MAS212 Scientific Computing and Simulation

Dr. Sam Dolan

School of Mathematics and Statistics,
University of Sheffield

Autumn 2019

http://sam-dolan.staff.shef.ac.uk/mas212/

G18 Hicks Building
s.dolan@sheffield.ac.uk

Today's lecture

- Scientific computing modules:
  - `numpy`
  - `matplotlib`
  - `scipy`
- **Differential equations:**

  Phase portraits; equilibria; limit cycles.
- **Non-linear ODEs**: 3 examples:
  1. Logistic equation (1D)
  2. Predator-prey equation (2D autonomous conservative)
  3. van der Pol equation (2nd-order)

SciPy

What is SciPy?

SciPy is a collection of mathematical algorithms and functions built on the Numpy extension of Python.

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import scipy as sp
```

- Tutorial: `http://docs.scipy.org/doc/scipy-dev/reference/tutorial/index.html`

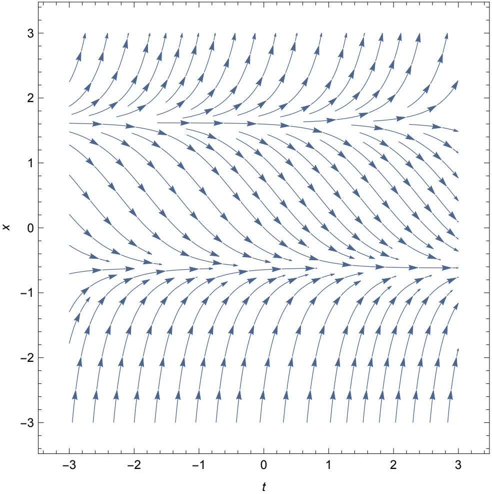
SciPy

- Various useful modules in the `scipy` package:
  - `sp.special` : special functions (Bessel, Legendre, Hypergeometric, etc).
  - `sp.integrate` : for integrating functions and sets of ODEs
  - `sp.optimize` : curve fitting, minimization, etc.
  - `sp.interpolate` : interpolation, splines, etc.
  - `sp.fftpack` : Fourier transforms.
  - `sp.linalg` : Linear algebra.
- We will solve differential equations with `scipy.integrate.odeint`

Ordinary differential equations

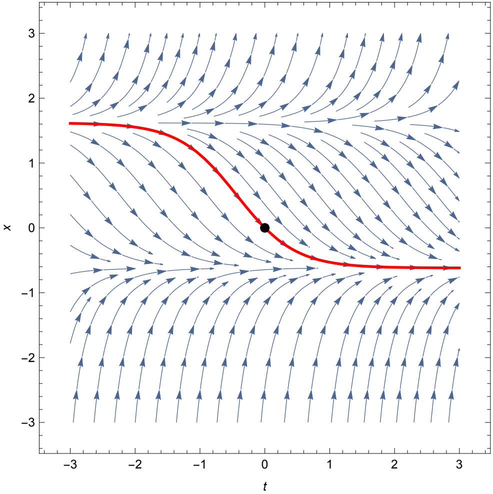
- Here is an example of an **ordinary differential equation** (ODE):
$$\frac{dx}{dt} = x^2 - x - 1.$$
- x is the dependent variable, and t is the **ind**ependent variable.
- a specific solution $x(t)$ is an **integral curve** of the ODE.
- to find an integral curve, we specify an **initial condition**, e.g.
$$x(t = 0) = 1$$

Ordinary differential equations



- Here is the gradient field $\frac{dx}{dt}$ at each point in the flow.

Ordinary differential equations



- Here is an **integral curve** with initial condition $x(0) = 0$.

Ordinary differential equations (ODEs)

- ODEs have **one** independent variable, t say
- There may be several dependent variables $x_i = \{x_1(t), x_2(t), \ldots\}$,
- and a set of functions F_j relating x_i and its derivatives,

$$F_j(x_i, \dot{x}_i, \ddot{x}_i, \ldots; t) = 0$$

where $\dot{x}_i = \frac{dx_i}{dt}$, $\ddot{x}_i = \frac{d^2 x_i}{dt^2}$, etc.

Ordinary differential equations (ODEs)

- ODEs have **one** independent variable, t say
- There may be several dependent variables $x_i = \{x_1(t), x_2(t), \ldots\}$,
- and a set of functions F_j relating x_i and its derivatives,

$$F_j(x_i, \dot{x}_i, \ddot{x}_i, \ldots; t) = 0$$

where $\dot{x}_i = \frac{dx_i}{dt}$, $\ddot{x}_i = \frac{d^2 x_i}{dt^2}$, etc.

- **Order** refers to highest derivative: kth order $\Leftrightarrow \frac{d^k x}{dt^k}$
- **Dimension** refers number of dependent variables $\mathbf{x} = [x_1 \ldots x_d]$, and the number of independent equations.
- **Autonomous** $\Leftrightarrow$ F_j have no explicit dependence on t
- **Linear** if F_j has only linear dependence on x_i, $\dot{x}_i$, $\ldots$ and their combinations. Otherwise it is **non-linear**.
- **Linear** $\Rightarrow$ superposition principle $\Rightarrow$ 'Easy'.

Ordinary differential equations

$$\frac{dx}{dt} = x^2 - x - 1$$

This example is ...

- ... **first-order**, as dx/dt is the highest derivative.
- ... **one-dimensional**, as x is the only dependent variable.
- ... **autonomous**, as the rate of change dx/dt does not depend on the independent variable t.
- ... **non-linear**, because of the non-linear term x^2 on the right-hand side.

1D autonomous equation

- Consider the 1st-order autonomous case:

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

- A solution is typically found by **separation of variables**
- Divide by $f(x)$ and integrate

$$\int \frac{dx}{f(x)} = t + c$$

- Some cases can be solved exactly, e.g,

$$f(x) = x \quad \Rightarrow \ln(x) = t + c \quad \Rightarrow x(t) = Ae^{t}$$

- What if integral can't be found analytically?
- Integrate numerically and invert to find $x(t)$? **No.**
- Numerically solve the differential equation with `odeint()`.

1D autonomous equation: example

- The **Logistic Equation** is a 1st order autonomous ODE:

$$\boxed{\frac{dx}{dt} = x(1-x), \quad x(0) = x_0}$$

- It has the exact solution (**show**):

$$x(t) = \frac{1}{1 + Ae^{-t}}.$$

(Here $A = 1/x_0 - 1$)

1D autonomous equation: example

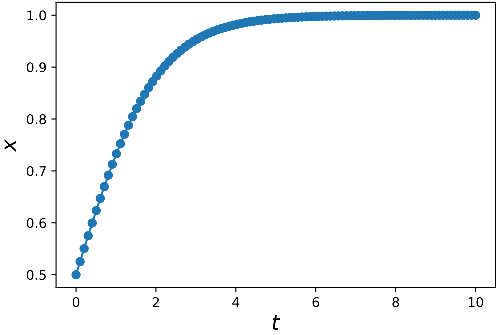
$$\boxed{\frac{dx}{dt} = x(1-x), \quad x(0) = x_0}$$

```
import matplotlib.pyplot as plt
from scipy.integrate import odeint
def logistic(x, t):
    """Returns the gradient dx/dt for the logistic equation"""
    return x*(1 - x)

ts = np.linspace(0.0, 10.0, 100) # values of independent variable
x0 = 0.5   # an initial condition, x(0) = x0
xs = odeint(logistic, x0, ts)
#  'odeint' returns an array of 'x' values, at the times in ts.
plt.xlabel('$t$', fontsize=16); plt.ylabel('$x$', fontsize=16)
plt.plot(ts, xs)
```

1D autonomous equation: example

$$\boxed{\frac{dx}{dt} = x(1-x), \quad x(0) = x_0}$$



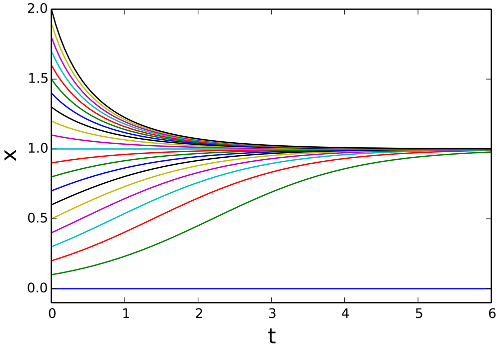
- Here $x_0 = 0.5$. Not very illuminating. ...
- Let's plot curves for several initial conditions ...

1D autonomous equation: example

$$\boxed{\frac{dx}{dt} = x(1-x), \quad x(0) = x_0}$$

```python
# Plot curves for several initial conditions
ics = np.linspace(0.0, 2.0, 21)  # a list of initial conditions
for x0 in ics:
    xs = odeint(logistic, x0, ts)
    plt.plot(ts, xs)
```

1D autonomous equation: example



- Two equilibrium positions: $x = 0$ and $x = 1$.
- $x = 0$ is an **unstable** equilibrium.
- $x = 1$ is a **stable** equilibrium.

2D autonomous equations

- Now consider a first order system with two dependent variables, x and y,

$$
\begin{aligned}
\frac{dx}{dt} &= f(x, y; t), \\
\frac{dy}{dt} &= g(x, y; t).
\end{aligned}
$$

- System is **autonomous** iff f and g do not depend on t.
- **Example:** Modelling the populations of rabbits and foxes.

2D autonomous equations: example

Predator-prey equations

Also known as *Lotka-Volterra equations*, the predator-prey equations are a pair of coupled first-order non-linear ordinary differential equations.

They represent a simplified model of the change in populations of two species which interact via predation. For example, foxes (predators) and rabbits (prey). Let x and y represent rabbit and fox populations, respectively. Then

$$\frac{dx}{dt} = ax - bxy$$
$$\frac{dy}{dt} = -cy + dxy$$

Here a, b, c and d are parameters, which are assumed to be positive.

Predator-prey equations

$$\frac{dx}{dt} = ax - bxy$$
$$\frac{dy}{dt} = -cy + dxy$$

```
def dZ_dt(Z, t, a=1, b=1, c=1, d=1): # a,b,c,d optional arguments.
    x, y = Z[0], Z[1]
    dxdt, dydt = x*(a - b*y), -y*(c - d*x)
    return [dxdt, dydt]

ts = np.linspace(0, 12, 100)
Z0 = [1.5, 1.0]    # initial conditions for x and y
Zs = odeint(dZ_dt, Z0, ts, args=(1,1,1,1))
    # use optional argument 'args' to pass parameters to dZ_dt
prey = Zs[:,0] # first column
predators = Zs[:,1]  # second column
```

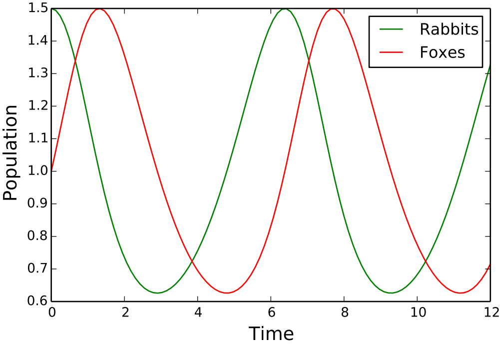
Predator-prey equations

$$\frac{dx}{dt} = ax - bxy$$
$$\frac{dy}{dt} = -cy + dxy$$

```
# Let's plot 'rabbit' and 'fox' populations as a function of time
plt.plot(ts, prey, "+", label="Rabbits")
plt.plot(ts, predators, "x", label="Foxes")
plt.xlabel("Time", fontsize=14)
plt.ylabel("Population", fontsize=14)
plt.legend();
```

Predator-prey equations

$$\frac{dx}{dt} = ax - bxy$$
$$\frac{dy}{dt} = -cy + dxy$$

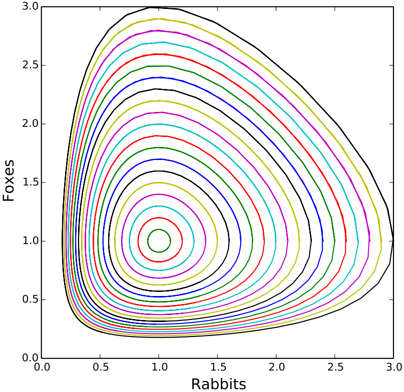


Predator-prey equations: Phase plot

- The ODEs are autonomous: no explicit dependence on t
- **Phase portrait**: Plot x vs y (instead of x, y vs t).
- One curve for each initial condition
- Curves will not cross (typically) for an autonomous system.

```
fig = plt.figure()
fig.set_size_inches(6,6)  # Square plot, 1:1 aspect ratio
ics = np.arange(1.0, 3.0, 0.1) # initial conditions
for r in ics:
    Z0 = [r, 1.0]
    Zs = odeint(dZ_dt, Z0, ts)
    plt.plot(Zs[:,0], Zs[:,1], "-")
plt.xlabel("Rabbits", fontsize=14)
plt.ylabel("Foxes", fontsize=14)
```

Predator-prey equations: Phase plot



- Curves do not cross
- Closed curves ⇔ **Periodic** solutions
- Equilibrium at $x = y = 1 \quad \Rightarrow \quad \dot{x} = \dot{y} = 0$

The Van der Pol oscillator

The (undriven) Van der Pol oscillator is a non-conservative oscillator with non-linear damping, satisfying

$$\boxed{\ddot{x} - a(1 - x^2)\dot{x} + x = 0}$$

- This is a **second-order** ODE with one parameter, *a*
- $|x| > 1$: loses energy
- $|x| < 1$: absorbs energy

- Originally, used as a model for an electric circuit with a vacuum tube.
- Used to model biological processes such as heart beat, circadian rhythms, biochemical oscillators, and pacemaker neurons.

Van der Pol oscillator

$$\boxed{\ddot{x} - a(1 - x^2)\dot{x} + x = 0}$$

First-order reduction:

Any second-order equation can be written as two coupled first-order equations, by introducing a new variable.

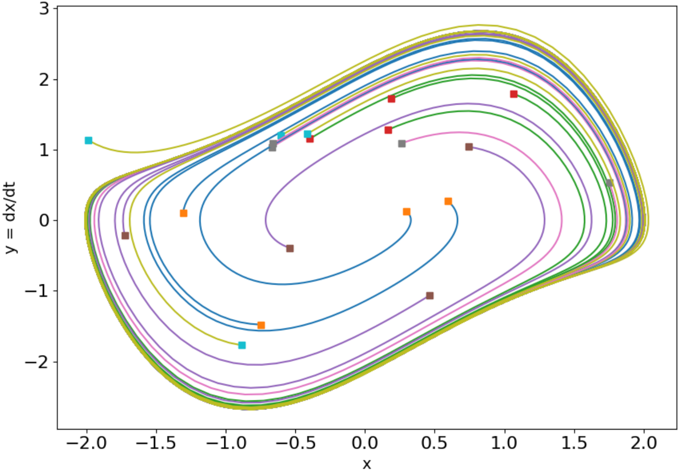
- Let $y = \frac{dx}{dt}$. Then

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= a(1 - x^2)y - x \end{aligned}$$

- (Not unique: we could make another choice, such as $z = \dot{x} + x$.)

$$
\begin{aligned}
\dot{x} &= y \\
\dot{y} &= a(1 - x^2)y - x
\end{aligned}
$$

```
def dZ_dt(Z, t, a = 1.0):
    x, y = Z[0], Z[1]
    dxdt = y
    dydt = a*(1-x**2)*y - x
    return [dxdt, dydt]
def random_ic(scalefac=2.0):  # stochastic initial condition
    return scalefac*(2.0*np.random.rand(2) - 1.0)
ts = np.linspace(0.0, 40.0, 400)
nlines = 20
for ic in [random_ic() for i in range(nlines)]:
    Zs = odeint(dZ_dt, ic, ts, args=(1.0))
    plt.plot(Zs[:,0], Zs[:,1])
    plt.plot([Zs[0,0]],[Zs[0,1]], 's') # plot the first point
```



- All curves tend towards a **limit cycle**

Van der Pol oscillator: Limit cycles

- Investigate how the limit cycle varies with the parameter *a*:

```
avals = np.arange(0.2, 2.0, 0.2)  # parameters
minpt = int(len(ts) / 2)  # look at late-time behaviour
for a in avals:
    Zs = odeint(dZ_dt, random_ic(), ts, args=(a,))
    plt.plot(Zs[minpt:,0], Zs[minpt:,1])
```

Van der Pol oscillator: Limit cycles