Week 5, Lecture 9 - Dynamical models

Aaron Meyer

Outline

- Administrative Issues
- Review ODE models
  - Some common constructions
- Classic analysis:
  - Stability analysis
  - Pseudo-steady-state
- Implementation:
  - Numerical integration
  - Stiff systems
  - Matrix exponentials

Slides partly adapted from those by Bruce Tidor.

Ordinary Differential Equations

- ODE models are typically most useful when we already have an idea of the system components
  - As opposed to *data-driven* approaches when we don't know how to connect the data
  - Incredibly powerful for making specific predictions about how a system works
- Limits of these approaches:
  - Results can be extremely sensitive to missing components or model errors
  - Can quickly explode in complexity
  - May rely on variables that are impossible to measure

Applications of ODE models: Molecular kinetics

Remember BE100!

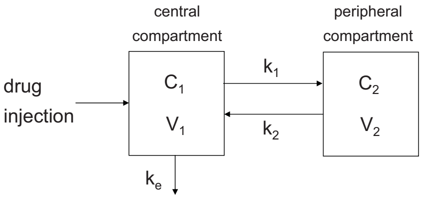
Let's say we have to ligands that dimerize, then this dimer binds to a receptor as one unit:

$$L_f + L_f \leftrightarrow L_D$$

$$L_D + R_f \leftrightarrow R_b$$

If we want to know about how these species interact, we can model their behavior with the rate equations that describe this process.

Applications of ODE models: Pharmacokinetics



Applications of ODE models: Pharmacokinetics

- Central compartment corresponds to the plasma in the body.
  - V_1 is the distribution volume of plasma in the body.
  - C_1 is the concentration of drug in the plasma.
- Peripheral compartment represents a group of organs that significantly take up the particular drug.
  - V_2 is the volume of these group of organs.
  - C_2 is the concentration of drug in the group of organs.

Applications of ODE models: Pharmacokinetics

- k_e is the rate constant for clearance.
  - $k_e C_1 V_1$ is the mass of drug/time that's cleared.
- k_1 is the rate constant for mass transfer from the central to peripheral compartment.
  - $k_1 C_1 V_1$ is the mass of drug/time that transfers from the central to peripheral compartment.
- k_2 is the rate constant for mass transfer from the peripheral to central compartment.
  - $k_2 C_2 V_2$ is the mass of drug/time that transfers from the peripheral to the central compartment.

Applications of ODE models: Pharmacokinetics

- Have a bolus i.v. injection
  - No drug in both compartments for t<0
  - D μg of drug administered at once at t=0
  - Drug distribution occurs instantaneously in the central compartment.
    - Also get well-mixed instantaneously.
    - Concentration in central compartment at $t = 0$ is D μg/mL
- No chemical reactions in the compartment

Applications of ODE models: Population kinetics

Lotka-Volterra Equations

Note about difference from other models we've covered

- ODE models can be part of inference techniques just as elsewhere
  - If we have a symbolic integral, then fitting an ODE model to data is just non-linear least squares
- But we often don't have a symbolic expression of the answer
  - Have to simulate the model every time
  - Can only focus on the input-output we get from the black box
- In this respect, what we do with ODE models will be very similar to what you could do with any computational simulation

Analytic vs Numerical Modeling

- Analytic
  - Wider range of parameters
  - Avoid numerical problems
  - Physical intuition more direct
  - Often must simplify model
- Numerical
  - Can handle complex models
  - Dependence on parameters & initial conditions
  - Physical insight may be difficult to extract
  - Convergence, numerical stability

Reality often requires handling in between:

- Use analytic treatment to study entire parameter space
- Use numerical treatment to study interesting regions
- Use both to handle complex behavior

Stability Analysis

- Can solve for steady-states of a system

$$\frac{\delta F}{\delta t} = 0$$

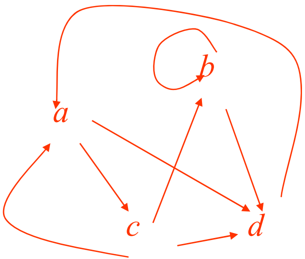
- Results of this can be both stable or unstable points
  - With stable points, slope of $\frac{\delta F}{\delta t}$ is negative
  - In multivariate case, this means eigenvalues of Jacobian are negative
- Steady-state points aren't necessarily realistic or feasible!
  - NNLSQ can solve for points
  - Only simulating system ensures they are accessible

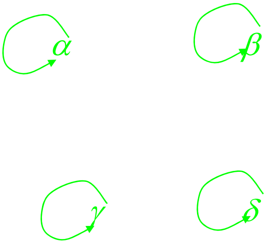
Generalization

- Linear models are easier to simulate and understand than non-linear
  - Linearity: If $\mathbf{x}_1$ and $\mathbf{x}_2$ are both solutions, then $c_1\mathbf{x}_1 + c_2\mathbf{x}_2$ is also a solution
- Linear systems tend to be separable (effective decoupling)
- Non-linear systems exhibit interesting properties

Linearity & Separability

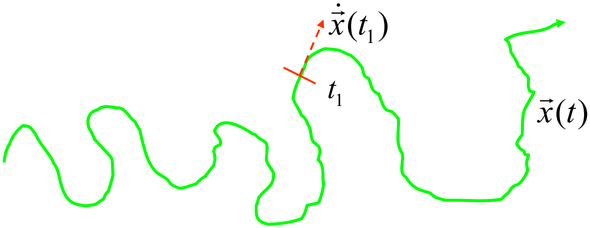
$$\begin{pmatrix} a' \\ b' \\ c' \\ d' \end{pmatrix} = \begin{pmatrix} \kappa_{11} & \kappa_{12} & \kappa_{13} & \kappa_{14} \\ \kappa_{21} & \kappa_{22} & \kappa_{23} & \kappa_{24} \\ \kappa_{31} & \kappa_{32} & \kappa_{33} & \kappa_{34} \\ \kappa_{41} & \kappa_{42} & \kappa_{43} & \kappa_{44} \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \qquad \begin{pmatrix} \alpha' \\ \beta' \\ \gamma' \\ \delta' \end{pmatrix} = \begin{pmatrix} \lambda_{11} & 0 & 0 & 0 \\ 0 & \lambda_{22} & 0 & 0 \\ 0 & 0 & \lambda_{33} & 0 \\ 0 & 0 & 0 & \lambda_{44} \end{pmatrix} \cdot \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix}$$





Phase Portraits

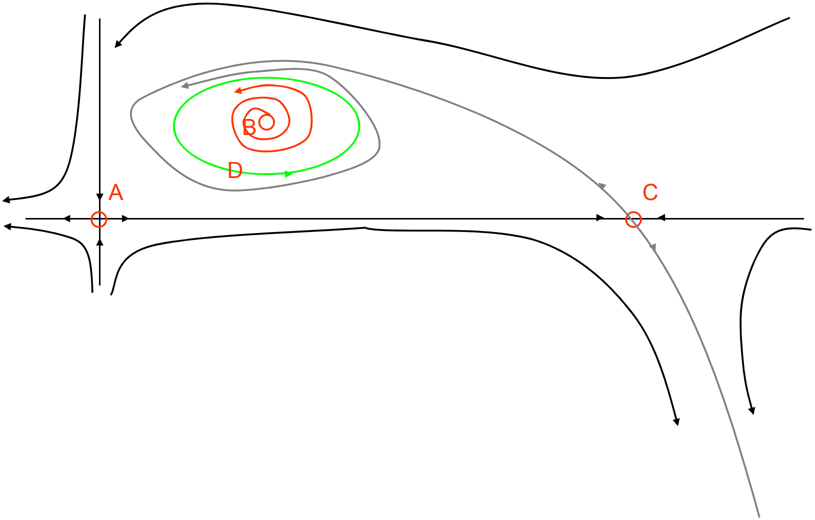
$$\left.\begin{aligned}\dot{x}_1 &= f_1(x_1, x_2)\\ \dot{x}_2 &= f_2(x_1, x_2)\end{aligned}\right\} \longrightarrow \dot{\vec{x}} = \vec{f}(\vec{x})$$



Non-linear systems

- No general analytic approach to finding trajectory
- So, goal is to understand qualitative trajectory behavior

Features in Phase Portraits



Solving a Set of Equations for Phase Portrait

- Numerical computation
  - i.e., Runge-Kutta integration
- Qualitative
  - Sufficient for some purposes
- Analytic
  - Elegant, though not always tractable

Example – fixed points

$$\dot{x} = x + e^{-y}$$
$$\dot{y} = -y$$

<u>Nonlinear</u>, because can't be represented as

$$\dot{x} = ax + by$$
$$\dot{y} = cx + dy$$

Step 1: Find fixed points

Fixed points (also called stationary points) are those points where the time-derivative of each coordinate is zero. $\dot{x} = 0$ and $\dot{y} = 0$

$$0 = x + e^{-y} \quad \Rightarrow \quad -x = e^{-y} = 1$$
$$0 = -y \quad \Rightarrow \quad y = 0$$

Thus, one fixed point at $(x, y) = (-1,0)$

Example – stability

Step 2: Determine stability of fixed points

- If the systems moves slightly away from each fixed point, will it return or will it move further away?
- Another way to ask the same question is to ask whether, as time approaches infinity, does the system tend toward or away from a given stable point.
- Note y solution must be of form:
  - $y = y_0 e^{-t}$ (because $\dot{y} = \frac{dy}{dt} = -y$)
  - So $y \to 0$ for $t \to \infty$
- Thus, $\dot{x} = x + e^{-y}$ becomes $\dot{x} \to x + 1$ for long times
  - This has exponentially growing solutions
  - Toward ∞ for $x > -1$ and $-\infty$ for $x < -1$

Thus, overall solution grows exponentially in at least one dimension, and so is unstable.

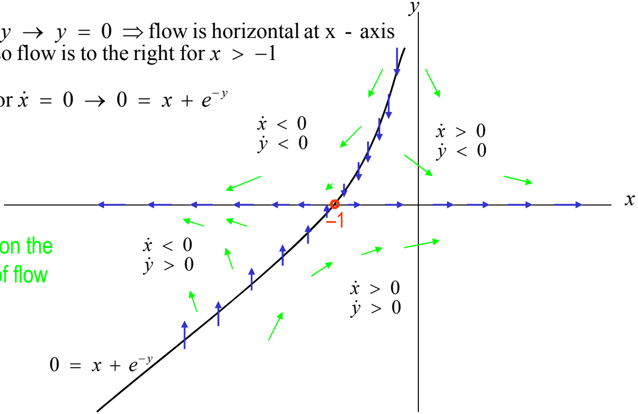
Example – nullclines

Step 3: Sketch nullclines

Nullclines are the sets of points for which $\dot{x} = 0$ or $\dot{y} = 0$, so flow is either horizontal or vertical.

$\dot{y} = 0$ for $0 = -y \rightarrow y = 0 \Rightarrow$ flow is horizontal at x - axis
$\dot{x} = x + 1$ here, so flow is to the right for $x > -1$

Flow is vertical for $\dot{x} = 0 \rightarrow 0 = x + e^{-y}$

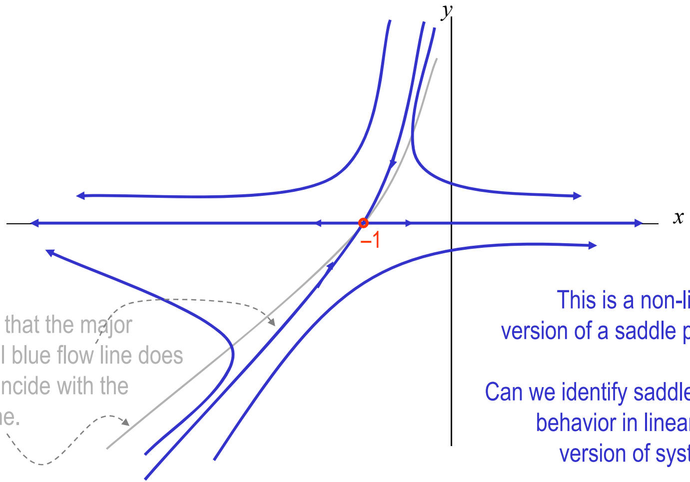


The nullclines partition the space into classes of flow direction:

$$\dot{x}, \dot{y} \left\{ \begin{matrix} < \\ > \end{matrix} \right\} 0$$

Example – computed

Step 4: Plot flow lines



Notice that the major vertical blue flow line does not coincide with the nullcline.

This is a non-linear version of a saddle point.

Can we identify saddle-like behavior in linearized version of system?

Existence & Uniqueness

Non-linear $\dot{\mathbf{x}} = f(\mathbf{x})$ and given an initial condition.

- Existence and uniqueness of solution guaranteed if f is continuously differentiable
- Corollary: Trajectories **do not** intersect, because if they did, then there would be two solutions for the same initial condition at the crossing point

Linearization About Fixed Points

Let $\begin{Bmatrix} \dot{x} = f(x, y) \\ \dot{y} = g(x, y) \end{Bmatrix}$ be a non - linear system with fixed point (x^*, y^*)

$$0 = f(x^*, y^*) = g(x^*, y^*)$$

Let $\begin{Bmatrix} u = x - x^* \\ v = y - y^* \end{Bmatrix}$ be deviations from fixed point

$\downarrow$ Change of variable

linear

$$\begin{aligned} \dot{u} &= \dot{x} \quad (x^* \text{ is constant}) \\ &= f(u + x^*, v + y^*) \\ &= \cancelto{0}{f(x^*, y^*)} + u\frac{\partial f}{\partial x} + v\frac{\partial f}{\partial y} + \cancelto{0}{O(u^2, v^2, uv)} \quad \text{Taylor series expansion} \end{aligned}$$

$$\text{Likewise, } \dot{v} = u\frac{\partial g}{\partial x} + v\frac{\partial g}{\partial y} + O(u^2, v^2, uv)$$

Solving Linearized Systems

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{pmatrix} \cdot \begin{pmatrix} u \\ v \end{pmatrix}$$

$$\dot{\vec{x}} = \overleftrightarrow{A}\vec{x} \qquad \overleftrightarrow{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad \text{let } \vec{x}(t) = e^{\lambda t}\vec{v}$$

$$\lambda e^{\lambda t}\vec{v} = \overleftrightarrow{A} e^{\lambda t}\vec{v} \qquad \lambda\vec{v} = \overleftrightarrow{A}\vec{v}$$

$$(\overleftrightarrow{A} - \lambda I)\vec{v} = 0$$

$$\begin{pmatrix} a-\lambda & b \\ c & d-\lambda \end{pmatrix}\vec{v} = 0$$

$$\det\begin{pmatrix} a-\lambda & b \\ c & d-\lambda \end{pmatrix} = 0$$

$$(a-\lambda)(d-\lambda) - bc = 0$$

$$\lambda^2 - \tau\lambda + \Delta = 0$$

$$\lambda_1 = \frac{\tau + \sqrt{\tau^2 - 4\Delta}}{2} \qquad \lambda_2 = \frac{\tau - \sqrt{\tau^2 - 4\Delta}}{2}$$

τ=trace

Δ=determinant

If $\lambda_1 \neq \lambda_2$, then v_1 & v_2 are linearly independent and solutions of the following form are valid.

$$\vec{x}(t) = c_1 e^{\lambda_1 t}\vec{v}_1 + c_2 e^{\lambda_2 t}\vec{v}_2$$

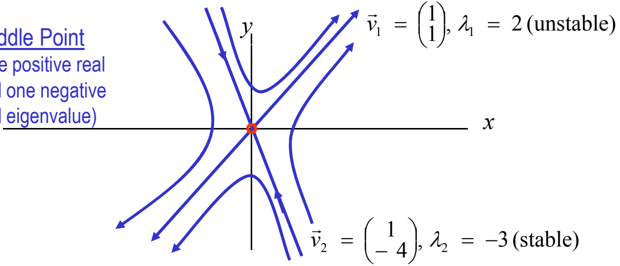
Example

$$\left.\begin{array}{l}\dot{x} = x + y \\ \dot{y} = 4x - 2y \\ (x,y)_{t=0} = (2,-3)\end{array}\right\} \begin{array}{c}\tau = -1 \\ \rightarrow \\ \Delta = -6\end{array} \left\{\begin{array}{l}\lambda_1 = 2 \\ \lambda_2 = -3\end{array}\right. \qquad \begin{array}{l}\vec{v}_1 = \begin{pmatrix}1\\1\end{pmatrix} \\ \vec{v}_2 = \begin{pmatrix}1\\-4\end{pmatrix}\end{array}$$

$$\vec{x}(t) = c_1\begin{pmatrix}1\\1\end{pmatrix}e^{2t} + c_2\begin{pmatrix}1\\-4\end{pmatrix}e^{-3t} \qquad \text{with } c_1 = c_2 = 1 \text{ from init. cond.}$$

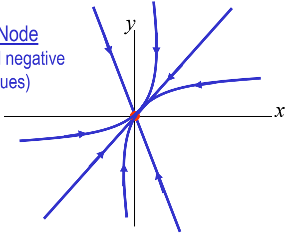
Can draw phase portrait directly from eigenvalues & eigenvectors:

Saddle Point
(one positive real
and one negative
real eigenvalue)

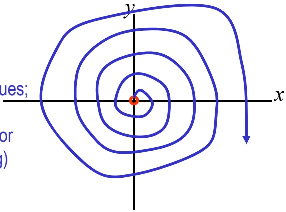


More Examples

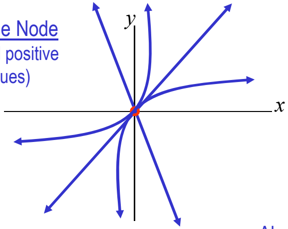
Stable Node
(two real negative eigenvalues)



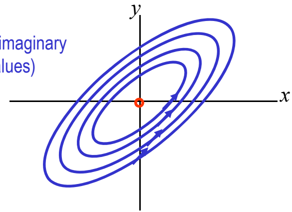
Spiral
(two complex eigenvalues; Can be growing or shrinking)



Unstable Node
(two real positive eigenvalues)

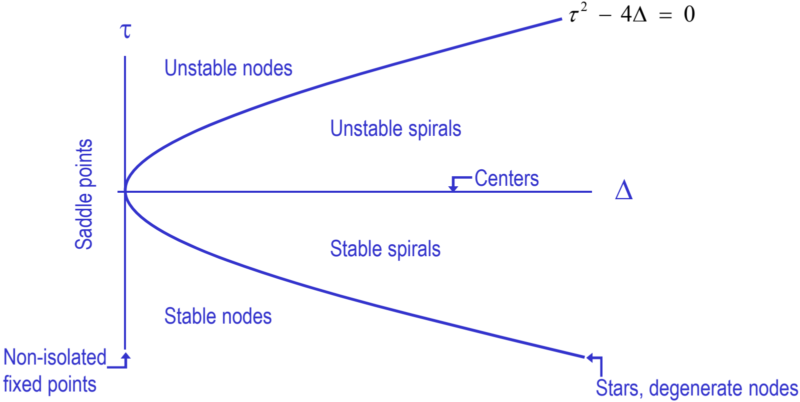


Center
(purely imaginary eigenvalues)



Also, equal eigenvalues lead to stars & degenerate nodes

Classification of Fixed Points



Relevance for Nonlinear Dynamics

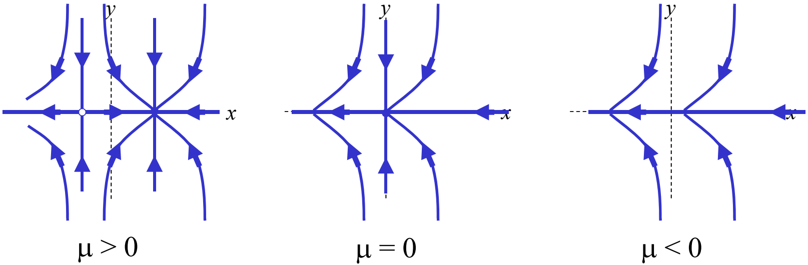
- So, we have said that we can find fixed points of nonlinear dynamics, linearize about each fixed point, and characterize the dynamics about each fixed point in the non-linear model by the corresponding linear model.
- Is this always true? Do the nonlinearities ever disturb this approach?
- A theorem can be proven which states
  - That all the regions on the previous slide are "robust" (nodes, spirals, saddles) and correspond between linear and nonlinear models.
  - But that all the lines on the previous slide are "delicate" (centers, stars, degenerate nodes, non-isolated fixed points) and can have different behaviors in linear and non-linear models.

Bifurcations

- The phase portraits we have been looking at describe the trajectory of the system for a given set of initial conditions. However, for "fixed" parameters (rate constants in eqns, for instance).
- What we might like is a series of phase portraits corresponding to different sets of parameters.
- **Many** will be qualitatively similar. The interesting ones will be where a small change of parameters creates a qualitative change in the phase portrait (bifurcations).
- What we will find is that fixed points & closed orbits can be created/destroyed and stabilized/destabilized.

Saddle-Node Bifurcation

$$\dot{x} = \mu - x^2$$
$$\dot{y} = -y$$



Genetic Control Network

Griffith (1971) model of genetic control:

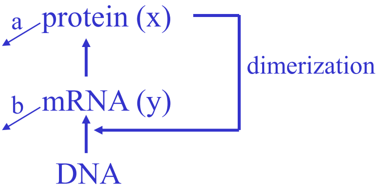
- x = protein concentration
- y = mRNA concentration

$$\dot{x} = -ax + y$$

protein degrades and is synthesized from mRNA

$$\dot{y} = \frac{x^2}{1+x^2} - by$$

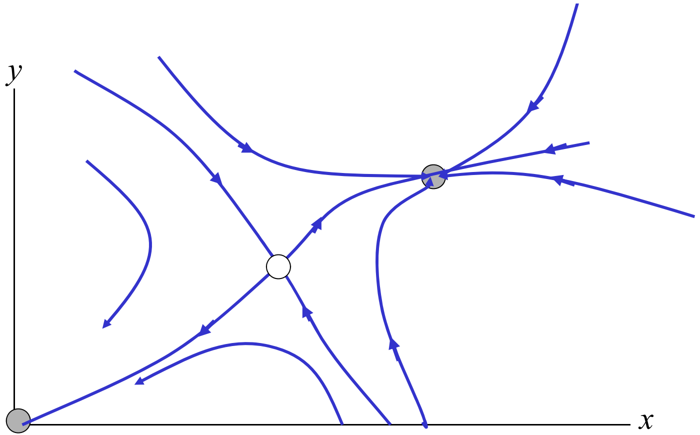
mRNA degrades and is stimulated by protein dimer



Genetic Control Network

Biochemical version of a bistable switch:

1. Only stable points are no protein and mRNA or a fixed composition
2. If degradation rates too great, only stable point is origin



Implementation - Testing

- Many properties one can test
  - Mass balance
  - Changes upon parameter adjustment
- Good to test these before and after integration

Implementation

SciPy provides two interfaces for ODE solving:

- scipy.integrate.ode
- scipy.integrate.odeint

Notes:

- Both can solve stiff and non-stiff equations.
- ode has a number of different methods. Pay attention to the "set_integrator" option.

Implementation - Example

The second order differential equation for the angle θ of a pendulum acted on by gravity with friction can be written:

$$\theta''(t) + b * \theta'(t) + c * \sin(\theta(t)) = 0$$

where b and c are positive constants, and a prime (') denotes a derivative. To solve this equation with `odeint`, we must first convert it to a system of first order equations. By defining the angular velocity $\omega(t) = \theta'(t)$, we obtain the system:

$$\theta'(t) = \omega(t)$$

$$\omega'(t) = -b * \omega(t) - c * \sin(\theta(t))$$

Implementation - Example

Let y be the vector $[\theta, \omega]$. We implement this system in python as:

```
def pend(y, t, b, c):
    theta, omega = y
    dydt = [omega, -b*omega - c*np.sin(theta)]
    return dydt
```

We assume the constants are $b = 0.25$ and $c = 5.0$:

```
b, c = 0.25, 5.0
```

Implementation - Example

For initial conditions, we assume the pendulum is nearly vertical with $\theta(0) = \pi - 0.1$, and it initially at rest, so $\omega(0) = 0$. Then the vector of initial conditions is

```
y0 = [np.pi - 0.1, 0.0]
```

We generate a solution 101 evenly spaced samples in the interval $0 \leq t \leq 10$. So our array of times is:

```
t = np.linspace(0, 10, 101)
```

Implementation - Example

Call odeint to generate the solution. To pass the parameters b and c to `pend`, we give them to `odeint` using the `args` argument.

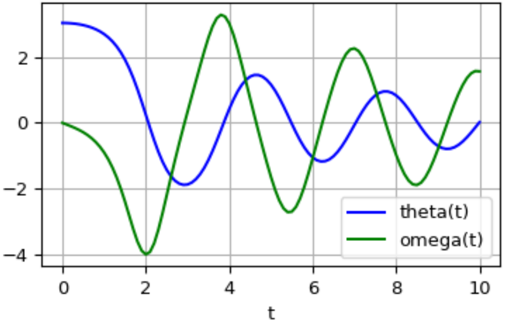
```
from scipy.integrate import odeint
sol = odeint(pend, y0, t, args=(b, c))
```

The solution is an array with shape (101, 2). The first column is $\theta(t)$, and the second is $\omega(t)$. The following code plots both components.

Implementation - Example

```python
import matplotlib.pyplot as plt
plt.plot(t, sol[:, 0], 'b', label='theta(t)')
plt.plot(t, sol[:, 1], 'g', label='omega(t)')
plt.legend(loc='best')
plt.xlabel('t')
plt.grid()
plt.show()
```

Implementation - Example



Implementation - Stiff Systems

- Very roughly, most ODE solvers take steps inversely proportional to the rate at which the state is changing
- For systems where there are two processes operating on differing timescales, this can be problematic
  - If everything happens really fast, the system will come to equilibrium quickly
  - If everything is slow, you can take longer steps
- *Stiff* solvers additionally require the Jacobian matrix
  - This very roughly allows them to keep track of these differences in timescales
- `odeint` can automatically find this for you
  - Sometimes it's faster/better to provide this as parameter `Dfun`

Implementation - Matrix Exponential

If J is the Jacobian matrix of an ODE model, $y(t) = e^{Jt} y_0$.

Matrix exponential is also implemented.

- scipy.linalg.expm
  - This method is numerically stable, but there are faster implementations elsewhere.
- A commonly used package is expokit

For linear systems, this can be $>$1000x faster.

Further Reading

- scipy.linalg.expm
- scipy.integrate.odeint
- Steven Strogatz, Nonlinear Dynamics and Chaos