Outline of topics covered

- Simple SI model structure
- Systems of Ordinary Differential Equations (ODE)
- Conservation principles
- Simple solvers
- Time step constraint (stability)
- Stiff equations
- Flexible time step control
- MATLAB ODE solvers

Simple SI models

Susceptible-Infected (SI) models represent the transmission of disease by contact.
S and I are number of susceptible and infected individuals in a population, respectively.
These ODE represent the rate of susceptible individuals infected by contact transmission with a rate β [fraction of population infected per infected individual].

$$\frac{d\,S}{dt} = -\beta\, I\, S$$
$$\frac{d\,I}{dt} = +\beta\, I\, S$$

Initial population is required: $S(t=0) = S_o$ and $I(t=0) = I_o$, then the numbers can be determined by solving these equations.

SI models represented by coupled ODE

- All SI models are specified as systems of ODE.
- Model coefficients can be constants, functions of time or functions of the model state (S, I).
- Models evolve over time from initial conditions (initial value problem).
- Models are typically solved numerically.

Conservation Principles

Models have conservation principles which must be obeyed by numerical solutions. These can act as tests, constraints or diagnostics.

$$\frac{d\,S}{dt} = -\beta\, I\, S$$
$$\frac{d\,I}{dt} = +\beta\, I\, S$$
$$\frac{d\,(S+I)}{dt} = 0$$

For the simple SI model above, the total population ($S + I = S_o + I_o =$ constant).
All population numbers must be non-negative.

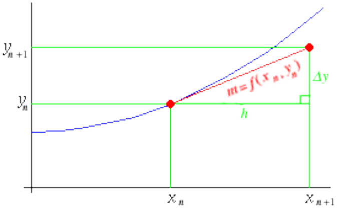
Numerical Solutions to ODE (1)

The basic idea is that the time derivative

$$\frac{d\,S}{dt} = f(S, I, t)$$

is the slope of the solution at the present time (t) and model state (S, I). Moving a short time (dt) along that slope, will estimate the solution at the next time, or

$$S(t + dt) = S(t) + dt\, f(S, I, t) + \text{truncation}$$



Truncation is the error made by making a finite jump. The bigger dt the bigger the error.

This formulation is only one of many ways to convert a continuous derivative into a finite difference statement.

Numerical Solutions to ODE (2)

There are 3 general types of ODE solvers:

- Runge-Kutta methods: Multiple fractional jumps across the time interval dt and average results to get a final estimate.
- Richardson Extrapolation methods: Estimate a (simple) functional form for the slope in the near future and integrate (analytically) the solution.
- Predictor-Corrector methods: Use current and past information to estimate the solution at $t + dt$. Then use the estimated future solution to improve the slope estimate for the final estimate.

We will use Runge-Kutta methods programmed in MATLAB.

Simple Solution Method

Consider the simple mortality equation

$$\frac{d\,S}{dt} = -m\,S$$

$$S(t) = S_o\,e^{-m\,t}$$

then the Euler solution method is

$$S(t + dt) = S(t) - dt\,mS(t) = (1 - dt\,m)S(t)$$

With the condition that $S(t = 0) = S_o$.

Stability and Step Constraints

$$S(t + dt) = S(t) - dt\, mS(t) = (1 - dt\, m)S(t)$$

- Clearly, $dt\, m < 1$ or else the solution will oscillate and grow.
- Even if the solution is stable, the quality of the solution is better as dt becomes smaller.
- But, smaller dt means more steps and more computer time.
- More about step size, but first...

Stiff ODE: Problem with numerical solutions

Stiff ODE are equations with terms having different rates of change. Consider an SI model with recruitment and mortality. For large I, the infection rate ($\beta\, I$) can be large compared to r.

$$\frac{d\,S}{dt} = -\beta\, I\, S + r\, S$$
$$\frac{d\,I}{dt} = +\beta\, I\, S - m\, I$$

The time step of the equations must be small enough to represent these changes.

Flexible step control

Modern ODE solvers internally calculate dt to maintain a stable solution and to maintain a certain quality in the solution.
An example of the method is illustrated by step halving/doubling:

- Calculate $S(t + dt)$ from $S(t)$ using a step size dt.
- Calculate $\hat{S}(t + dt)$ from $S(t)$ using a 2 steps size $dt/2$.
- Compare $\hat{S}(t + dt)$ and $S(t + dt)$ to estimate the truncation error
  - if the trunction error is too small, double dt,
  - if the trunction error is too big, halve dt,

MATLAB ODE solvers use flexible step size to maintain stable, quality solutions.

ODE solvers in MATLAB

- ODE45: A Runge-Kutta based solver which is recommended for non-stiff equations giving moderate quality solutions. This is the generally recommended solver.
- ODE15s: A moderate quality solver for stiff equations. Recommended if ODE45 fails to give a reasonable solution.
- ODE23s: A better solver for stiff equations if ODE15s fails to find an appropriate solution.

Example MATLAB usage:

```
nVar=2;
y0=zeros(nVar,1); % initial conditions
tspan=[0 100]; % time span
% RHS is a matlab function defining the ODEs
[t,y]=ode45(@RHS,tspan,y0);
```