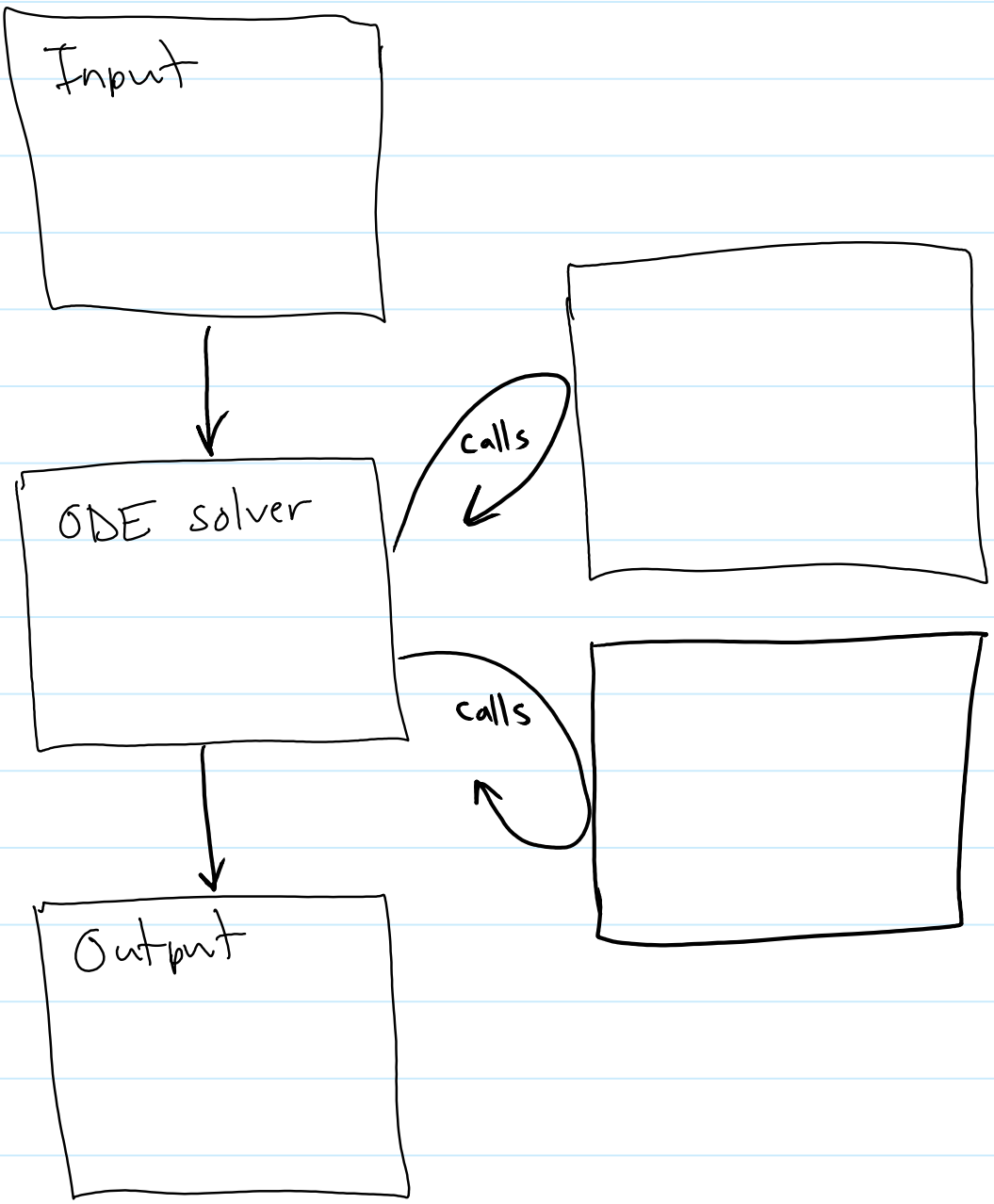
Matlab Example

$$\frac{dy}{dt} = t - y \qquad y(0) = 1$$

Exact solution is $y(t) = 2e^{-t} + t - 1$



System Architecture for Numerical DE Solvers



Matlab's ODE Suite

Matlab has a bunch of built-in ODE solving routines:

ode45, ode23, ode113, ode155, …

Each has its own strengths and weaknesses. Here's how to use them.

Set up the IVP in standard form

- Creat the dynamics function:
- Set the initial state:
- Choose start/end times:

Call the ODE solver

$[t, y] =$

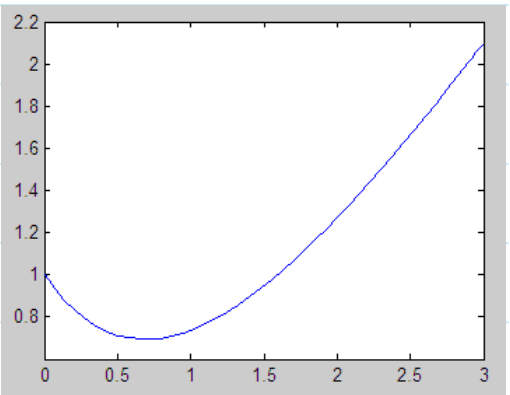
Interpret output

Example

$\frac{dy}{dt} = t - y \qquad y(0) = 1$

```
function dydt = simple_de(t, z)
    dydt = t - z;
```

```
>> [t,y] = ode45(@simple_de, [0 3], 1);
>> plot(t, y);
```



Pass-Through Parameters

Any parameters listed after the 4th parameter in the call to the ODE solver is passed also to the dynamics function.

eg. Recall our previous golf dynamics function

```
function dzdt = simple_golf(t, z)
    % z(1) = x(t)
    % z(2) = y(t)
    % z(3) = y'(t)
    Vx = 30;
    dzdt = [ Vx ; z(3) ; -9.81 ];
```

But we can pass V_x through as a parameter.

```
function dzdt = simple_golf2(t, z, Vx)
    % z(1) = x(t)
    % z(2) = y(t)
    % z(3) = y'(t)
    dzdt = [ Vx ; z(3) ; -9.81 ];
```

Then, to pass V_x from the command where we call the ODE solver…

```
>> [t y] = ode45(@simple_golf2, [0 5], [0;0;20], [], 30);
```

Events and Options in Matlab's ODE Suite

Before calling the ODE solver, you might need to set up some of the options that govern how the solver behaves.
eg.

Use the function "odeset" for this.
For example, to specify an events function…

```
>> opts =
```

Then call the ODE solver with the options structure (stored in "opts").

```
>> [t, y] = ode45(@simple_golf2, [0 5], [0;0;20]
```

A full example…

Novelty Golf Driving Range

Dynamics function: golf.m

Events function: golf_events.m

L31: Higher-Order Methods

Goal: To learn some numerical ODE methods other than Euler's Method.

Recall Euler's Method

$$y_{n+1} =$$

$$l_{n+1} =$$

Problem: In general, we don't know $\hat{y}_n(t_{n+1})$.

Instead, we approximate it with a higher-order method.

Modified Euler Method

This is also called "Improved Euler", and "2nd-order Runge-Kutta".

1. Start with an Euler step

$$y^E_{n+1} = y_n + h_n f(t, y_n)$$

2. Evaluate f at the new point. ie. get derivatives at (t_{n+1}, y^E_{n+1})

3. Use the average of the two slopes

Modiefied Euler is a 2nd-order method. Its local error is

The global error of the Modified Euler method is

We can estimate the local error of Euler's Method using

If this combination of methods was implemented into Matlab, it would be called "ode12".

Adaptive Time-Stepping

We can reduce the local error by simply taking smaller time steps.

However, our method becomes more expensive as we take more and more steps. Thus, choosing a really short time step may not be practical.

We can use our estimate of the local error to choose our time steps.

Pseudocode

1)

2)

3)

4)

5)

Matlab's odeset function allows you to set two bounds on the size of the local error.

AbsTol is the maximum allowable local error

i.e.

RelTol is the maximum allowable relative local error

i.e.

In Matlab, you can set these tolerances using

```
>> opts = odeset('AbsTol', 0.001, 'RelTol', 0.01);
```

Matlab's ODE solvers will accept a time step if it satisfies

AbsTol and RelTol

(ie. it does have to satisfy both)

3rd-Order Runge-Kutta

$$\bar{f}_1 = f(t_n, y_n)$$

$$\bar{f}_2 = f\left(t_n + \frac{h_n}{3}, y_n + \frac{h_n}{3}\bar{f}_1\right)$$

$$\bar{f}_3 = f\left(t_n + \frac{2h_n}{3}, y_n + \frac{2}{3}h_n\bar{f}_2\right)$$

$$y_{n+1} = y_n + h_n\left(\frac{1}{4}\bar{f}_1 + 0\bar{f}_2 + \frac{3}{4}\bar{f}_3\right)$$

$\Rightarrow$ ode23 (Matlab demo: rk3.m)

Numerical Stability

Previously, we talked about the convergence of Euler's method.

Let $h \to 0$ and $n \to \infty$ s.t. $hn = \text{constant}$.
Euler's Method converges to $y(x_n)$ with global error $O(h)$.

Now we study the numerical stability.

h fixed, $n \to \infty$
Study how close y_n remains to $y(x)$ for large x.

We study the "test equation",

$$\begin{cases} y' = \lambda y & (\lambda < 0) \\ y(0) = 1 \end{cases}$$

Note: This is a linear ODE, but any nonlinear ODE can be linearized about a chosen point.

Solution is $y(x) =$

Thus, $\lim_{x \to \infty} y(x) =$

Numerical Stability $\Rightarrow$ we require that

Stability of Euler's Method

$$y_{n+1} = y_n + h f(x_n, y_n)$$
$$=$$
$$=$$

$\lim_{n \to \infty} y_n = 0 \iff$

An Implicit Method

The methods we've looked at so far are called "explicit" methods because we have an explicit formula to calculate the next point based on previous points.

An "implicit" method yields an equation involving the next point, as well as the previous point, but must be solved to find out what the next point is.

Trapezoid Method:

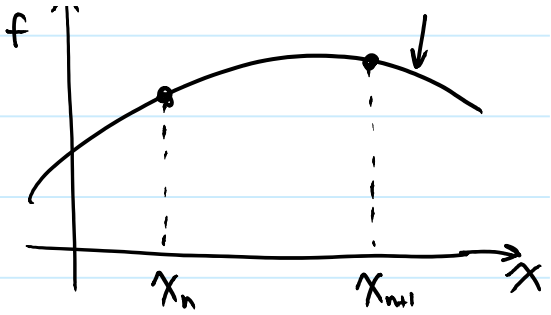
$$y'(x) = f(x,y)$$

Integrate both sides…

$$\int_{x_n}^{x_{n+1}} y'(x)\,dx = \int_{x_n}^{x_{n+1}} f(x,y(x))\,dx$$

If we know $y(x)$, then $\int_{x_n}^{x_{n+1}} y'(x)\,dx = y(x)\Big|_{x=x_n}^{x_{n+1}} = y(x_{n+1}) - y(x_n)$

How do we approximate $\int_{x_n}^{x_{n+1}} f(x,y(x))\,dx$?



- Global error is
- To find y_{n+1}, you have to solve a (possibly non-linear) equation
- Implicit methods tend to be more numerically stable than explicit methods.
  - Useful for "stiff" systems

<u>Stability of Trapezoid Method</u>

Again, $y' = \lambda y$, $y(0) = 1$, $\lambda < 0$

$$y_{n+1} = y_n + \frac{h}{2}\left(f(x_n, y_n) + f(x_{n+1}, y_{n+1})\right)$$

$$y_{n+1} =$$

Notice that

→

can take larger time steps and still be numerically stable.

Stability of Modified Euler: $h <$

Stability of Runge-Kutta 4: $h <$