ODE Background: Introduction to SIR Epidemic Model and Simulating ODE Models in MATLAB

Alun L. Lloyd

Department of Mathematics
Biomathematics Graduate Program
North Carolina State University

Aims of These Slides

You will learn:

- Setup of a simple epidemic model
- A little about how the model behaves (will see more detail on this in the practical during the workshop)
- How to simulate the model in MATLAB

I assume you know about functions and M-files in MATLAB. If not, see:

https://www.mathworks.com/help/matlab/matlab_prog/create-functions-in-files.html

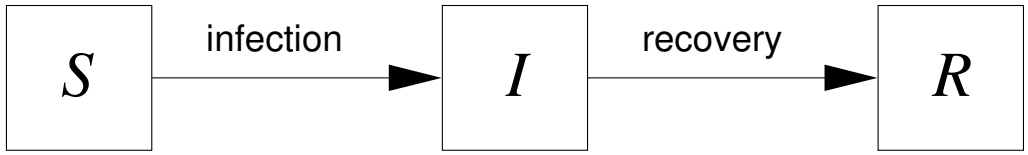
or

https://www.youtube.com/watch?v=qo3AtBoyBdM

SIR Model for Spread of Infection

Compartmental model, with three **state variables**:
Susceptibles, Infectives, Recovereds

Model tracks rate of change of these quantities



$$dS/dt = -\beta SI/N$$

$$dI/dt = \beta SI/N - \gamma I$$

$$dR/dt = \gamma I$$

Ignore births and deaths (e.g. short-lived outbreak)

"Standard incidence" term $\beta SI/N$ describes transmission process

β : "transmission parameter"
"well-mixed" population

Assume constant per-capita recovery rate of γ

$1/\gamma$ is average duration of infectiousness

Note: *S* + *I* + *R* = *N* (constant), so need only worry about *S* and *I*

Behavior of SIR Model

It turns out that model's behavior is governed by the value of the ratio $R_0 = \beta/\gamma$

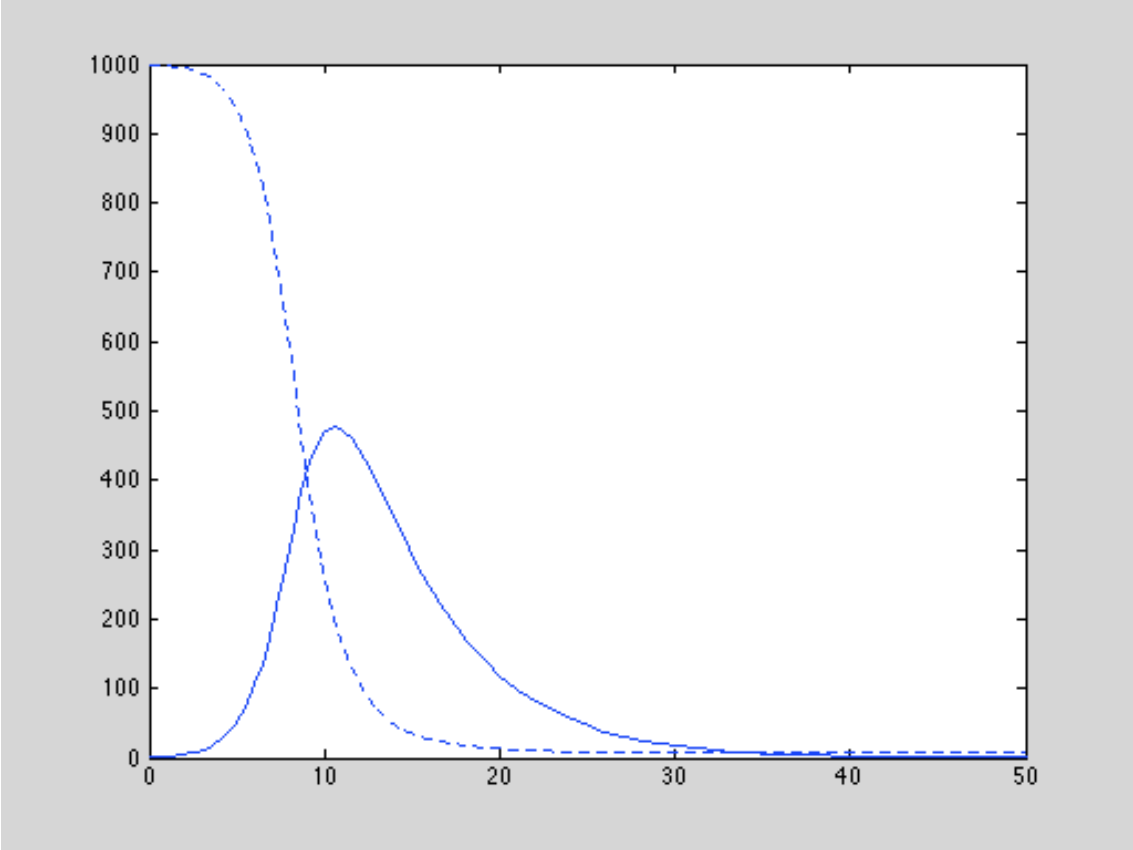
Outbreak can occur if $R_0 > 1$, cannot occur if $R_0 < 1$

$R_0 > 1$ plot:

$\beta = 1$, $\gamma = 0.2$, $N = 1000$
$S(0) = 999$, $I(0) = 1$

$S(t)$: dashed line

$I(t)$: solid line



Behavior of SIR Model

Behavior is governed by the value of the ratio $R_0 = \beta/\gamma$

Outbreak can occur if $R_0 > 1$, cannot occur if $R_0 < 1$

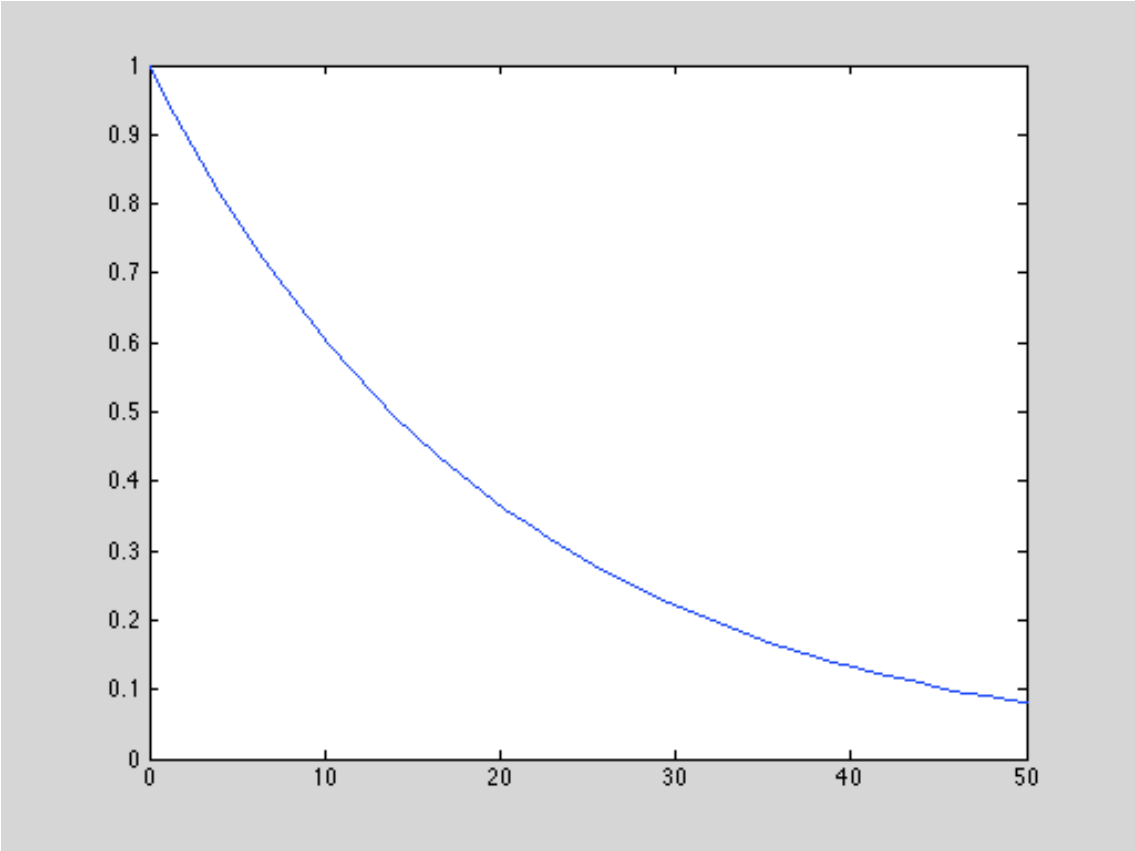
$R_0 < 1$ plot:

$\beta = 0.15$, $\gamma = 0.2$, $N = 1000$
$S(0) = 999$, $I(0) = 1$

$S(t)$: not shown (remains close to 999)

$I(t)$: solid line

note different scale on vertical axis



SIR Model : Forward Simulation

Nonlinearity of the transmission term means we cannot find an analytic solution of the model for *S* and *I* in terms of time

Numerically integrate (simulate) model in MATLAB, given a set of parameters and initial values for *S* and *I*

We shall use the `ode45` routine in MATLAB (syntax explained in two slides' time) to integrate the general initial value problem

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}) \quad ; \quad \mathbf{y}(0) = \mathbf{y}_0$$

Here **y** is a vector of states of the system (i.e. *S* and *I* for our SIR model), **f** is a vector function giving the time derivatives of the states, and $\mathbf{y}_0$ is a vector of initial conditions

SIR Model : Forward Simulation

We shall use the `ode45` routine in MATLAB. Syntax explained on the next slide

We need to tell ode45:

- the name of the function that calculates the values of the time derivatives of state variables (i.e. the components of the right hand sides of the differential equation)
- the time interval over which to integrate, and
- the initial conditions

Optionally, we can pass options that `ode45` is to use (e.g. integration tolerances) and a vector of parameters

MATLAB works with state variables arranged in a vector, so we shall use the first element (e.g. `y(1)`) to denote *S* and the second (e.g. `y(2)`) to denote *I*

ode45

```
[t,y]=ode45(@odefun,tspan,y0,options,pars);
```

`odefun` the name of the function that gives the right sides of our differential equations
(replace "odefun" with something more descriptive, but keep "@")

`tspan` vector that specifies the interval of times over which to integrate:

```
tspan = [t_initial, t_final]
```

or a vector of times at which we wish to obtain output :

```
tspan = [t_initial, t1, t2, … , t_final]
```

`y0` column vector of initial states (i.e. initial conditions) : `y0 = [ S0 ; I0 ]`

`options` options for the ODE solver, e.g. solution tolerances
use `[ ]` for no options; see `odeset` for information on options

`pars` a vector of parameter values that gets passed to `odefun`

`t` (returned) column vector of times at which output is given

`y` (returned) matrix of numerically calculated values of state variables over time

each row refers to a different time point, each column to a different state variable
e.g. `y(1,:)` are initial states, `y(end,:)` final states,
`y(:,2)` is a column vector of *I* values at all times — this is what we want to make an *I*(*t*) vs *t* plot

odefun

```
function f = odefun(t,y,pars)
```

Function `odefun` returns the entries of the right sides of the differential equations, $f(t,\mathbf{y})$, as a column vector

`t` (scalar) value of time at which to evaluate *f*

`y` column vector containing values of state variables

`pars` a vector of parameter values that gets passed to `odefun`

```
function f = sir_rhs(t,y,pars)

    f=zeros(2,1);

    beta=pars(1);
    gamma=pars(2);
    N=pars(3);

    S=y(1);
    I=y(2);

    f(1)=-beta*S*I/N;
    f(2)=beta*S*I/N-gamma*I;
end
```

need to return a column vector

could eliminate a number of these lines if we worked with y(1), pars(1) etc in the f(1) and f(2) lines

SIR Model Simulation

```
function sir_simulation

    beta=1.0;
    gamma=1.0/5.0;    % five day infectious period
    N=1000.0;

    pars=[beta,gamma,N];

    tspan=[0,50];      % simulate for 50 days

    y0=[999;1];        % one initial infective

    [t,y]=ode45(@sir_rhs,tspan,y0,[],pars);

    plot(t,y(:,2));   % plot prevalence of infection over time
end

function f = sir_rhs(t,y,pars)
    f=zeros(2,1);
    f(1)=-pars(1)*y(1)*y(2)/pars(3);
    f(2)=pars(1)*y(1)*y(2)/pars(3)-pars(2)*y(2);
end
```