Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

851-0585-04L – Modeling and Simulating Social Systems with MATLAB

Lecture 3 – Dynamical Systems

Karsten Donnay and Stefano Balietti

Chair of Sociology, in particular of

Modeling and Simulation

Schedule of the course

Introduction to MATLAB	20.02.	
	27.02.	
	05.03.	Introduction to social-science modeling and simulations
	12.03.	
Working on projects (seminar thesis)	19.03.	
	26.03.	
	02.04.	
	23.04.	
	30.04.	
	07.05.	
	14.05.	
	21.05.	Handing in seminar thesis and giving a presentation
	28.05.	

Schedule of the course

Topic	Date	Topic
Introduction to MATLAB	20.02.	
Introduction to MATLAB	27.02.	
	05.03.	Introduction to social-science modeling and simulations; Create and Submit a Research Plan
	12.03.	Introduction to social-science modeling and simulations
Working on projects (seminar thesis)	19.03.	Introduction to social-science modeling and simulations
Working on projects (seminar thesis)	26.03.	Introduction to social-science modeling and simulations
Working on projects (seminar thesis)	02.04.	Introduction to social-science modeling and simulations
Working on projects (seminar thesis)	23.04.	Introduction to social-science modeling and simulations
Working on projects (seminar thesis)	30.04.	Introduction to social-science modeling and simulations
Working on projects (seminar thesis)	07.05.	Introduction to social-science modeling and simulations
Working on projects (seminar thesis)	14.05.	Introduction to social-science modeling and simulations
Working on projects (seminar thesis)	21.05.	Handing in seminar thesis and giving a presentation
	28.05.	Handing in seminar thesis and giving a presentation

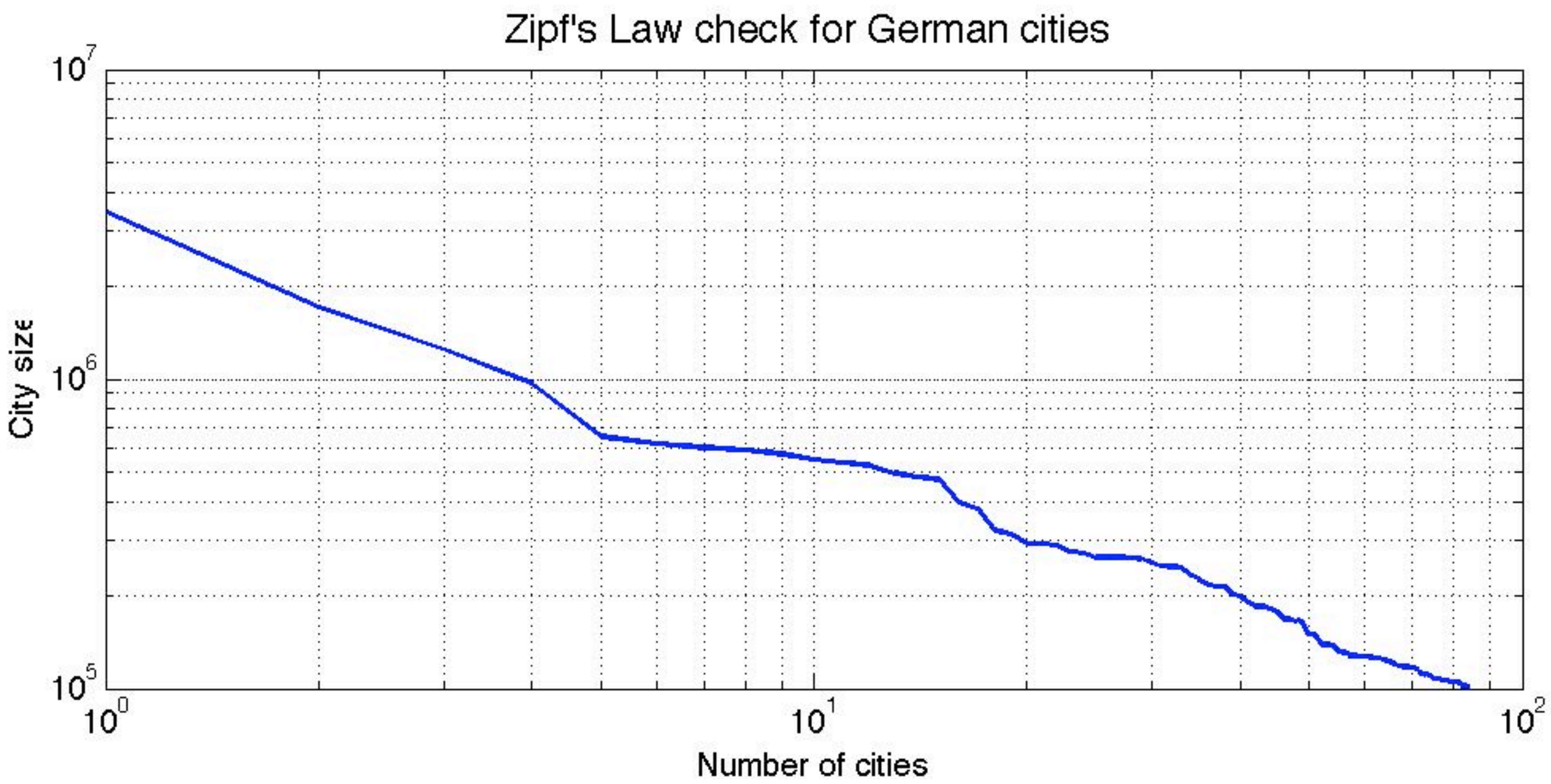
Goals of Lecture 3: students will

1. Consolidate knowledge acquired during lecture 2, through *brief* repetition of the main points and revision of the exercises.
2. Understand *fully* the importance and the **role** of a **Research Plan** and its main standard components.
3. Learn the *basic* definition of **Dynamical Systems** and Differential Equations.
4. Implement Dynamical System in MATLAB a series of examples from Physics and Social Science Literature (Pendulum, Lorenz attractor, Lotka-Volterra equations, Epidemics: Kermack-McKendrick model)

Repetition

- `plot()`, `loglog()`, `semilogy()`, `semilogx()`
- `hist()`, `sort()`
- `subplot()` vs `figure()`
- Log-scale plots reduce the complexity of some functional forms and fit well to draw large range of values
- The amplitude of the sample matters! **law of large numbers – LLN**. (Ex.1)
- Outlier observations can significantly skew the distribution, e.g. mean != median. (Ex. 2)
- `Cov()` and `corrcoef()` can measure correlation among variables. (Ex. 3)

Repetition – Ex. 4




```
>> loglog(germany(:,2),'LineWidth',3);
```

Project

- Implementation of a model from the Social Science literature in MATLAB

- During the next two weeks:
  - Form a group of two to four persons
  - Choose a topic among the project suggestions (available on line) or propose your own idea
  - Set up a GitHub repository for your group
  - Complete the research proposal (It is the readme file of your GitHub repository)

Research Plan Structure

- 1-2 (not more!) pages stating:

- Brief, general introduction to the problem
- How you abstract the problem with a model
- Fundamental questions you want to try to answer
- Existing literature you will base your model on and possible extensions
- Research methods you are planning to use

- Keep track of the changes

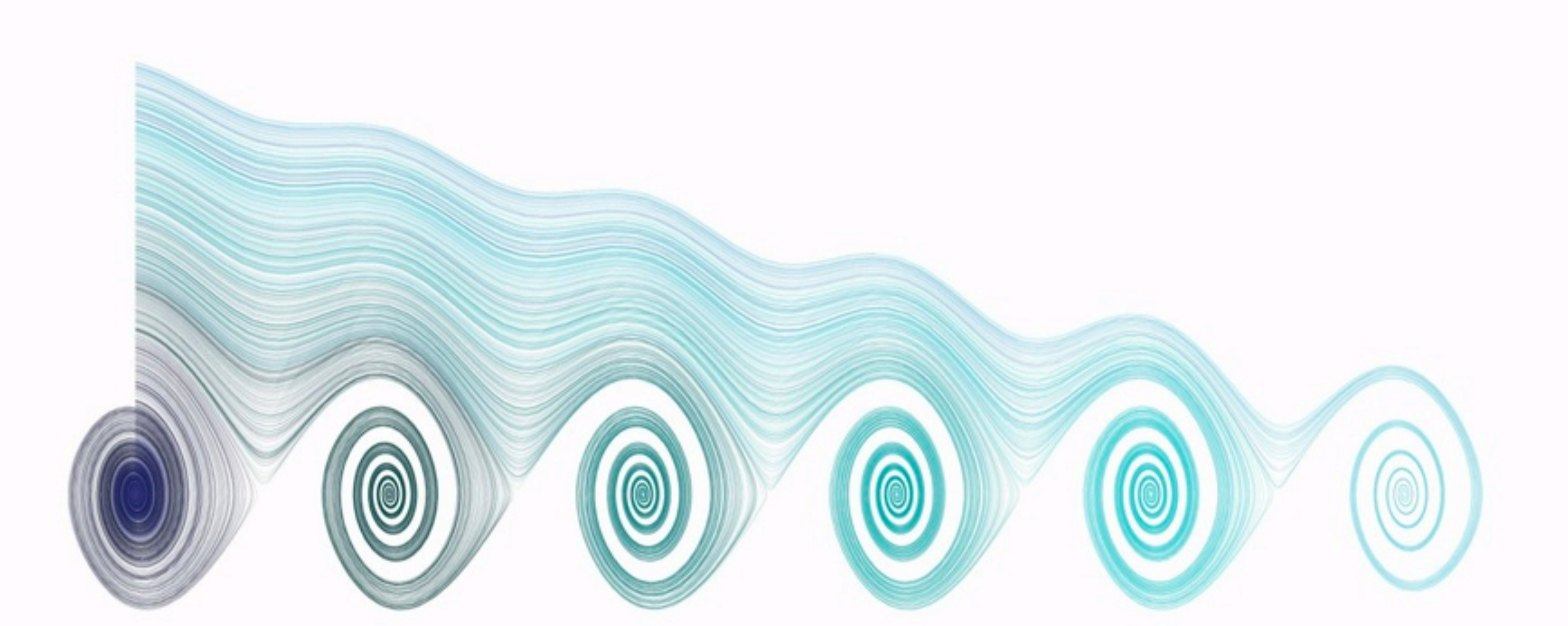
Research Plan

- Upon submission, the Research Plan can:
  - be accepted
  - be accepted under revision
  - be modified later on only if change is justified (create new version)

- Talk to us if you are not sure
- Final deadline for signing-up for a project is:

March 19th 2012

Dynamical systems



- Mathematical description of the time dependence of variables that characterize a given problem/scenario in its state space.

Dynamical systems

- A dynamical system is described by a set of linear/non-linear differential equations.
- Even though an analytical treatment of dynamical systems is usually very complicated, obtaining a numerical solution is (often) straight forward.
- Differential equation and difference equation are two different tools for operating with Dynamical Systems

K. Donnay & S. Balietti / kdonnay@ethz.ch sbalietti@ethz.ch

Differential Equations

- A **differential equation** is a mathematical equation for an *unknown function* (dependent variable) of one or more (independent) *explicative variables* that relates the values of the function itself and its derivatives of various orders
- Ordinary (ODE) : $\dfrac{df(x)}{dx} = f(x)$
- Partial (PDE) : $\dfrac{\partial f(x,y)}{\partial x} + \dfrac{\partial f(x,y)}{\partial y} = f(x,y)$

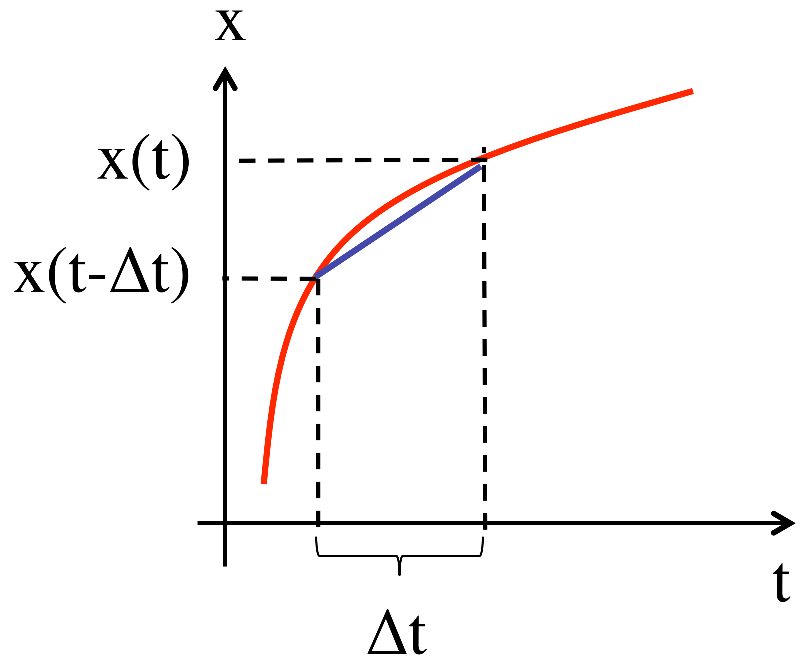
Numerical Solutions for Differential Equations

- Solving differential equations numerically can be done by a number of schemes. The easiest way is by the 1st order Euler's Method:

$$\frac{dx}{dt} = f(x,\ldots)$$

$$\frac{x(t) - x(t-\Delta t)}{\Delta t} = f(x,\ldots)$$

$$x(t) = x(t-\Delta t) + \Delta t\; f(x,..)$$



A famous example: Pendulum

- A pendulum is a simple dynamical system:

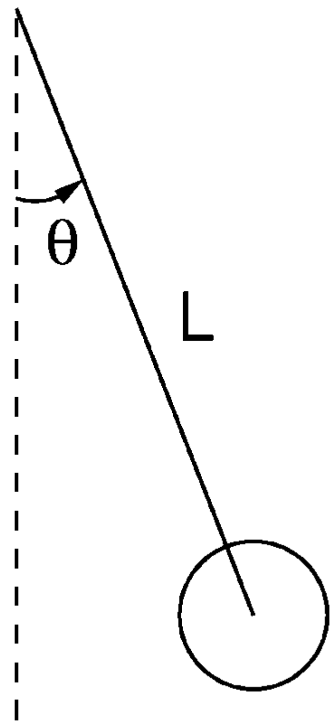
L = length of pendulum (m)

Θ = angle of pendulum

g = acceleration due to gravity (m/s^2)

The motion is described by:

$$\theta'' = -\frac{g}{L}\sin(\theta)$$



K. Donnay & S. Balietti / kdonnay@ethz.ch sbalietti@ethz.ch

Pendulum: MATLAB code

```
dt=0.01;            % time step
g=9.81;             % gravity
L=1.0;              % pendulum length

% initial condition
theta=0.5;          % angle
thetaPrime=0;       % angular velocity

% simulation loop
for t=0:dt:5
    thetaBis=-g/L*sin(theta);
    thetaPrime=thetaPrime+dt*thetaBis;
    theta=theta+dt*thetaPrime;
    plot([0 cos(theta-pi/2)],[0 sin(theta-pi/2)]);
    xlim([-1 1]);
    ylim([-1 0]);
    xlabel('x');
    ylabel('y');
    pause(0.01);
end
```

Set time step

```
dt=0.01;           % time step
g=9.81;            % gravity
L=1.0;             % pendulum length

% initial condition
theta=0.5;         % angle
thetaPrime=0;      % angular velocity

% simulation loop
for t=0:dt:5
    thetaBis=-g/L*sin(theta);
    thetaPrime=thetaPrime+dt*thetaBis;
    theta=theta+dt*thetaPrime;
    plot([0 cos(theta-pi/2)],[0 sin(theta-pi/2)]);
    xlim([-1 1]);
    ylim([-1 0]);
    xlabel('x');
    ylabel('y');
    pause(0.01);
end
```

Set constants

```
dt=0.01;            % time step
g=9.81;             % gravity
L=1.0;              % pendulum length

% initial condition
theta=0.5;          % angle
thetaPrime=0;       % angular velocity

% simulation loop
for t=0:dt:5
    thetaBis=-g/L*sin(theta);
    thetaPrime=thetaPrime+dt*thetaBis;
    theta=theta+dt*thetaPrime;
    plot([0 cos(theta-pi/2)],[0 sin(theta-pi/2)]);
    xlim([-1 1]);
    ylim([-1 0]);
    xlabel('x');
    ylabel('y');
    pause(0.01);
end
```

Set starting point of pendulum

```
dt=0.01;            % time step
g=9.81;             % gravity
L=1.0;              % pendulum length

% initial condition
theta=0.5;          % angle
thetaPrime=0;       % angular velocity

% simulation loop
for t=0:dt:5
    thetaBis=-g/L*sin(theta);
    thetaPrime=thetaPrime+dt*thetaBis;
    theta=theta+dt*thetaPrime;
    plot([0 cos(theta-pi/2)],[0 sin(theta-pi/2)]);
    xlim([-1 1]);
    ylim([-1 0]);
    xlabel('x');
    ylabel('y');
    pause(0.01);
end
```

Time loop: Simulate the pendulum

```
dt=0.01;            % time step
g=9.81;             % gravity
L=1.0;              % pendulum length

% initial condition
theta=0.5;          % angle
thetaPrime=0;       % angular velocity

% simulation loop
for t=0:dt:5
    thetaBis=-g/L*sin(theta);
    thetaPrime=thetaPrime+dt*thetaBis;
    theta=theta+dt*thetaPrime;
    plot([0 cos(theta-pi/2)],[0 sin(theta-pi/2)]);
    xlim([-1 1]);
    ylim([-1 0]);
    xlabel('x');
    ylabel('y');
    pause(0.01);
end
```

Perform 1st order Euler's method

```
dt=0.01;            % time step
g=9.81;             % gravity
L=1.0;              % pendulum length

% initial condition
theta=0.5;          % angle
thetaPrime=0;       % angular velocity

% simulation loop
for t=0:dt:5
    thetaBis=-g/L*sin(theta);
    thetaPrime=thetaPrime+dt*thetaBis;
    theta=theta+dt*thetaPrime;
    plot([0 cos(theta-pi/2)],[0 sin(theta-pi/2)]);
    xlim([-1 1]);
    ylim([-1 0]);
    xlabel('x');
    ylabel('y');
    pause(0.01);
end
```

Plot pendulum

```
dt=0.01;            % time step
g=9.81;             % gravity
L=1.0;              % pendulum length

% initial condition
theta=0.5;          % angle
thetaPrime=0;       % angular velocity

% simulation loop
for t=0:dt:5
    thetaBis=-g/L*sin(theta);
    thetaPrime=thetaPrime+dt*thetaBis;
    theta=theta+dt*thetaPrime;
    plot([0 cos(theta-pi/2)],[0 sin(theta-pi/2)]);
    xlim([-1 1]);
    ylim([-1 0]);
    xlabel('x');
    ylabel('y');
    pause(0.01);
end
```

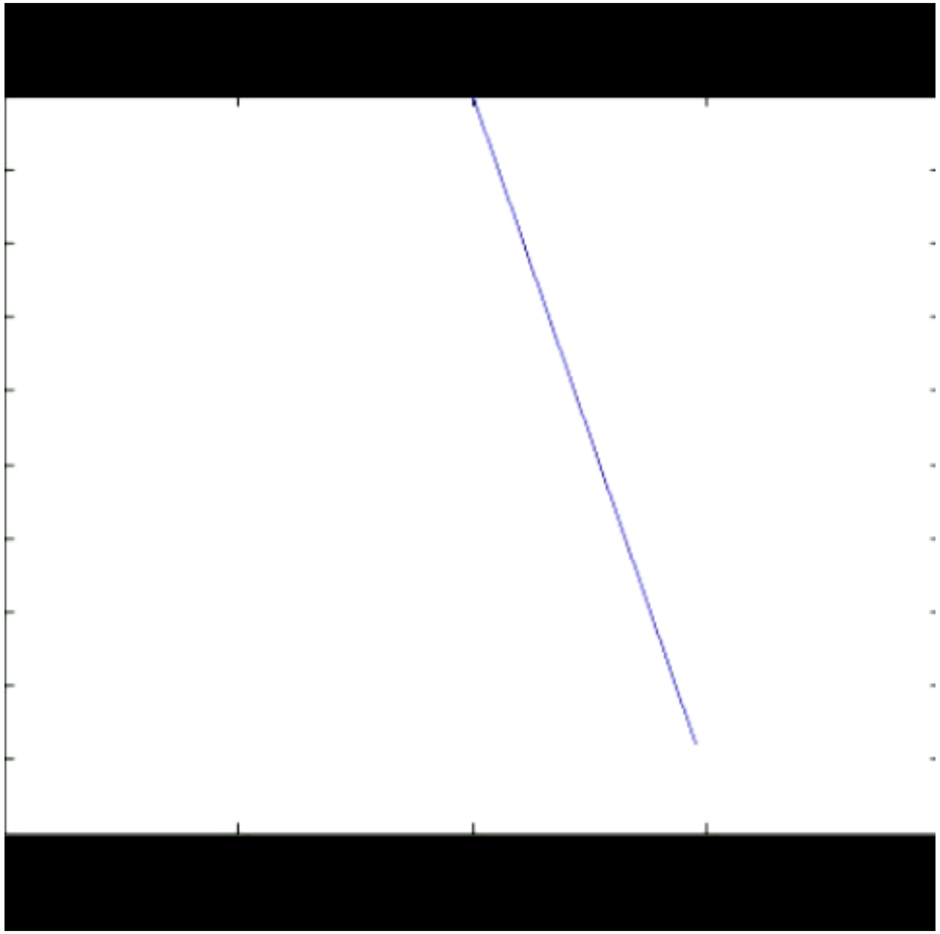
Set limits of window

```
dt=0.01;           % time step
g=9.81;            % gravity
L=1.0;             % pendulum length

% initial condition
theta=0.5;         % angle
thetaPrime=0;      % angular velocity

% simulation loop
for t=0:dt:5
    thetaBis=-g/L*sin(theta);
    thetaPrime=thetaPrime+dt*thetaBis;
    theta=theta+dt*thetaPrime;
    plot([0 cos(theta-pi/2)],[0 sin(theta-pi/2)]);
    xlim([-1 1]);
    ylim([-1 0]);
    xlabel('x');
    ylabel('y');
    pause(0.01);
end
```

Make a 10 ms pause

```
dt=0.01;            % time step
g=9.81;             % gravity
L=1.0;              % pendulum length

% initial condition
theta=0.5;          % angle
thetaPrime=0;       % angular velocity

% simulation loop
for t=0:dt:5
    thetaBis=-g/L*sin(theta);
    thetaPrime=thetaPrime+dt*thetaBis;
    theta=theta+dt*thetaPrime;
    plot([0 cos(theta-pi/2)],[0 sin(theta-pi/2)]);
    xlim([-1 1]);
    ylim([-1 0]);
    xlabel('x');
    ylabel('y');
    pause(0.01);
end
```

Pendulum: Executing MATLAB code

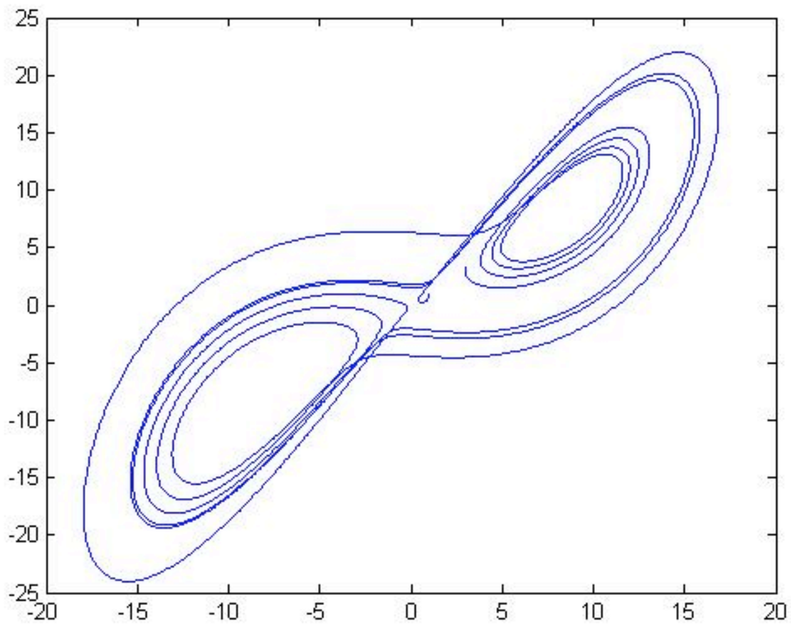


```
dt=0.01;            % time step
g=9.81;             % gravity
L=1.0;              % pendulum length

% initial condition
theta=0.5;          % angle
thetaPrime=0;       % angular velocity

% simulation loop
for t=0:dt:5
    thetaBis=-g/L*sin(theta);
    thetaPrime=thetaPrime+dt*thetaBis;
    theta=theta+dt*thetaPrime;
    plot([0 cos(theta-pi/2)],[0 sin(theta-pi/2)]);
    xlim([-1 1]);
    ylim([-1 0]);
    xlabel('x');
    ylabel('y');
    pause(0.01);
end
```

The file `pendulum.m` may be found on the website!

Lorenz attractor

- The Lorenz attractor defines a 3-dimensional trajectory by the differential equations:



$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = rx - y - xz$$

$$\frac{dz}{dt} = xy - bz$$

- σ, r, b are parameters.

Lorenz attractor: MATLAB code

```
dt=0.001;     % timestep
iter=5000;    % iterations
x(1)=1; y(1)=1; z(1)=40; t(1)=0;      % starting point
sigma=10; r=28; b=8/3;   % parameters


for i=2:iter
    dx = sigma*(y(i-1)-x(i-1));
    dy = r*x(i-1) - y(i-1) - x(i-1)*z(i-1);
    dz = x(i-1)*y(i-1) - b*z(i-1);
    x(i) = x(i-1) + dt*dx;
    y(i) = y(i-1) + dt*dy;
    z(i) = z(i-1) + dt*dz;
    t(i) = t(i-1) + dt;
    xlim([-20 20]);
    ylim([-25 25]);
end

plot (x,y);
```

Set time step

```
dt=0.001;     % timestep
iter=5000;    % iterations
x(1)=1; y(1)=1; z(1)=40; t(1)=0;     % starting point
sigma=10; r=28; b=8/3;   % parameters


for i=2:iter
    dx = sigma*(y(i-1)-x(i-1));
    dy = r*x(i-1) - y(i-1) - x(i-1)*z(i-1);
    dz = x(i-1)*y(i-1) - b*z(i-1);
    x(i) = x(i-1) + dt*dx;
    y(i) = y(i-1) + dt*dy;
    z(i) = z(i-1) + dt*dz;
    t(i) = t(i-1) + dt;
    xlim([-20 20]);
    ylim([-25 25]);
end

plot (x,y);
```

Set number of iterations

```
dt=0.001;     % timestep
iter=5000;    % iterations
x(1)=1; y(1)=1; z(1)=40; t(1)=0;     % starting point
sigma=10; r=28; b=8/3;  % parameters


for i=2:iter
    dx = sigma*(y(i-1)-x(i-1));
    dy = r*x(i-1) - y(i-1) - x(i-1)*z(i-1);
    dz = x(i-1)*y(i-1) - b*z(i-1);
    x(i) = x(i-1) + dt*dx;
    y(i) = y(i-1) + dt*dy;
    z(i) = z(i-1) + dt*dz;
    t(i) = t(i-1) + dt;
    xlim([-20 20]);
    ylim([-25 25]);
end

plot (x,y);
```

Set initial values

```
dt=0.001;    % timestep
iter=5000;   % iterations
x(1)=1; y(1)=1; z(1)=40; t(1)=0;     % starting point
sigma=10; r=28; b=8/3;     % parameters

for i=2:iter
    dx = sigma*(y(i-1)-x(i-1));
    dy = r*x(i-1) - y(i-1) - x(i-1)*z(i-1);
    dz = x(i-1)*y(i-1) - b*z(i-1);
    x(i) = x(i-1) + dt*dx;
    y(i) = y(i-1) + dt*dy;
    z(i) = z(i-1) + dt*dz;
    t(i) = t(i-1) + dt;
    xlim([-20 20]);
    ylim([-25 25]);
end

plot (x,y);
```

Set parameters

```
dt=0.001;     % timestep
iter=5000;    % iterations
x(1)=1; y(1)=1; z(1)=40; t(1)=0;      % starting point
sigma=10; r=28; b=8/3;  % parameters

for i=2:iter
    dx = sigma*(y(i-1)-x(i-1));
    dy = r*x(i-1) - y(i-1) - x(i-1)*z(i-1);
    dz = x(i-1)*y(i-1) - b*z(i-1);
    x(i) = x(i-1) + dt*dx;
    y(i) = y(i-1) + dt*dy;
    z(i) = z(i-1) + dt*dz;
    t(i) = t(i-1) + dt;
    xlim([-20 20]);
    ylim([-25 25]);
end

plot (x,y);
```

Solve the Lorenz-attractor equations

```
dt=0.001;     % timestep
iter=5000;    % iterations
x(1)=1; y(1)=1; z(1)=40; t(1)=0;     % starting point
sigma=10; r=28; b=8/3;   % parameters

for i=2:iter
    dx = sigma*(y(i-1)-x(i-1));
    dy = r*x(i-1) - y(i-1) - x(i-1)*z(i-1);
    dz = x(i-1)*y(i-1) - b*z(i-1);
    x(i) = x(i-1) + dt*dx;
    y(i) = y(i-1) + dt*dy;
    z(i) = z(i-1) + dt*dz;
    t(i) = t(i-1) + dt;
    xlim([-20 20]);
    ylim([-25 25]);
end

plot (x,y);
```

Compute gradient

```
dt=0.001;     % timestep
iter=5000;    % iterations
x(1)=1; y(1)=1; z(1)=40; t(1)=0;      % starting point
sigma=10; r=28; b=8/3;   % parameters


for i=2:iter
    dx = sigma*(y(i-1)-x(i-1));
    dy = r*x(i-1) - y(i-1) - x(i-1)*z(i-1);
    dz = x(i-1)*y(i-1) - b*z(i-1);
    x(i) = x(i-1) + dt*dx;
    y(i) = y(i-1) + dt*dy;
    z(i) = z(i-1) + dt*dz;
    t(i) = t(i-1) + dt;
    xlim([-20 20]);
    ylim([-25 25]);
end

plot (x,y);
```

Perform 1st order Euler's method

```
dt=0.001;    % timestep
iter=5000;   % iterations
x(1)=1; y(1)=1; z(1)=40; t(1)=0;     % starting point
sigma=10; r=28; b=8/3;   % parameters


for i=2:iter
    dx = sigma*(y(i-1)-x(i-1));
    dy = r*x(i-1) - y(i-1) - x(i-1)*z(i-1);
    dz = x(i-1)*y(i-1) - b*z(i-1);
    x(i) = x(i-1) + dt*dx;
    y(i) = y(i-1) + dt*dy;
    z(i) = z(i-1) + dt*dz;
    t(i) = t(i-1) + dt;
    xlim([-20 20]);
    ylim([-25 25]);
end

plot (x,y);
```

Update time

```
dt=0.001;    % timestep
iter=5000;   % iterations
x(1)=1; y(1)=1; z(1)=40; t(1)=0;     % starting point
sigma=10; r=28; b=8/3;   % parameters


for i=2:iter
    dx = sigma*(y(i-1)-x(i-1));
    dy = r*x(i-1) - y(i-1) - x(i-1)*z(i-1);
    dz = x(i-1)*y(i-1) - b*z(i-1);
    x(i) = x(i-1) + dt*dx;
    y(i) = y(i-1) + dt*dy;
    z(i) = z(i-1) + dt*dz;
    t(i) = t(i-1) + dt;
    xlim([-20 20]);
    ylim([-25 25]);
end

plot (x,y);
```

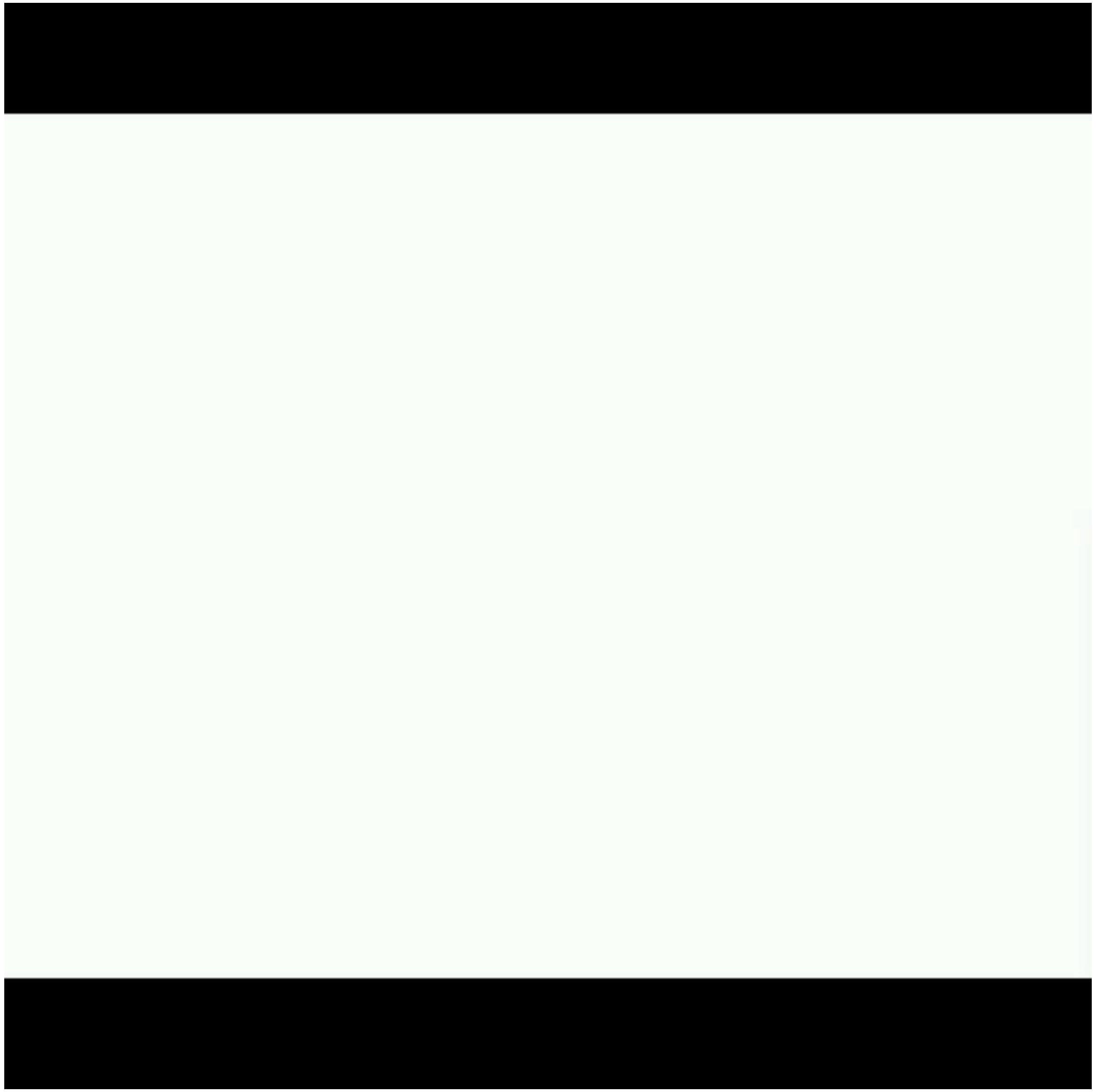
Plot the results

```
dt=0.001;     % timestep
iter=5000;    % iterations
x(1)=1; y(1)=1; z(1)=40; t(1)=0;     % starting point
sigma=10; r=28; b=8/3;    % parameters


for i=2:iter
    dx = sigma*(y(i-1)-x(i-1));
    dy = r*x(i-1) - y(i-1) - x(i-1)*z(i-1);
    dz = x(i-1)*y(i-1) - b*z(i-1);
    x(i) = x(i-1) + dt*dx;
    y(i) = y(i-1) + dt*dy;
    z(i) = z(i-1) + dt*dz;
    t(i) = t(i-1) + dt;
    xlim([-20 20]);
    ylim([-25 25]);
end

plot (x,y);
```

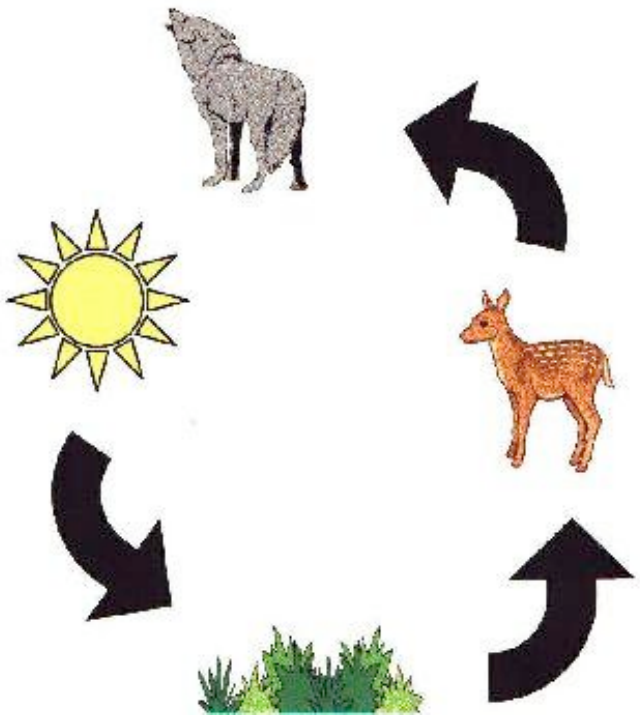
Animation

```
dt=0.001;    % timestep
iter=5000;   % iterations
x(1)=1; y(1)=1; z(1)=40; t(1)=0;     % starting point
sigma=10; r=28; b=8/3;   % parameters


for i=2:iter
    dx = sigma*(y(i-1)-x(i-1));
    dy = r*x(i-1) - y(i-1) - x(i-1)*z(i-1);
    dz = x(i-1)*y(i-1) - b*z(i-1);
    x(i) = x(i-1) + dt*dx;
    y(i) = y(i-1) + dt*dy;
    z(i) = z(i-1) + dt*dz;
    t(i) = t(i-1) + dt;
    xlim([-20 20]);
    ylim([-25 25]);
end
plot (x,y);
```

The file `lorenzattractor.m` may be found on the website!



Food chain



Lotka-Volterra equations

- The Lotka-Volterra equations describe the interaction between two species, prey vs. predators, e.g. rabbits vs. foxes.

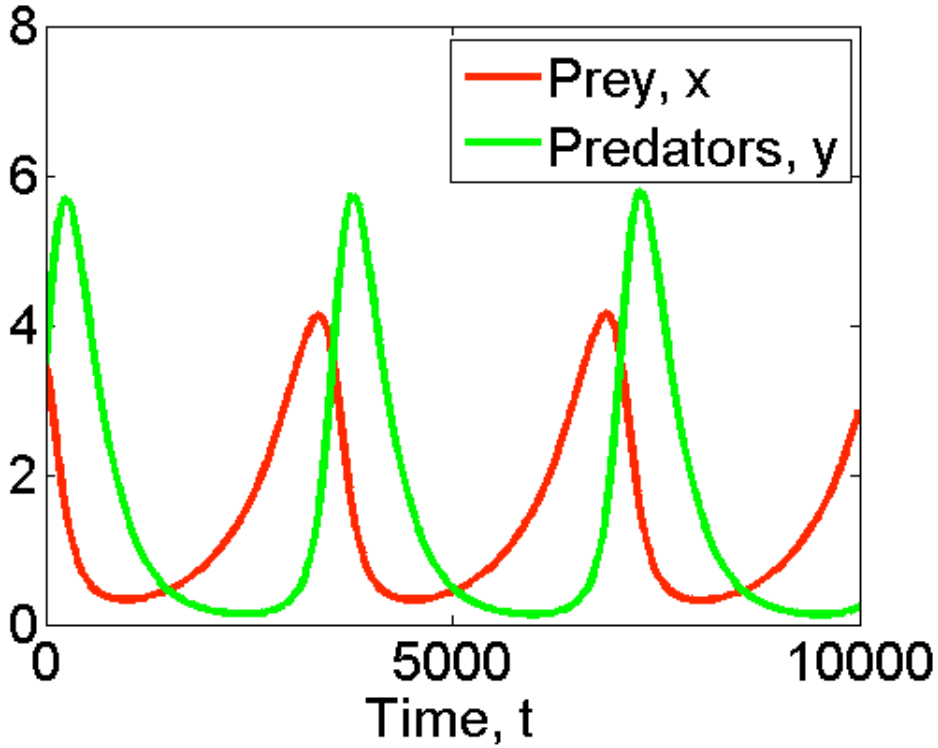
x: number of prey
y: number of predators
α, β, γ, δ: parameters

$$\frac{dx}{dt} = x(\alpha - \beta y)$$

$$\frac{dy}{dt} = -y(\gamma - \delta x)$$

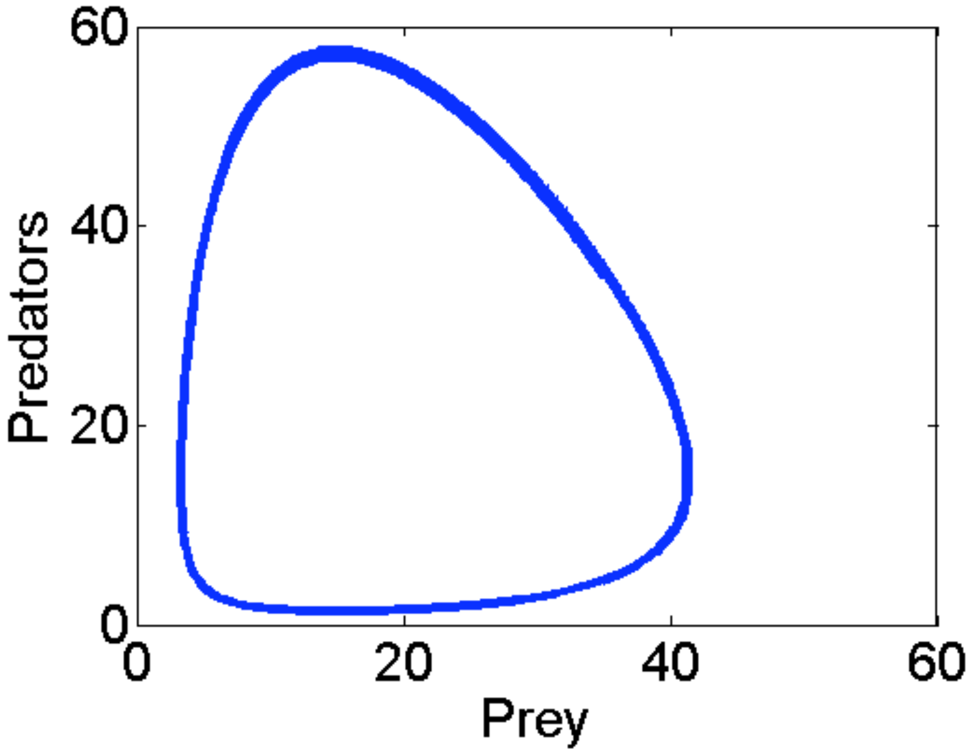
Lotka-Volterra equations

- The Lotka-Volterra equations describe the interaction between two species, prey vs. predators, e.g. rabbits vs. foxes.

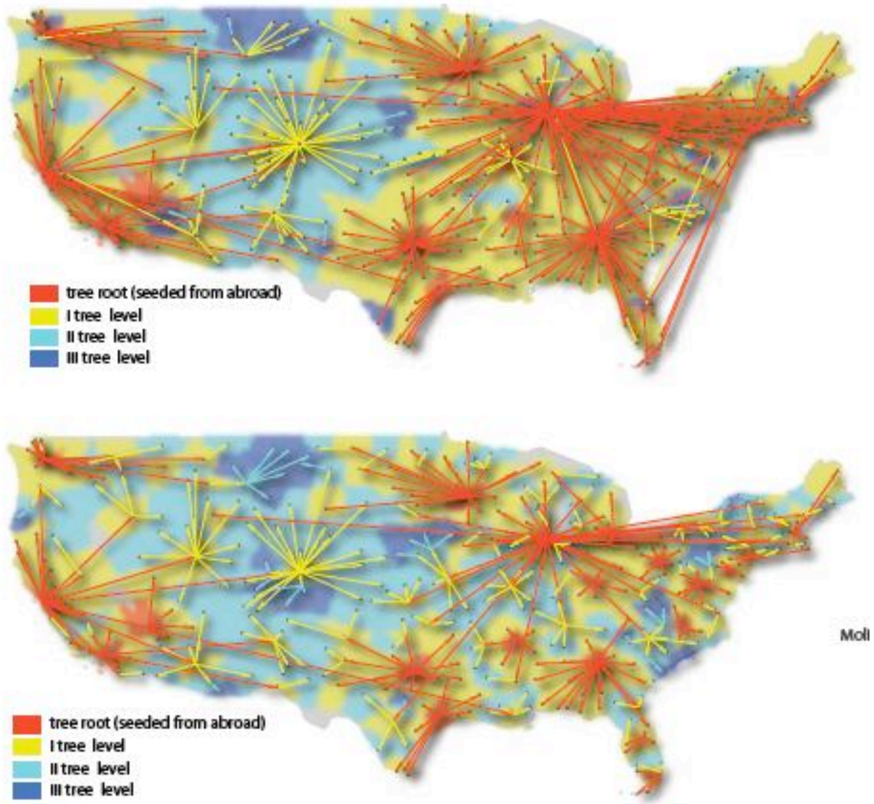


Lotka-Volterra equations

- The Lotka-Volterra equations describe the interaction between two species, prey vs. predators, e.g. rabbits vs. foxes.



Epidemics



Source: Balcan, et al. 2009

SIR model

- A general model for epidemics is the SIR model, which describes the interaction between Susceptible, Infected and Removed (immune) persons, for a given disease.

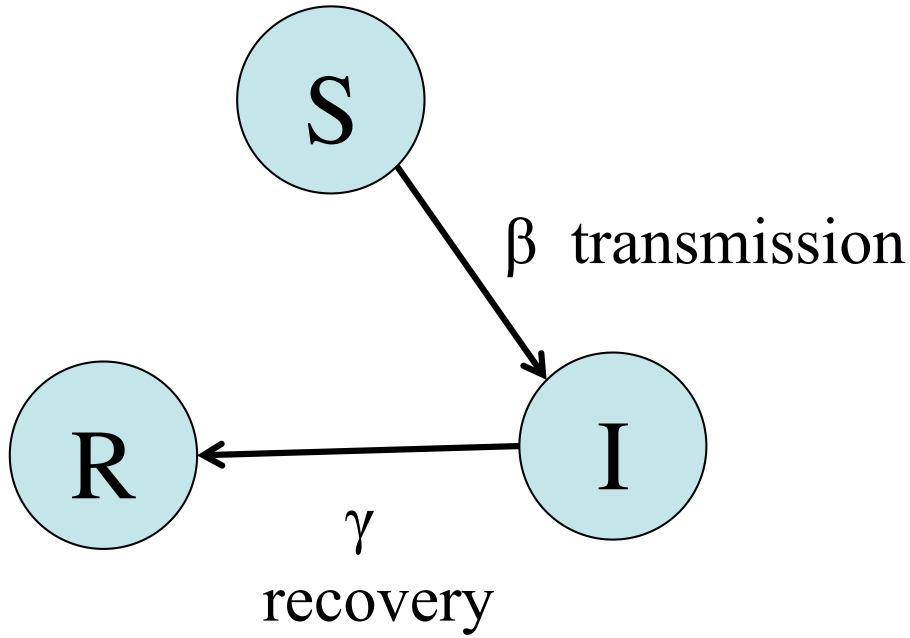
Kermack-McKendrick model

- Spread of diseases like the plague and cholera? A popular SIR model is the Kermack-McKendrick model.
- The model assumes:
  - A constant population size.
  - A zero incubation period.
  - The duration of infectivity is as long as the duration of the clinical disease.

Kermack-McKendrick model

- The Kermack-McKendrick model is specified as:

S: Susceptible persons
I: Infected persons
R: Removed (immune) persons
β: Infection rate
γ: Immunity rate



Kermack-McKendrick model

- The Kermack-McKendrick model is specified as:

S: Susceptible persons

I: Infected persons

R: Removed (immune) persons

β: Infection rate

γ: Immunity rate

$$\frac{dS}{dt} = -\beta\, I(t) S(t)$$

$$\frac{dI}{dt} = \beta\, I(t) S(t) - \gamma\, I(t)$$

$$\frac{dR}{dt} = \gamma\, I(t)$$

Kermack-McKendrick model

- The Kermack-McKendrick model is specified as:

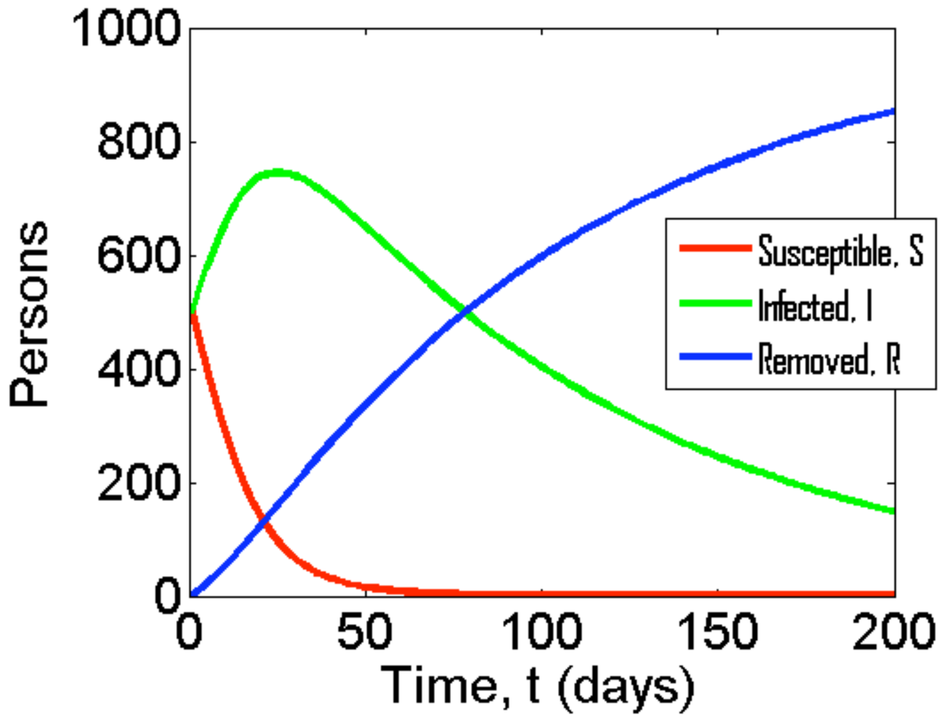
S: Susceptible persons

I: Infected persons

R: Removed (immune) persons

β: Infection rate

γ: Immunity rate



Exercise 1

- Implement and simulate the Kermack-McKendrick model in MATLAB.

  Use the starting values:
  S=I=500, R=0, β=0.0001, γ =0.01

Exercise 2

- A key parameter for the Kermack-McKendrick model is the epidemiological threshold, $\beta S/\gamma$.

  Plot the time evolution of the model and investigate the influence of the epidemiological threshold, in particular the cases:

1. $\beta S/\gamma < 1$
2. $\beta S/\gamma > 1$

Starting values: S=I=500, R=0, β=0.0001

Exercise 3 - optional

- Implement the Lotka-Volterra model and investigate the influence of the timestep, **`dt`**.
- How small must the timestep be in order for the 1st order Euler's method to give reasonable accuracy?
- Check in the MATLAB help how the functions **`ode23, ode45`** etc, can be used for solving differential equations.

References

- Matlab and Art: http://vlab.ethz.ch/ROM/DBGT/MATLAB_and_art.html
- Kermack, W.O. and McKendrick, A.G. "A Contribution to the Mathematical Theory of Epidemics." *Proc. Roy. Soc. Lond. A* **115**, 700-721, 1927.
- "Lotka-Volterra Equations". http://mathworld.wolfram.com/Lotka-VolterraEquations.html
- http://en.wikipedia.org/wiki/Numerical_ordinary_differential_equations
- "Lorenz Attractor". http://mathworld.wolfram.com/LorenzAttractor.html