# ODE to Joy

## Introduction to Ordinary Differential Equations

Jens Hahn

Humboldt-Universität zu Berlin
Group of Theoretical Biophysics

November 21st 2018

# This lecture

1. Why do we need ODEs?
2. Theoretical background
3. Systems Biology
4. Numerical solution
5. Solving ODEs in Python
6. Assignments

# Why do we need ODEs?

## The pros and cons for modelling

ODEs describe the change of something (dependent variable) in dependence of something else (independent variable)

### Why to use

- Many tools/methods
  - Setting up
  - Simulation
  - Analysis
  - Fitting
- Fast and cheap

### Why to avoid

- Many parameters
- Non-promiscuous
- Many kinetic rates
- Nothing is deterministic and continuous

# Ordinary Differential Equations

## Classification

General form (explicit):

$$\vec{y}^{\,(n)} = \vec{f}\left(t, \vec{y}, \vec{y}^{\,\prime}, \dots, \vec{y}^{\,(n-1)}\right)$$

$$\begin{pmatrix} \vec{y}_1^{\,(n)} \\ \vec{y}_2^{\,(n)} \\ \vdots \\ \vec{y}_m^{\,(n)} \end{pmatrix} = \begin{pmatrix} f_1\left(t, \vec{y}, \vec{y}^{\,\prime}, \dots, \vec{y}^{\,(n-1)}\right) \\ f_2\left(t, \vec{y}, \vec{y}^{\,\prime}, \dots, \vec{y}^{\,(n-1)}\right) \\ \vdots \\ f_m\left(t, \vec{y}, \vec{y}^{\,\prime}, \dots, \vec{y}^{\,(n-1)}\right) \end{pmatrix}$$

Every ODES can be turned into a system of $1^{\text{st}}$ order equations!

# Ordinary Differential Equations

## Classification

linear $y^{(n)} = \sum_{i=0}^{n-1} a_i(t) y^{(i)} + r(x)$

homogeneous $y^{(n)} = \sum_{i=0}^{n-1} a_i(t) y^{(i)}$

autonomous $\frac{\mathrm{d}\vec{y}}{\mathrm{d}t} = \vec{f}(\vec{y})$

# ODEs in Systems Biology

## Classification

$$\frac{\mathrm{d}\vec{y}}{\mathrm{d}t} = \vec{f}(\vec{y})$$

- $1^{\text{st}}$ order, homogeneous, autonomous, non-linear
- Initial value problems (IVP)
- NO negative values

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = \sum \mathrm{Rates}_{\mathrm{Production}} - \sum \mathrm{Rates}_{\mathrm{Consumption}}$$

# Chemical reaction

## ODE description

$$\text{Glucose} + \text{ATP} \xrightarrow{\nu_{\text{Hxk1}}} \text{Glucose-6P} + \text{ADP}$$

$$
\begin{aligned}
\frac{\mathrm{d}[Glc]}{\mathrm{d}t} &= -\nu_{\text{Hxk1}}([Glc],[ATP];p) \\
\frac{\mathrm{d}[G6P]}{\mathrm{d}t} &= +\nu_{\text{Hxk1}}([Glc],[ATP];p) \\
\frac{\mathrm{d}[ATP]}{\mathrm{d}t} &= -\nu_{\text{Hxk1}}([Glc],[ATP];p) \\
\frac{\mathrm{d}[ADP]}{\mathrm{d}t} &= +\nu_{\text{Hxk1}}([Glc],[ATP];p)
\end{aligned}
$$

# Chemical reaction

## Example

$$\mathrm{FBP} \xrightarrow{\nu_{\mathrm{Fba1}}} \mathrm{DHAP} + \mathrm{GAP}$$

# Chemical reaction

## Example

$$\mathrm{FBP} \xrightarrow{\nu_{\mathrm{Fba1}}} \mathrm{DHAP} + \mathrm{GAP}$$

$$\begin{aligned}
\frac{\mathrm{d}[FBP]}{\mathrm{d}t} &= -\nu_{\mathrm{Fba1}}([FBP]; p) \\
\frac{\mathrm{d}[DHAP]}{\mathrm{d}t} &= +\nu_{\mathrm{Fba1}}([FBP]; p) \\
\frac{\mathrm{d}[GAP]}{\mathrm{d}t} &= +\nu_{\mathrm{Fba1}}([FBP]; p)
\end{aligned}$$

# Kinetic rate laws

## Choose wisely

$A \rightarrow B$

$$\text{Mass action: } \nu_{\text{A-B}} = \text{k} \cdot [A]$$

$$\text{Michaelis Menten: } \nu_{\text{A-B}} = \frac{\text{v}_{\text{Max}} \cdot [A]}{\text{K}_{\text{M}} + [A]}$$

$$\text{Convenience: } \nu_{\text{A-B}} = \text{E} \cdot \frac{\text{kf} \cdot \frac{B}{\text{K}_{\text{M,B}}} - \text{kr} \cdot \frac{A}{\text{K}_{\text{M,A}}}}{(1 + \frac{B}{\text{K}_{\text{M,B}}}) + (1 + \frac{A}{\text{K}_{\text{M,A}}}) - 1}$$

# Background

## Get the algorithm

**Not all ODEs can be solved analytically, but we can exploit the first order description to define algorithms!!**

1. Read the $\frac{\mathrm{d}\vec{y}}{\mathrm{d}t}$ as $\frac{\Delta\vec{y}}{\Delta t} = \vec{f}(\vec{y}, t)$
2. Split it: $\Delta\vec{y} = \vec{f}(\vec{y}, t) \cdot \Delta t$
3. Discretise it: $\vec{y}_{i+1} - \vec{y}_i = \vec{f}(\vec{y}, t) \cdot \Delta t$
4. Separate: $\vec{y}_{i+1} = \vec{y}_i + \vec{f}(\vec{y}, t) \cdot \Delta t$

# Background

## Get the algorithm

**Not all ODEs can be solved analytically, but we can exploit the first order description to define algorithms!!**

1. Read the $\frac{\mathrm{d}\vec{y}}{\mathrm{d}t}$ as $\frac{\Delta\vec{y}}{\Delta t} = \vec{f}(\vec{y}, t)$
2. Split it: $\Delta\vec{y} = \vec{f}(\vec{y}, t) \cdot \Delta t$
3. Discretise it: $\vec{y}_{i+1} - \vec{y}_i = \vec{f}(\vec{y}, t) \cdot \Delta t$
4. Separate: $\vec{y}_{i+1} = \vec{y}_i + \vec{f}(\vec{y}, t) \cdot \Delta t$

# Background

## Get the algorithm

**Not all ODEs can be solved analytically, but we can exploit the first order description to define algorithms!!**

1. Read the $\frac{d\vec{y}}{dt}$ as $\frac{\Delta\vec{y}}{\Delta t} = \vec{f}(\vec{y}, t)$
2. Split it: $\Delta\vec{y} = \vec{f}(\vec{y}, t) \cdot \Delta t$
3. Discretise it: $\vec{y}_{i+1} - \vec{y}_i = \vec{f}(\vec{y}, t) \cdot \Delta t$
4. Separate: $\vec{y}_{i+1} = \vec{y}_i + \vec{f}(\vec{y}, t) \cdot \Delta t$

# Background

## Get the algorithm

**Not all ODEs can be solved analytically, but we can exploit the first order description to define algorithms!!**

1. Read the $\frac{d\vec{y}}{dt}$ as $\frac{\Delta\vec{y}}{\Delta t} = \vec{f}(\vec{y}, t)$
2. Split it: $\Delta\vec{y} = \vec{f}(\vec{y}, t) \cdot \Delta t$
3. Discretise it: $\vec{y}_{i+1} - \vec{y}_i = \vec{f}(\vec{y}, t) \cdot \Delta t$
4. Separate: $\vec{y}_{i+1} = \vec{y}_i + \vec{f}(\vec{y}, t) \cdot \Delta t$

# Background

## Get the algorithm

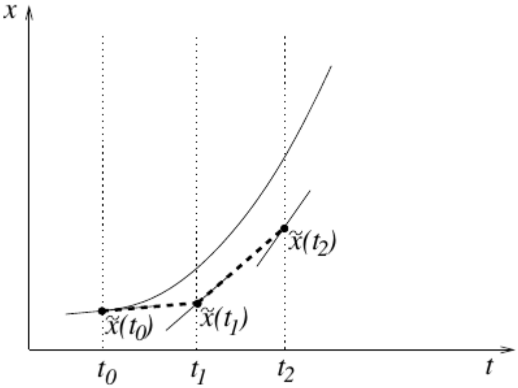
**Not all ODEs can be solved analytically, but we can exploit the first order description to define algorithms!!**

1. Read the $\frac{d\vec{y}}{dt}$ as $\frac{\Delta\vec{y}}{\Delta t} = \vec{f}(\vec{y}, t)$
2. Split it: $\Delta\vec{y} = \vec{f}(\vec{y}, t) \cdot \Delta t$
3. Discretise it: $\vec{y}_{i+1} - \vec{y}_i = \vec{f}(\vec{y}, t) \cdot \Delta t$
4. Separate: $\vec{y}_{i+1} = \vec{y}_i + \vec{f}(\vec{y}, t) \cdot \Delta t$

# Explicit Euler Method

## The simplest way

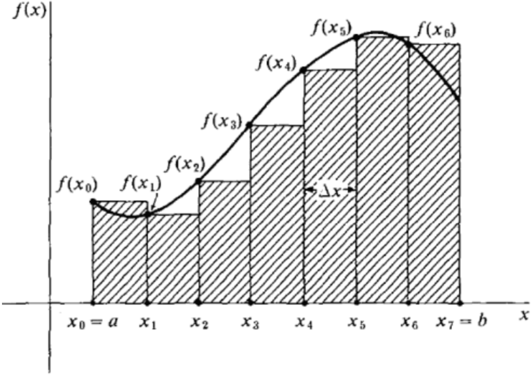
$$\widetilde{x}(t_{n+1}) = \widetilde{x}(t_n) + \mathrm{h} \cdot f(\widetilde{x}(t_n), t_n)$$




# Explicit Euler Method

## The simplest way

$$\widetilde{x}(t_{n+1}) = \widetilde{x}(t_n) + \mathrm{h} \cdot f(\widetilde{x}(t_n), t_n)$$

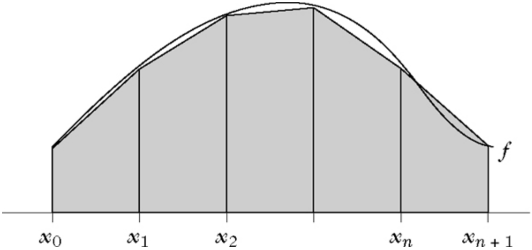


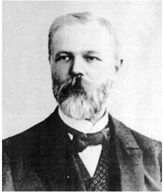

# Heun's Method

## The trapezoidal rule

Use trapezoidal rule to approximate the integral:

$$\widetilde{x}(t_{n+1}) = \widetilde{x}(t_n) + \frac{h}{2}\left( f(t_n, \widetilde{x}(t_n)) + f\left(t_n + h, \underbrace{\widetilde{x}(t_n) + h \cdot f(\widetilde{x}(t_n), t_n)}_{\text{Euler's method}}\right)\right)$$



# Adaptive Step Size

## Step Size Control

### Save time and computational cost

- The error decreases with the step size
- The computational cost increases with the step size
- Adapt the step size automatically

### Siff ODE Systems

- Some ODE systems contain very fast AND very slow components
- This can disturb the step size control
- Use implicit methods!

# State of the Art

## Overview

### Single-step methods

- Euler's method (implicit and explicit)
- Heun's method
- Dormand-Price (DOPRI) (explicit)
- Runge-Kutta method (implicit and explicit)

### Multi-step methods

- Adams-Bashforth method (explicit)
- Adams-Moulton (implicit)
- Backward Differentiation Formula (BDF) (implicit)

# What do we need?

## Numerical implementation of IVP

1. Initial values & parameters
2. Timegrid : time of simulation (start & end)
3. Function to calculate the derivatives (Equations)

# SciPy package

## ODEint

Load the solver from the scipy package:

```
from scipy.integrate import odeint
```

### odeint

- Uses packages lsoda written in FORTRAN
- Automated stiff system detection (Multi-step methods)
- Standard ODE solver in Python

# Implement solver

## Load parameters

```
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt

# load parameters
S0 = 100 # mM
P0 = 0 # mM
k = 1 # 1/min
# simulation time
start = 0 # min
end = 100 # min
plotpoints = 1000
timegrid = np.linspace(start, end, plotpoints)
```

# Implement solver

## Write derivative function

```
# function for derivatives
def f(y, t):
 S = y[0]
 P = y[1]
 dS = 0.5 - k * S
 dP = k * S
 return [dS, dP]
```

# Implement solver
## Start simulation

```
y0 = [S0,P0] # get initial vector
result = odeint(f, y0, timegrid)

# plot results
S_data = result[:, 0] # plot points substrate
P_data = result[:, 1] # plot points product
plt.plot(timegrid, S_data, label='substrate')
plt.plot(timegrid, P_data, label='product')
```

# Assignments

## Lotka-Volterra model

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - \beta xy$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \delta xy - \gamma y$$

$$\alpha = 1.5$$
$$\beta = 1$$
$$\delta = 3.$$
$$\gamma = 1$$

# Assignments

## Biochemical reaction model