Mathematics for Neuroscience

Barry Dillon

August 26, 2024

ISRC-CN3 Summer School
Ulster University

About this lecture

About this lecture

- Mostly about numerical solutions to Ordinary Differential Equations
  - Euler’s method, Runge-Kutte, built-in solvers

About this lecture

- Mostly about numerical solutions to Ordinary Differential Equations
    - Euler’s method, Runge-Kutte, built-in solvers
- Simulating Hodgkin-Huxley and the Leaky Integrate and Fire models

About this lecture

- Mostly about numerical solutions to Ordinary Differential Equations
    - Euler’s method, Runge-Kutte, built-in solvers

- Simulating Hodgkin-Huxley and the Leaky Integrate and Fire models

- Analysis of ODEs
    - Phase planes
    - Equilibrium points and null-clines
    - Stability, oscillations, bi-stability

About this lecture

- Mostly about numerical solutions to Ordinary Differential Equations
  - Euler's method, Runge-Kutte, built-in solvers
- Simulating Hodgkin-Huxley and the Leaky Integrate and Fire models
- Analysis of ODEs
  - Phase planes
  - Equilibrium points and null-clines
  - Stability, oscillations, bi-stability

$\rightarrow$ You can follow along with the Jupyter Notebook

The pre-reading material

Jupyter notebook with notes and examples for you to play around with.

The pre-reading material

Jupyter notebook with notes and examples for you to play around with.

- Ordinary Differential Equations
  - What they are
  - Analytical solutions
  - Understanding of differentiation and integration

The pre-reading material

Jupyter notebook with notes and examples for you to play around with.

- Ordinary Differential Equations
  - What they are
  - Analytical solutions
  - Understanding of differentiation and integration
- Linear Algebra
  - Vectors
  - Matrices, square matrices
  - Eigenvalues and eigenvectors

Outline

1. Numerical solutions to ODEs
2. Simulating HH and LIF
3. Qualitative analysis of ODEs
4. Summary

1. Numerical solutions to ODEs

2. Simulating HH and LIF

3. Qualitative analysis of ODEs

4. Summary

Population growth

A very simple ODE:

$$\frac{dN}{dt} = aN$$

- Describes how a population N grows with time t
- Assumes that the reproduction rate of each member in the population is a

Population growth

A very simple ODE:

$$\frac{dN}{dt} = aN$$

- Describes how a population N grows with time t
- Assumes that the reproduction rate of each member in the population is a

The solution can be derived analytically:

$$\frac{dN}{N} = a\,dt$$

Population growth

A very simple ODE:

$$\frac{dN}{dt} = aN$$

- Describes how a population N grows with time t
- Assumes that the reproduction rate of each member in the population is a

The solution can be derived analytically:

$$\frac{dN}{N} = a\ dt$$
$$\int \frac{dN}{N} = \int a dt$$

Population growth

A very simple ODE:

$$\frac{dN}{dt} = aN$$

- Describes how a population N grows with time t
- Assumes that the reproduction rate of each member in the population is a

The solution can be derived analytically:

$$\frac{dN}{N} = a\ dt$$

$$\int \frac{dN}{N} = \int a dt$$

$$\log(N) = at + C$$

Population growth

A very simple ODE:

$$\frac{dN}{dt} = aN$$

- Describes how a population N grows with time t
- Assumes that the reproduction rate of each member in the population is a

The solution can be derived analytically:

$$\frac{dN}{N} = a\ dt$$
$$\int \frac{dN}{N} = \int adt$$
$$\log(N) = at + C$$

Population growth

A very simple ODE:

$$\frac{dN}{dt} = aN$$

- Describes how a population N grows with time t
- Assumes that the reproduction rate of each member in the population is a

The solution can be derived analytically:

$$e^{log(N)} = e^{at+C}$$

Population growth

A very simple ODE:

$$\frac{dN}{dt} = aN$$

- Describes how a population N grows with time t
- Assumes that the reproduction rate of each member in the population is a

The solution can be derived analytically:

$$\begin{aligned} e^{log(N)} &= e^{at+C} \\ N &= e^{C}e^{at} \end{aligned}$$

Population growth

A very simple ODE:

$$\frac{dN}{dt} = aN$$

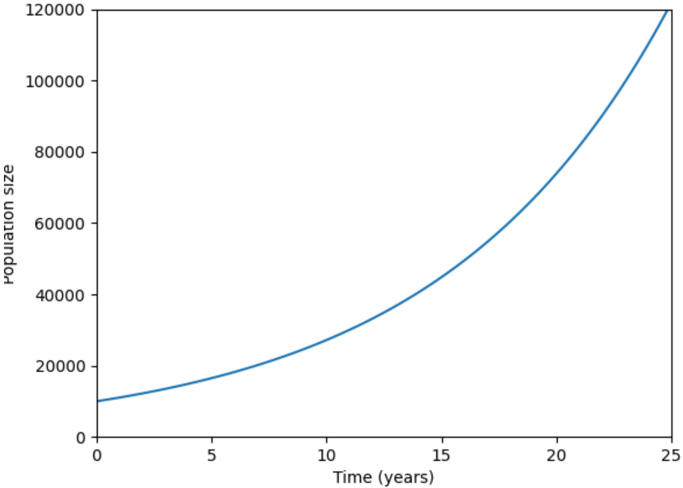
- Describes how a population N grows with time t
- Assumes that the reproduction rate of each member in the population is a

The solution can be derived analytically:

$$\begin{aligned} e^{log(N)} &= e^{at+C} \\ N &= e^C e^{at} \\ N &= N_0 e^{at} \end{aligned}$$

where $e^C = N_0$ is the initial population, i.e. at $t = 0$.

Population growth



Euler's method

Taylor expansion of a function $y(t)$ around a point a:

$$y(t) = y(a) + (t-a)\frac{dy}{dt}\bigg|_{x=a} + \frac{1}{2!}(t-a)^2\frac{d^2y}{dt^2}\bigg|_{x=a} + \ldots$$

Euler's method

Taylor expansion of a function $y(t)$ around a point a:

$$y(t) = y(a) + (t-a)\frac{dy}{dt}\Big|_{x=a} + \frac{1}{2!}(t-a)^2\frac{d^2y}{dt^2}\Big|_{x=a} + \dots$$

So if we have a first order ODE, we can approximate the solution at a point t_{n+1} with

$$y(t_{n+1}) \simeq y(t_n) + \Delta t\, f(y, t_n) \quad \text{where} \quad \frac{dy}{dt} = f(y,t) \quad \text{and} \quad \Delta t = t_{n+1} - t_n.$$

This is Euler's method.

Euler's method

Taylor expansion of a function $y(t)$ around a point a:

$$y(t) = y(a) + (t-a)\frac{dy}{dt}\bigg|_{x=a} + \frac{1}{2!}(t-a)^2\frac{d^2y}{dt^2}\bigg|_{x=a} + \ldots$$

So if we have a first order ODE, we can approximate the solution at a point t_{n+1} with

$$y(t_{n+1}) \simeq y(t_n) + \Delta t\, f(y, t_n) \quad \text{where} \quad \frac{dy}{dt} = f(y,t) \quad \text{and} \quad \Delta t = t_{n+1} - t_n.$$

This is Euler's method.

The population growth example:

$$\frac{dN}{dt} = aN \quad \Rightarrow \quad N(t_{n+1}) \simeq N(t_n) + \Delta t\; a\, N(t_n)$$

where $N(t_0)$ is a boundary (inital value) condition.

Euler's method

$\rightarrow$ code example from the Jupyter notebook

To do this in code we:

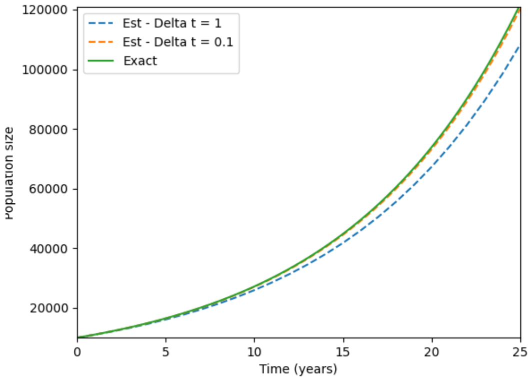
- choose a time step Δt
- choose an initial value $N(t_0)$
- use $N(t_0)$ compute $N(t_1)$
- use $N(t_1)$ compute $N(t_2)$
- $\ldots$
- build the solution $N(t)$ iteratively

Euler's method

$\rightarrow$ code example from the Jupyter notebook

To do this in code we:

- choose a time step Δt
- choose an initial value $N(t_0)$
- use $N(t_0)$ compute $N(t_1)$
- use $N(t_1)$ compute $N(t_2)$
- $\ldots$
- build the solution $N(t)$ iteratively



smaller time steps $\Rightarrow$ better accuracy

Runge-Kutta method

RK uses multiple slope evaluations in each interval Δt to obtain higher accuracy.

$\rightarrow$ they indirectly approximate higher-order terms in the Taylor series.

Runge-Kutta method

RK uses multiple slope evaluations in each interval Δt to obtain higher accuracy.

$\rightarrow$ they indirectly approximate higher-order terms in the Taylor series.

The RK approximation is

$$y(t_{n+1}) = y(t_n) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Runge-Kutta method

RK uses multiple slope evaluations in each interval Δt to obtain higher accuracy.

$\rightarrow$ they indirectly approximate higher-order terms in the Taylor series.

The RK approximation is

$$y(t_{n+1}) = y(t_n) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$k_1 = \Delta t\, f(y_n)$ just the term from the Euler method

Runge-Kutta method

RK uses multiple slope evaluations in each interval Δt to obtain higher accuracy.

$\rightarrow$ they indirectly approximate higher-order terms in the Taylor series.

The RK approximation is

$$y(t_{n+1}) = y(t_n) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$k_1 = \Delta t\, f\,(y_n)$ just the term from the Euler method

$k_2 = \Delta t\, f\left(y_n + \frac{1}{2}k_1\right)$ derivative at the midpoint

Runge-Kutta method

RK uses multiple slope evaluations in each interval Δt to obtain higher accuracy.

$\rightarrow$ they indirectly approximate higher-order terms in the Taylor series.

The RK approximation is

$$y(t_{n+1}) = y(t_n) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$k_1 = \Delta t\, f\left(y_n\right) \qquad \text{just the term from the Euler method}$$

$$k_2 = \Delta t\, f\left(y_n + \tfrac{1}{2}k_1\right) \qquad \text{derivative at the midpoint}$$

$$k_3 = \Delta t\, f\left(y_n + \tfrac{1}{2}k_2\right) \qquad \text{derivative at the midpoint}$$

Runge-Kutta method

RK uses multiple slope evaluations in each interval Δt to obtain higher accuracy.

$\rightarrow$ they indirectly approximate higher-order terms in the Taylor series.

The RK approximation is

$$y(t_{n+1}) = y(t_n) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$\begin{aligned}
k_1 &= \Delta t\, f\,(y_n) && \text{just the term from the Euler method} \\
k_2 &= \Delta t\, f\left(y_n + \tfrac{1}{2}k_1\right) && \text{derivative at the midpoint} \\
k_3 &= \Delta t\, f\left(y_n + \tfrac{1}{2}k_2\right) && \text{derivative at the midpoint} \\
k_4 &= \Delta t\, f\,(y_n + k_3) && \text{derivative at the endpoint}
\end{aligned}$$

Runge-Kutta method

RK uses multiple slope evaluations in each interval Δt to obtain higher accuracy.

$\rightarrow$ they indirectly approximate higher-order terms in the Taylor series.

The RK approximation is

$$y(t_{n+1}) = y(t_n) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$\begin{aligned}
k_1 &= \Delta t\, f\,(y_n) && \text{just the term from the Euler method} \\
k_2 &= \Delta t\, f\left(y_n + \tfrac{1}{2}k_1\right) && \text{derivative at the midpoint} \\
k_3 &= \Delta t\, f\left(y_n + \tfrac{1}{2}k_2\right) && \text{derivative at the midpoint} \\
k_4 &= \Delta t\, f\,(y_n + k_3) && \text{derivative at the endpoint}
\end{aligned}$$

The solution is then calculated iteratively as in the Euler case.

Runge-Kutta method

$\rightarrow$ code example from the Jupyter notebook

To implement RK in code for the population growth case, $\frac{dN}{dt} = aN$, we:

- choose a time step Δt
- choose an initial value $N(t_0)$

Runge-Kutta method

$\rightarrow$ code example from the Jupyter notebook

To implement RK in code for the population growth case, $\frac{dN}{dt} = aN$, we:

- choose a time step Δt
- choose an initial value $N(t_0)$
- use $N(t_0)$ to compute $k_1 = \Delta t\, f(N_0) = \Delta t\; a\; N(t_0)$

Runge-Kutta method

$\rightarrow$ code example from the Jupyter notebook

To implement RK in code for the population growth case, $\frac{dN}{dt} = aN$, we:

- choose a time step Δt
- choose an initial value $N(t_0)$
- use $N(t_0)$ to compute $k_1 = \Delta t\, f(N_0) = \Delta t\; a\; N(t_0)$
- use k_1 to compute $k_2 = \Delta t\, f\left(N_0 + \frac{1}{2}k_1\right) = \Delta t\; a\; N(t_0)\left(1 + \frac{1}{2}\Delta t\; a\right)$

Runge-Kutta method

→ code example from the Jupyter notebook

To implement RK in code for the population growth case, $\frac{dN}{dt} = aN$, we:

- choose a time step Δt
- choose an initial value $N(t_0)$
- use $N(t_0)$ to compute $k_1 = \Delta t\, f(N_0) = \Delta t\; a\; N(t_0)$
- use k_1 to compute $k_2 = \Delta t\, f\left(N_0 + \frac{1}{2}k_1\right) = \Delta t\; a\; N(t_0)\left(1 + \frac{1}{2}\Delta t\; a\right)$
- use k_2 to compute $k_3 = \Delta t\, f\left(N_0 + \frac{1}{2}k_2\right) = \Delta t\; a\; N(t_0)\left(1 + \frac{1}{2}\Delta t\; a + \frac{1}{4}\Delta t^2\; a^2\right)$

Runge-Kutta method

$\rightarrow$ code example from the Jupyter notebook

To implement RK in code for the population growth case, $\frac{dN}{dt} = aN$, we:

- choose a time step Δt
- choose an initial value $N(t_0)$
- use $N(t_0)$ to compute $k_1 = \Delta t\, f(N_0) = \Delta t\; a\; N(t_0)$
- use k_1 to compute $k_2 = \Delta t\, f\left(N_0 + \frac{1}{2}k_1\right) = \Delta t\; a\; N(t_0)\left(1 + \frac{1}{2}\Delta t\; a\right)$
- use k_2 to compute $k_3 = \Delta t\, f\left(N_0 + \frac{1}{2}k_2\right) = \Delta t\; a\; N(t_0)\left(1 + \frac{1}{2}\Delta t\; a + \frac{1}{4}\Delta t^2\; a^2\right)$
- use k_3 to compute $k_4 = \Delta t\, f\left(N_0 + k_3\right) = \Delta t\; a\; N(t_0)\left(1 + \Delta t\; a + \frac{1}{2}\Delta t^2\; a^2 + \frac{1}{4}\Delta t^3\; a^3\right)$

Runge-Kutta method

$\rightarrow$ code example from the Jupyter notebook

To implement RK in code for the population growth case, $\frac{dN}{dt} = aN$, we:

- choose a time step Δt
- choose an initial value $N(t_0)$
- use $N(t_0)$ to compute $k_1 = \Delta t\, f(N_0) = \Delta t\; a\; N(t_0)$
- use k_1 to compute $k_2 = \Delta t\, f\left(N_0 + \frac{1}{2}k_1\right) = \Delta t\; a\; N(t_0)\left(1 + \frac{1}{2}\Delta t\; a\right)$
- use k_2 to compute $k_3 = \Delta t\, f\left(N_0 + \frac{1}{2}k_2\right) = \Delta t\; a\; N(t_0)\left(1 + \frac{1}{2}\Delta t\; a + \frac{1}{4}\Delta t^2\; a^2\right)$
- use k_3 to compute $k_4 = \Delta t\, f\left(N_0 + k_3\right) = \Delta t\; a\; N(t_0)\left(1 + \Delta t\; a + \frac{1}{2}\Delta t^2\; a^2 + \frac{1}{4}\Delta t^3\; a^3\right)$
- compute $N(t_1) = N(t_0) + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$
- $\ldots$
- build the solution $N(t)$ iteratively

Runge-Kutta method

$\rightarrow$ code example from the Jupyter notebook

$\rightarrow$ higher order terms in Δt provide higher accuracy!

The linearity of $f(N)$ makes this simpler, we can see it exactly

Runge-Kutta method

$\rightarrow$ code example from the Jupyter notebook

$\rightarrow$ higher order terms in Δt provide higher accuracy!

The linearity of $f(N)$ makes this simpler, we can see it exactly

$$\frac{dN}{dt} = aN \quad \Rightarrow \quad N(t) = N_0 e^{at}$$

Runge-Kutta method

→ code example from the Jupyter notebook

→ higher order terms in Δt provide higher accuracy!

The linearity of $f(N)$ makes this simpler, we can see it exactly

$$\frac{dN}{dt} = aN \quad \Rightarrow \quad N(t) = N_0 e^{at}$$

performing a Taylor expansion:

$$\simeq N_0 \left[1 + at + \tfrac{1}{2}a^2t^2 + \tfrac{1}{6}a^3t^3 + \tfrac{1}{12}a^4t^4 + \mathcal{O}\left((at)^5\right)\right]$$

Runge-Kutta method

$\rightarrow$ code example from the Jupyter notebook

$\rightarrow$ higher order terms in Δt provide higher accuracy!

The linearity of $f(N)$ makes this simpler, we can see it exactly

$$\frac{dN}{dt} = aN \quad \Rightarrow \quad N(t) = N_0 e^{at}$$

performing a Taylor expansion:

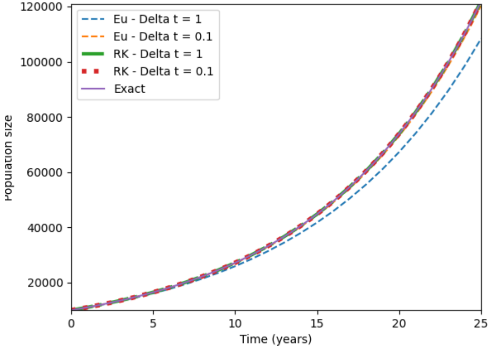
$$\simeq N_0 \left[1 + at + \tfrac{1}{2}a^2t^2 + \tfrac{1}{6}a^3t^3 + \tfrac{1}{12}a^4t^4 + \mathcal{O}\left((at)^5\right)\right]$$

$$\equiv \text{Runge-Kutta solution} + \mathcal{O}\left((at)^5\right)$$

This is true in general, RK is accurate to fourth order in the expansion.
(Euler accurate to first order...)

Runge-Kutta method

→ code example from the Jupyter notebook



Tip: try plotting the relative errors of these approximations.

Built-in solvers

You don’t need to implement these yourselves...

There are lots of packages out there, scipy provides one good option:

```
from scipy.integrate import solve_ivp

def dNdt_ivp(t,N,a):
 return a*N

sol = solve_ivp(dNdt_ivp, [0, 25], [N0], args=(a,), dense_output=True)
```

See the docs:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html

The Hodgkin-Huxley model

Describes changes in a neurons membrane potential (V) as a function of time.

$$
\begin{aligned}
C\frac{\mathrm{d}V}{\mathrm{d}t} &= I_A - \overline{g}_{\mathrm{Na}} m^3 h (V - V_{\mathrm{Na}}) - \overline{g}_{\mathrm{K}} n^4 (V - V_{\mathrm{K}}) - \overline{g}_l (V - V_l) \\
\frac{\mathrm{d}n}{\mathrm{d}t} &= \alpha_n(V)(1 - n) - \beta_n(V) n \\
\frac{\mathrm{d}m}{\mathrm{d}t} &= \alpha_m(V)(1 - m) - \beta_m(V) m \\
\frac{\mathrm{d}h}{\mathrm{d}t} &= \alpha_h(V)(1 - h) - \beta_h(V) h
\end{aligned}
$$

V : membrane electric potential difference between the inside and outside of the cell

The Hodgkin-Huxley model

Describes changes in a neurons membrane potential (V) as a function of time.

$$
\begin{aligned}
C\frac{\mathrm{d}V}{\mathrm{d}t} &= I_A - \overline{g}_{\mathrm{Na}} m^3 h (V - V_{\mathrm{Na}}) - \overline{g}_{\mathrm{K}} n^4 (V - V_{\mathrm{K}}) - \overline{g}_l (V - V_l) \\
\frac{\mathrm{d}n}{\mathrm{d}t} &= \alpha_n(V)(1 - n) - \beta_n(V) n \\
\frac{\mathrm{d}m}{\mathrm{d}t} &= \alpha_m(V)(1 - m) - \beta_m(V) m \\
\frac{\mathrm{d}h}{\mathrm{d}t} &= \alpha_h(V)(1 - h) - \beta_h(V) h
\end{aligned}
$$

I_A : the current applied to the neuron

The Hodgkin-Huxley model

Describes changes in a neurons membrane potential (V) as a function of time.

$$
\begin{aligned}
C\frac{\mathrm{d}V}{\mathrm{d}t} &= I_A - \overline{g}_{\mathrm{Na}} m^3 h (V - V_{\mathrm{Na}}) - \overline{g}_{\mathrm{K}} n^4 (V - V_{\mathrm{K}}) - \overline{g}_l (V - V_l) \\
\frac{\mathrm{d}n}{\mathrm{d}t} &= \alpha_n(V)(1 - n) - \beta_n(V) n \\
\frac{\mathrm{d}m}{\mathrm{d}t} &= \alpha_m(V)(1 - m) - \beta_m(V) m \\
\frac{\mathrm{d}h}{\mathrm{d}t} &= \alpha_h(V)(1 - h) - \beta_h(V) h
\end{aligned}
$$

V_{Na}, V_{K}, V_l: Sodium, Potassium, and leak potentials

The Hodgkin-Huxley model

Describes changes in a neurons membrane potential (V) as a function of time.

$$
\begin{aligned}
C\frac{\mathrm{d}V}{\mathrm{d}t} &= I_A - \overline{g}_{\mathrm{Na}} m^3 h(V - V_{\mathrm{Na}}) - \overline{g}_{\mathrm{K}} n^4 (V - V_{\mathrm{K}}) - \overline{g}_l (V - V_l) \\
\frac{\mathrm{d}n}{\mathrm{d}t} &= \alpha_n(V)(1 - n) - \beta_n(V)n \\
\frac{\mathrm{d}m}{\mathrm{d}t} &= \alpha_m(V)(1 - m) - \beta_m(V)m \\
\frac{\mathrm{d}h}{\mathrm{d}t} &= \alpha_h(V)(1 - h) - \beta_h(V)h
\end{aligned}
$$

$\overline{g}_{\mathrm{Na}}$, $\overline{g}_{\mathrm{K}}$, $\overline{g}_l$: Sodium, Potassium, and leak conductances

The Hodgkin-Huxley model

Describes changes in a neurons membrane potential (V) as a function of time.

$$
\begin{aligned}
C\frac{\mathrm{d}V}{\mathrm{d}t} &= I_A - \overline{g}_{\mathrm{Na}} m^3 h (V - V_{\mathrm{Na}}) - \overline{g}_{\mathrm{K}} n^4 (V - V_{\mathrm{K}}) - \overline{g}_l (V - V_l) \\
\frac{\mathrm{d}n}{\mathrm{d}t} &= \alpha_n(V)(1 - n) - \beta_n(V)n \\
\frac{\mathrm{d}m}{\mathrm{d}t} &= \alpha_m(V)(1 - m) - \beta_m(V)m \\
\frac{\mathrm{d}h}{\mathrm{d}t} &= \alpha_h(V)(1 - h) - \beta_h(V)h
\end{aligned}
$$

C : capacitance of the membrane $Q = CV$

The Hodgkin-Huxley model

Describes changes in a neurons membrane potential (V) as a function of time.

$$
\begin{aligned}
C\frac{\mathrm{d}V}{\mathrm{d}t} &= I_A - \overline{g}_{\mathrm{Na}} m^3 h (V - V_{\mathrm{Na}}) - \overline{g}_{\mathrm{K}} n^4 (V - V_{\mathrm{K}}) - \overline{g}_l (V - V_l) \\
\frac{\mathrm{d}n}{\mathrm{d}t} &= \alpha_n(V)(1 - n) - \beta_n(V)n \\
\frac{\mathrm{d}m}{\mathrm{d}t} &= \alpha_m(V)(1 - m) - \beta_m(V)m \\
\frac{\mathrm{d}h}{\mathrm{d}t} &= \alpha_h(V)(1 - h) - \beta_h(V)h
\end{aligned}
$$

$n,\ m,\ h$: gating variables $\in [0, 1]$

The Hodgkin-Huxley model

Describes changes in a neurons membrane potential (V) as a function of time.

$$
\begin{aligned}
C\frac{\mathrm{d}V}{\mathrm{d}t} &= I_A - \overline{g}_{\mathrm{Na}} m^3 h (V - V_{\mathrm{Na}}) - \overline{g}_{\mathrm{K}} n^4 (V - V_{\mathrm{K}}) - \overline{g}_l (V - V_l) \\
\frac{\mathrm{d}n}{\mathrm{d}t} &= \alpha_n(V)(1-n) - \beta_n(V)n \\
\frac{\mathrm{d}m}{\mathrm{d}t} &= \alpha_m(V)(1-m) - \beta_m(V)m \\
\frac{\mathrm{d}h}{\mathrm{d}t} &= \alpha_h(V)(1-h) - \beta_h(V)h
\end{aligned}
$$

$\alpha_{n,m,h}(V)$ & $\beta_{n,m,h}(V)$: transcendental functions of V chosen to fit expt data

The Hodgkin-Huxley model

- System of ODEs
- The solutions depend on one another
- First order - IVP
  use methods like RK
- Non-linear!
  can't write out simple solutions
- More interesting results :-)

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = I - \overline{g}_{\mathrm{Na}} m^3 h (V - V_{\mathrm{Na}}) - \overline{g}_{\mathrm{K}} n^4 (V - V_{\mathrm{K}}) - \overline{g}_l (V - V_l)$$

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \alpha_n(V)(1-n) - \beta_n(V)n$$

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \alpha_m(V)(1-m) - \beta_m(V)m$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \alpha_h(V)(1-h) - \beta_h(V)h$$

The Hodgkin-Huxley model

We can define a function in python to return the derivatives of the parameters:

```
def HH(t,x,I):
  V, n, m, h = x
  alpha_n = 0.01*(V+55)/(1-np.exp(-0.1*(V+55)))
  ...
  beta_n = 0.125*np.exp(-0.0125*(V+65))
  ...
  dVdt = (1/C)*(I - gNa*m**3*h*(V-VNa) - gK*n**4*(V-VK) - gL*(V-VL))
  dndt = alpha_n*(1-n) - beta_n*n
  dmdt = alpha_m*(1-m) - beta_m*m
  dhdt = alpha_h*(1-h) - beta_h*h
  return [dVdt, dndt, dmdt, dhdt]
```

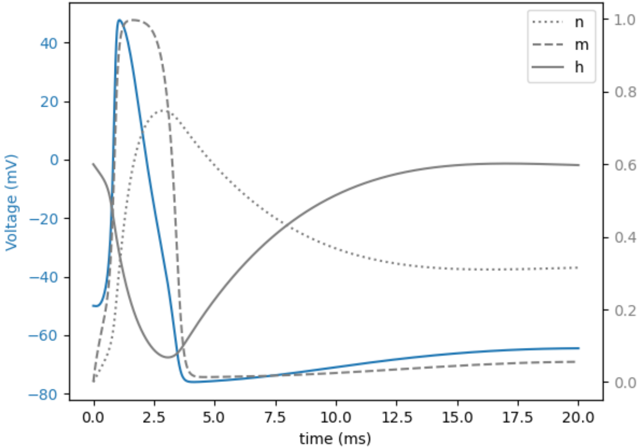
The Hodgkin-Huxley model

Then we can use this function to generate the solution:

```
# Define parameters
C = 1
gNa = 120
gK = 36
gL = 0.3
VNa = 50
VK = -77
VL = -54.402

I=0
# Simulate model
HH_sol = solve_ivp(HH, [0,20], [-50,0,0,0.6], dense_output = True, args = (I,))
```

The Hodgkin-Huxley model



Try the exercise in the Jupyter notebook!

The Leaky Integrate and Fire model

Much simpler approximation to the physical system than HH

The model is described by a single ODE

$$\tau_m \frac{dV}{dt} = (V_{\text{rest}} - V) + R_m\, I_e$$

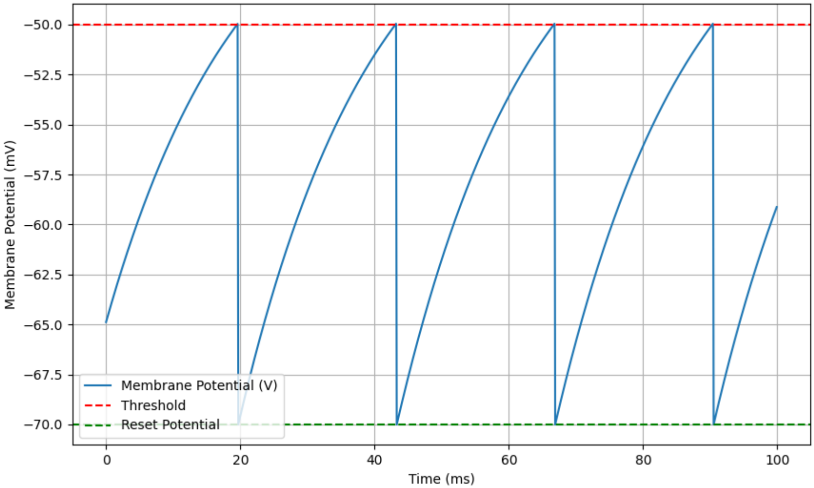
with a reset condition:

$$\text{if } V > V_{\text{threshold}} \;:\; V \leftarrow V_{\text{reset}}.$$

The ODE is linear, it is the reset condition that generates the 'spike'

We can easily solve this using the same method as for HH

The Leaky Integrate and Fire model



Qualitative analysis of ODEs

So far:

- Used Euler's method to solve an ODE - $(N(t))$
- Used Runge-Kutte method - $(N(t))$
- Moved on to systems of ODEs - $(V(t),\ n(t),\ m(t),\ h(t))$
- In each case we:
  - choose some intial conditions
  - evolve each variable in time

Qualitative analysis of ODEs

So far:

- Used Euler's method to solve an ODE - $(N(t))$
- Used Runge-Kutte method - $(N(t))$
- Moved on to systems of ODEs - $(V(t),\ n(t),\ m(t),\ h(t))$
- In each case we:
  - choose some intial conditions
  - evolve each variable in time

We want a better way to understand the solutions from a global perspective

i.e. **within the whole space of solutions**

Qualitative analysis of ODEs

So far:

- Used Euler's method to solve an ODE - $(N(t))$
- Used Runge-Kutte method - $(N(t))$
- Moved on to systems of ODEs - $(V(t),\ n(t),\ m(t),\ h(t))$
- In each case we:
  - choose some intial conditions
  - evolve each variable in time

We want a better way to understand the solutions from a global perspective

i.e. **within the whole space of solutions**

Easy to demonstrate in 2 dimensions, so, we'll introduce another model..

The Morris-Lecar model

2D approximation to the HH model

$\rightarrow$ assume that Na/Ca gates operate on much faster timescales ($t \rightarrow t_\infty$)
$\Rightarrow$ don't need $\frac{dm}{dt}$ or $\frac{dh}{dt}$

The Morris-Lecar model

2D approximation to the HH model

$\rightarrow$ assume that Na/Ca gates operate on much faster timescales ($t \rightarrow t_\infty$)
$\Rightarrow$ don't need $\frac{dm}{dt}$ or $\frac{dh}{dt}$

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = I - g_L(V - V_L) - g_K w(V - V_K) - g_{Ca} m_\infty(V)(V - V_{Ca})$$
$$\frac{\mathrm{d}w}{\mathrm{d}t} = \phi(w_\infty(V) - w)/\tau_w(V)$$

where

$$m_\infty(V) = 0.5(1 + \tanh((V - V_1)/V_2))$$
$$w_\infty(V) = 0.5(1 + \tanh((V - V_3)/V_4))$$
$$1/\tau_w(V) = \cosh((V - V_3)/2V_4)$$

The Morris-Lecar model

$\rightarrow$ code example from the Jupyter notebook

Let's simulate the model using solve_ivp from scipy for $I = 0$ and
$[V_0, w_0] = [-40, 0],\ [-20, 0],\ [-15, 0],\ [+20, 0]$

The Morris-Lecar model

→ code example from the Jupyter notebook

Let's simulate the model using solve_ivp from scipy for $I = 0$ and $[V_0, w_0] = [-40, 0],\ [-20, 0],\ [-15, 0],\ [+20, 0]$

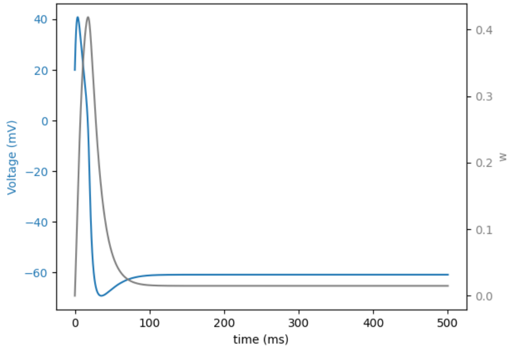
```
# Define the ODEs
def ML(t,x,I):
 V = x[0]
 w = x[1]
 ...
 return [dVdt, dwdt]

# Simulate model for different initial conditions
ML_sol1 = solve_ivp(ML, [0,500], [-40,0.0], dense_output = True, args = (I,))
...
```

The Morris-Lecar model

$\rightarrow$ code example from the Jupyter notebook

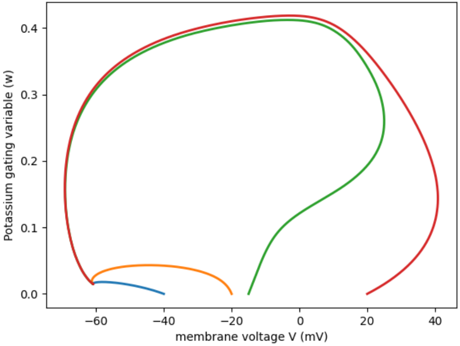
We can plot the solutions $V(t)$ and $w(t)$ as a function of time, e.g. for $[+20, 0]$:



The Morris-Lecar model

→ code example from the Jupyter notebook

But, only two variables V and w, we can plot $w(V)$ for the different initial values:



The Morris-Lecar model

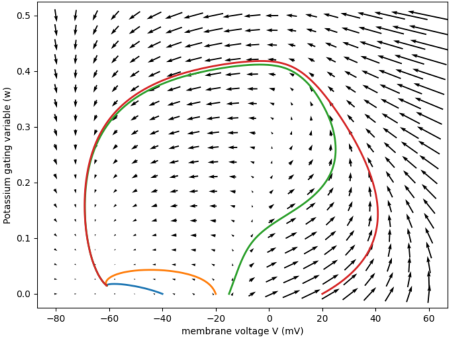
$V(t+\Delta t) = V(t) + \Delta t\ \frac{\Delta V}{\Delta t}$ & $w(t+\Delta t) = w(t) + \Delta t\ \frac{\Delta w}{\Delta t}$

Velocity vectors $\left(\frac{dV}{dt}, \frac{dw}{dt}\right)$ tell which direction the solutions flow in time
+ how fast they move

The Morris-Lecar model

$V(t{+}\Delta t) = V(t) + \Delta t\, \frac{\Delta V}{\Delta t}$ & $w(t{+}\Delta t) = w(t) + \Delta t\, \frac{\Delta w}{\Delta t}$

Velocity vectors $\left(\frac{dV}{dt}, \frac{dw}{dt}\right)$ tell which direction the solutions flow in time + how fast they move



Equilibrium points and null clines

Let's write:

$$C\frac{dV}{dt} = I + F(V, w)$$
$$\frac{dw}{dt} = \phi(w_\infty(V) - w)/\tau_w(V)$$

Equilibrium points given by points satisying:

$$\frac{dV}{dt} = 0 \quad \& \quad \frac{dw}{dt} = 0$$

Equilibrium points and null clines

Let's write:

$$C\frac{dV}{dt} = I + F(V, w)$$
$$\frac{dw}{dt} = \phi(w_\infty(V) - w)/\tau_w(V)$$

Equilibrium points given by points satisying:

$$\frac{dV}{dt} = 0 \quad \& \quad \frac{dw}{dt} = 0$$

These conditions amount to:

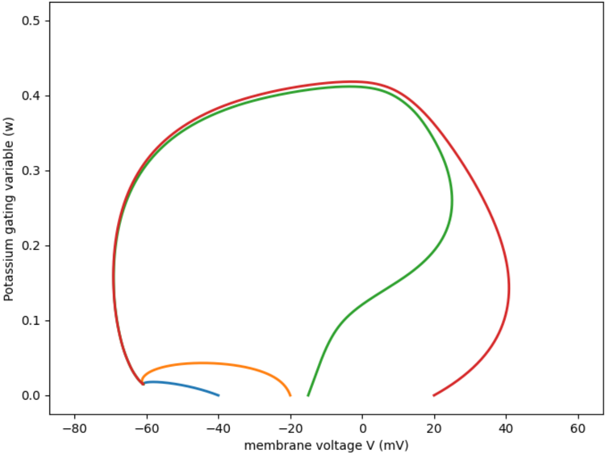
$$I + F(V, w) = 0 \quad \text{and} \quad w = w_\infty(V).$$

The solutions to these equations are called **nullclines**
- lines where either V or w is constant

nullclines all intersect at equilibrium points

Equilibrium points and null clines

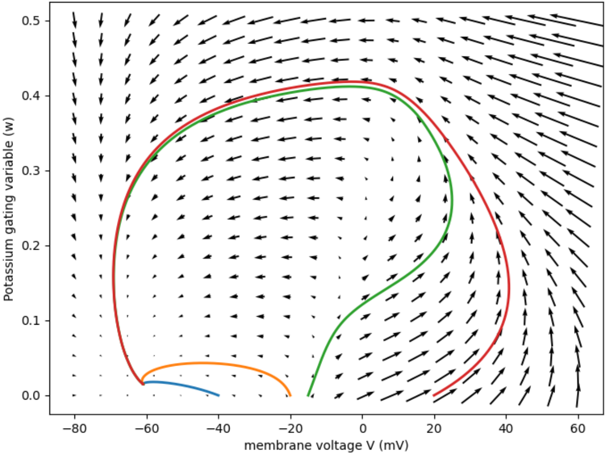
solid lines:
different initial values



Equilibrium points and null clines

solid lines:
different initial values

arrows:
vector-field $\left(\frac{dV}{dt}, \frac{dw}{dt}\right)$

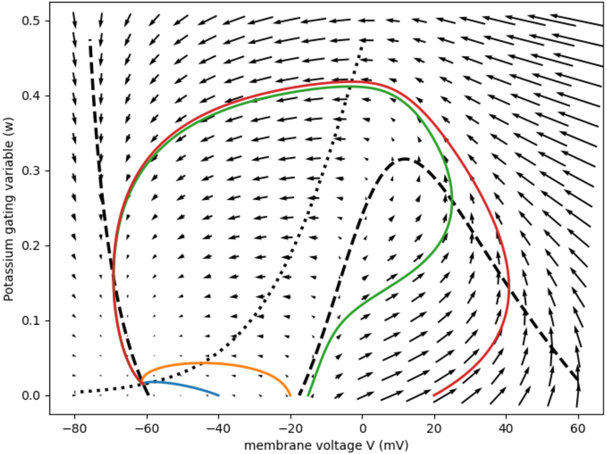


Equilibrium points and null clines

solid lines:
different initial values

arrows:
vector-field $\left(\frac{dV}{dt}, \frac{dw}{dt}\right)$

dashed - V nullcline
dotted - w nullcline



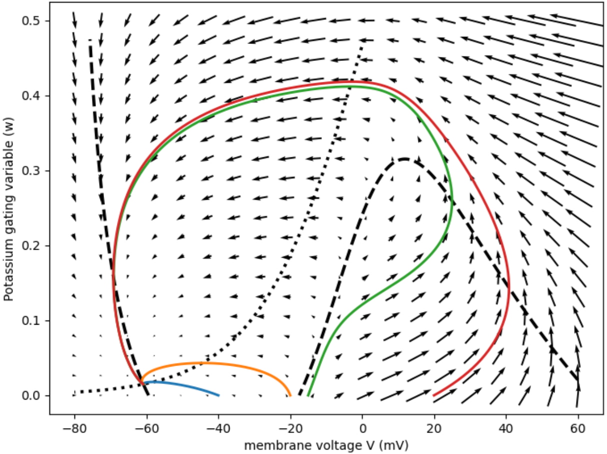
Equilibrium points and null clines

solid lines:
different initial values

arrows:
vector-field $\left(\frac{dV}{dt}, \frac{dw}{dt}\right)$

dashed - V nullcline
dotted - w nullcline

asymptotically stable equilibrium point



Stability of equilibrium points

In general:

$$\frac{dx}{dt} = f(x,y), \qquad \frac{dy}{dt} = g(x,y)$$

Euilibrium point at $(\bar{x}, \bar{y})$ where $f(\bar{x}, \bar{y}) = 0$ and $g(\bar{x}, \bar{y}) = 0$.

Stability of equilibrium points

In general:

$$\frac{dx}{dt} = f(x,y), \qquad \frac{dy}{dt} = g(x,y)$$

Euilibrium point at $(\overline{x},\overline{y})$ where $f(\overline{x},\overline{y}) = 0$ and $g(\overline{x},\overline{y}) = 0$.

Stable equilibrium $\Rightarrow$ Perturbations from $(\overline{x},\overline{y}) \to 0$ as time goes on.

Stability of equilibrium points

In general:

$$\frac{dx}{dt} = f(x, y), \qquad \frac{dy}{dt} = g(x, y)$$

Euilibrium point at $(\bar{x}, \bar{y})$ where $f(\bar{x}, \bar{y}) = 0$ and $g(\bar{x}, \bar{y}) = 0$.

Stable equilibrium $\Rightarrow$ Perturbations from $(\bar{x}, \bar{y}) \rightarrow 0$ as time goes on.

Make small perturbations: $x = \bar{x} + u$, $y = \bar{y} + v$

Then Taylor expand, assuming the perturbations are small:

$$\frac{du}{dt} = f(\bar{x} + u, \bar{y} + v) \approx f(\bar{x}, \bar{y}) + \frac{\partial f}{\partial x}(\bar{x}, \bar{y})u + \frac{\partial f}{\partial y}(\bar{x}, \bar{y})v + \ldots$$

$$\frac{dv}{dt} = g(\bar{x} + u, \bar{y} + v) \approx g(\bar{x}, \bar{y}) + \frac{\partial g}{\partial x}(\bar{x}, \bar{y})u + \frac{\partial g}{\partial y}(\bar{x}, \bar{y})v + \ldots .$$

we want to find solutions for the perturbations to first order

Stability of equilibrium points

But we can re-write this as a matrix equation with $\mathbf{u} = (u, v)^T$:

$$\frac{d\mathbf{u}}{dt} = J\mathbf{u} \quad \text{where} \quad J = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{bmatrix}_{(\bar{x},\bar{y})}.$$

The matrix of partial derivatives J is called the Jacobian.

Stability of equilibrium points

But we can re-write this as a matrix equation with $\mathbf{u} = (u, v)^T$:

$$\frac{d\mathbf{u}}{dt} = J\mathbf{u} \quad \text{where} \quad J = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{bmatrix}_{(\bar{x},\bar{y})}.$$

The matrix of partial derivatives J is called the Jacobian.

Now let's look for solutions of the form $\mathbf{u} = \mathrm{e}^{\lambda t}\mathbf{u}_0$

$\Rightarrow \lambda$ is a scalar, we now have an eigenvalue equation: (see pre-read)

$$\lambda \mathbf{u}_0 = J\mathbf{u}_0$$

Stability of equilibrium points

But we can re-write this as a matrix equation with $\mathbf{u} = (u, v)^T$:

$$\frac{d\mathbf{u}}{dt} = J\mathbf{u} \quad \text{where} \quad J = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{bmatrix}_{(\bar{x},\bar{y})}.$$

The matrix of partial derivatives J is called the Jacobian.

Now let's look for solutions of the form $\mathbf{u} = \mathrm{e}^{\lambda t}\mathbf{u}_0$

$\Rightarrow \lambda$ is a scalar, we now have an eigenvalue equation: (see pre-read)

$$\lambda \mathbf{u}_0 = J\mathbf{u}_0$$

- $\lambda_{1,2} < 0 \quad \Rightarrow$ Stable
- $\lambda_{1(2)} < 0, \ \lambda_{2(1)} > 0, \quad \Rightarrow$ Unstable saddle-point
- $\lambda_{1,2} > 0 \quad \Rightarrow$ Unstable

Bifurcations

As we change parameters in the system, e.g. the current I, the phase diagram changes

E.g. a bifurcation - a change in the type/number of equilibrium (fixed) points.

Bifurcations

As we change parameters in the system, e.g. the current I, the phase diagram changes

E.g. a bifurcation - a change in the type/number of equilibrium (fixed) points.

In general J has two eigenvalues $\lambda_{1,2}$ that are the roots of the quadratic

$$\lambda^2 - \mathrm{Trace}(J)\lambda + \det(J) = 0$$

where

$$\mathrm{Trace}(J) = \frac{\partial f}{\partial x}(\bar{x},\bar{y}) + \frac{\partial g}{\partial y}(\bar{x},\bar{y}), \quad \det(J) = \frac{\partial f}{\partial x}(\bar{x},\bar{y})\frac{\partial g}{\partial y}(\bar{x},\bar{y}) - \frac{\partial f}{\partial y}(\bar{x},\bar{y})\frac{\partial g}{\partial x}(\bar{x},\bar{y}).$$

Bifurcations

As we change parameters in the system, e.g. the current I, the phase diagram changes

E.g. a bifurcation - a change in the type/number of equilibrium (fixed) points.

If we start with a stable fixed point ($\lambda_{1,2} > 0$), we consider two changes as I varies:

- **Saddle-node bifurcation**
  - one λ goes < 0 as $\det(J)$ does through 0
  - then left with an unstable fixed point

Bifurcations

As we change parameters in the system, e.g. the current I, the phase diagram changes

E.g. a bifurcation - a change in the type/number of equilibrium (fixed) points.

If we start with a stable fixed point ($\lambda_{1,2} > 0$), we consider two changes as I varies:

- **Saddle-node bifurcation**
  - one λ goes < 0 as $\det(J)$ does through 0
  - then left with an unstable fixed point

- **Hopf bifurcation**
  - $\mathrm{Trace}(J) = 0$ and $\det(J) > 0$
  - **Complex eigenvalues** $\rightarrow \lambda = \lambda^*$

Saddle-node bifurcations

If we start with a stable fixed-point, we require $\det(J)$ to cross zero

In Morris-Lecar, if we assume that τ_m is slow-varying, we have

$$J = \begin{bmatrix} \frac{1}{C}\frac{\partial F}{\partial V} & \frac{1}{C}\frac{\partial F}{\partial w} \\ \frac{\phi}{\tau_w}\frac{\partial w_\infty}{\partial V} & -\frac{\phi}{\tau_w} \end{bmatrix}_{(\bar{V}(I),\bar{w}(I))}$$

and so we can derive:

$$\det(J) = -\frac{\phi}{C\tau_w}\left(\frac{\partial F}{\partial V} + \frac{\partial F}{\partial w}\frac{\partial w_\infty}{\partial V}\right) = \frac{\phi}{C\tau_w}\frac{\mathrm{d}I_{ss}}{\mathrm{d}V}$$

where I_{ss} is the current at the equilibrium point.

Now, by inspection (see the notebook) we can see that $\frac{dI_{ss}}{dV} \geq 0$
$\Rightarrow$ no $\det(J) = 0$ and no saddle-node bifurcation in Morris-Lecar!

Hopf bifurcations

Complex eigenvalues $\Rightarrow \mathbf{u} = e^{(\alpha \pm i\beta)t}\mathbf{u}_0, \qquad e^{i\beta t} = \cos(\beta t) + i\sin(\beta t)$

rightarrow the general solution looks like:

$$\mathbf{u} = e^{\alpha}\left(c_1\cos(\beta t) + c_2\sin(\beta t)\right)\mathbf{u}_0$$

so we have:

- The real part α determines whether oscillations grow or die
- The imaginary part β determines the oscillation frequency

Hopf bifurcations

Complex eigenvalues $\Rightarrow \mathbf{u} = e^{(\alpha \pm i\beta)t}\mathbf{u}_0$, $\qquad e^{i\beta t} = \cos(\beta t) + i\sin(\beta t)$

rightarrow the general solution looks like:

$$\mathbf{u} = e^{\alpha}\left(c_1\cos(\beta t) + c_2\sin(\beta t)\right)\mathbf{u}_0$$

so we have:

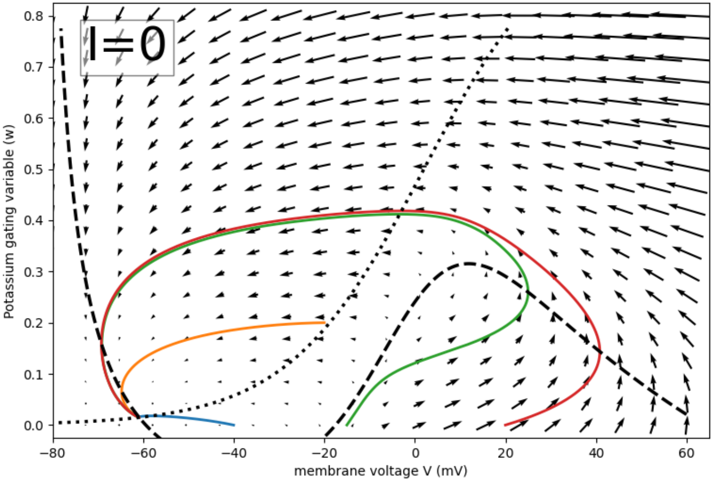
- The real part α determines whether oscillations grow or die
- The imaginary part β determines the oscillation frequency

A Hopf bifurcation occurs when $\text{Trace}(J) = 0$, which in Morris-Lecar means:

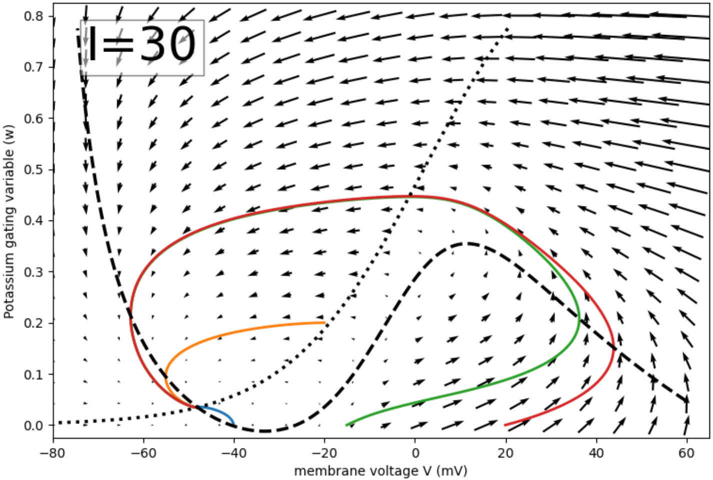
$$\frac{1}{C}\frac{\partial F}{\partial V}(\overline{V}, \overline{w}) = \frac{\phi}{\tau_w}$$

$\rightarrow$ look out for **bistabilities** - different (V_0, w_0) showing different stable behaviours

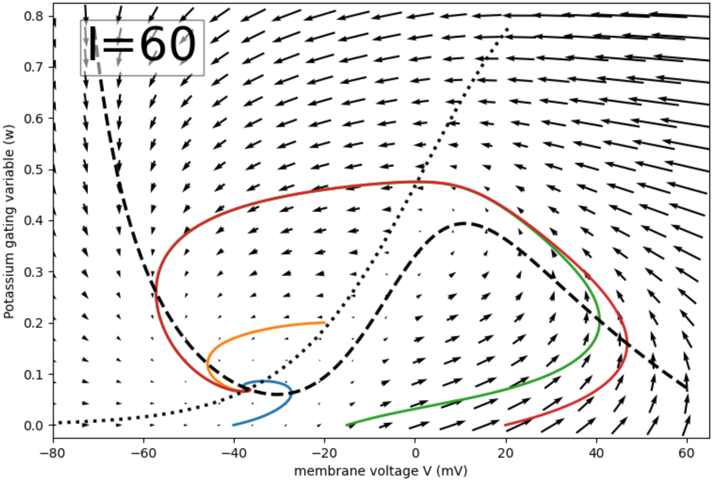
Hopf bifurcations



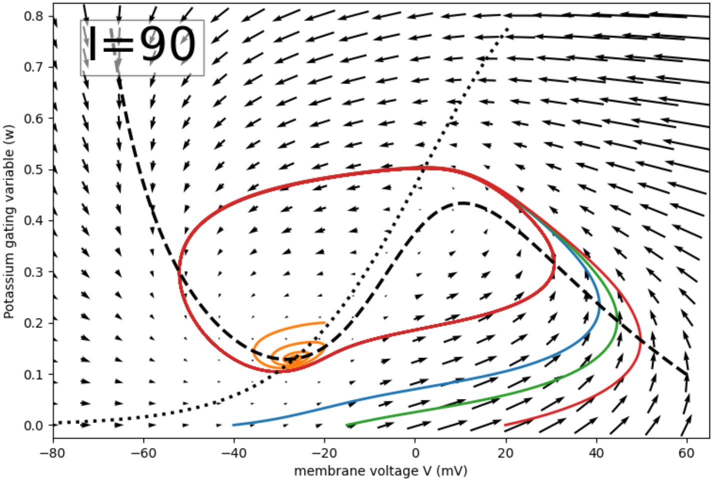
Hopf bifurcations



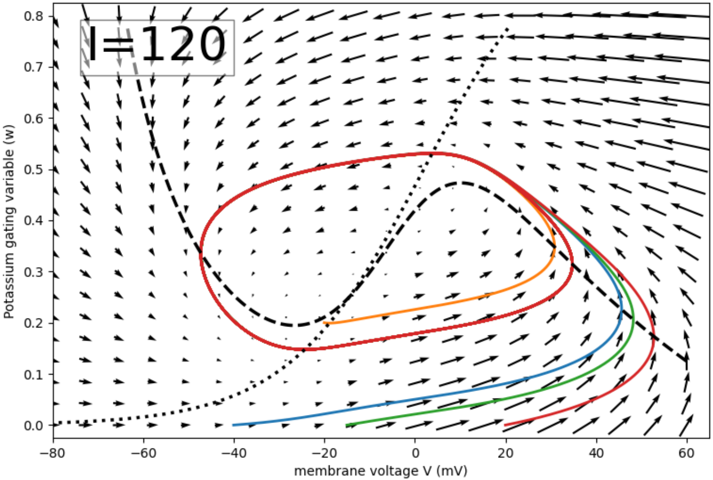
Hopf bifurcations



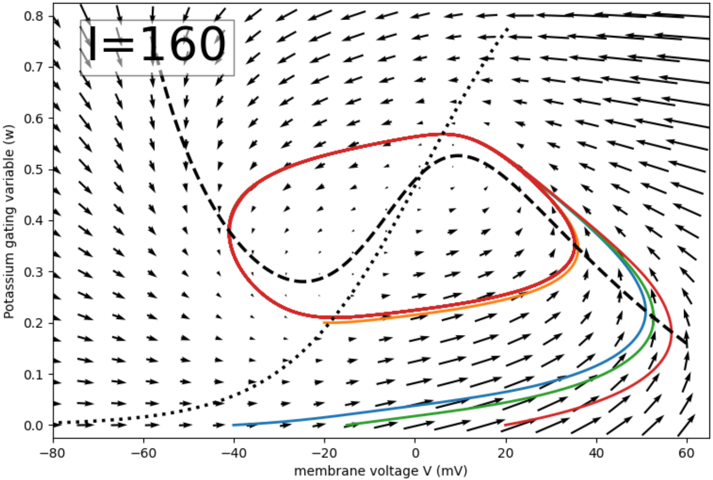
Hopf bifurcations



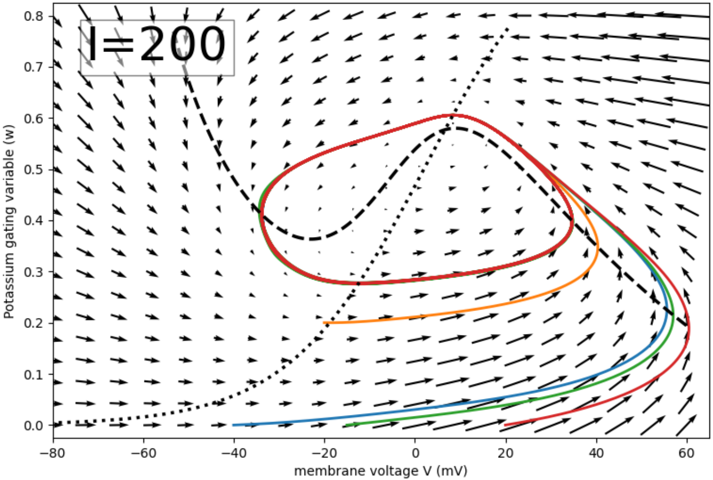
Hopf bifurcations



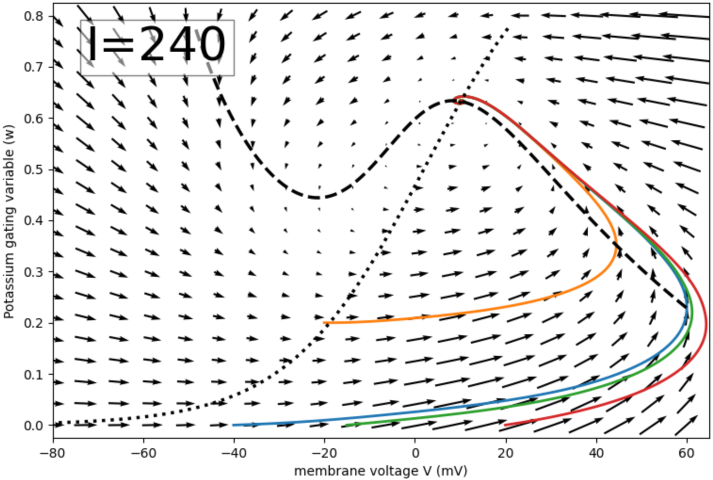
Hopf bifurcations



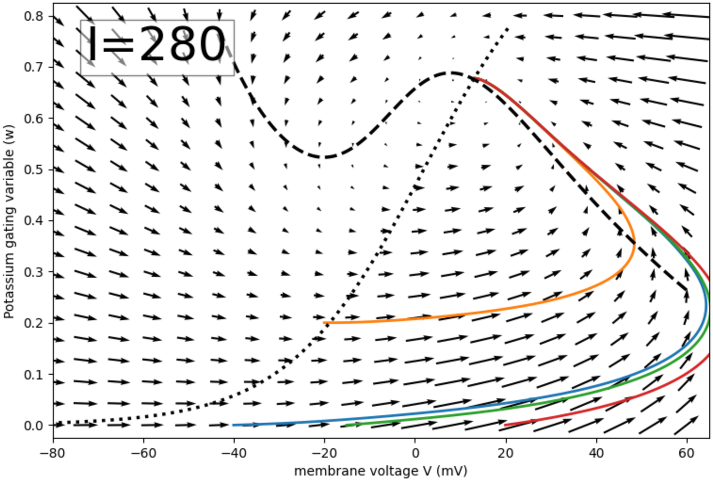
Hopf bifurcations



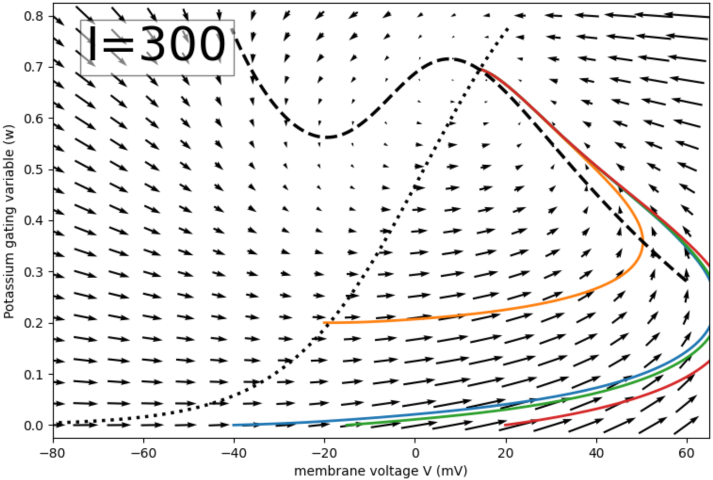
Hopf bifurcations



Hopf bifurcations



Hopf bifurcations



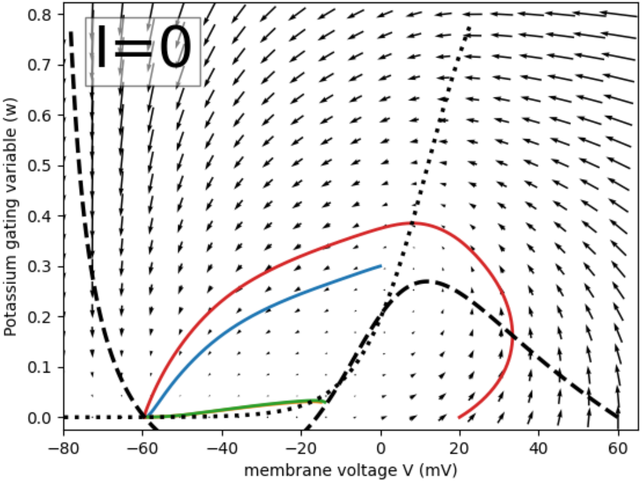
Global bifurcations

Oscillations emerging with zero frequency

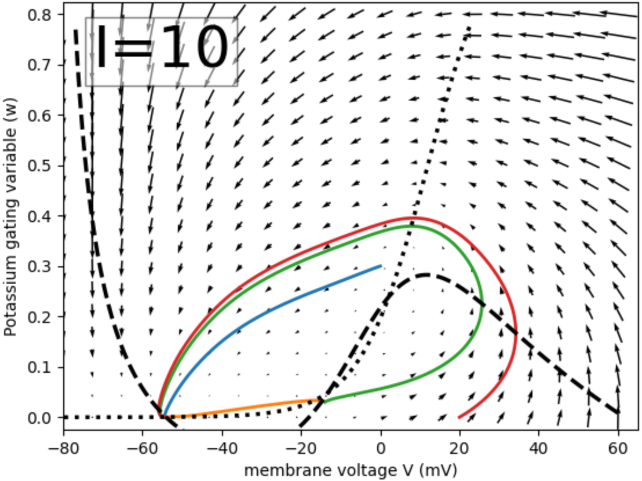
Several mechanisms for this, we'll consider **SNIC bifurcation**:

- If $F(V, w_\infty(V))$ is non-monotonic (has turning points)
- $\Rightarrow$ then the system can simultaneously have more than one equilibrium point
- e.g. we'll see that we can have 3 - a stable point, a saddle, and an unstable point
- as we raise I, the nullcline for V rises
- the saddle point and the stable point come closer together and annihilate at $I = I_c \simeq 39$
- at $I = I_c$ the limit cycle has infinite period $\rightarrow$ zero frequency.

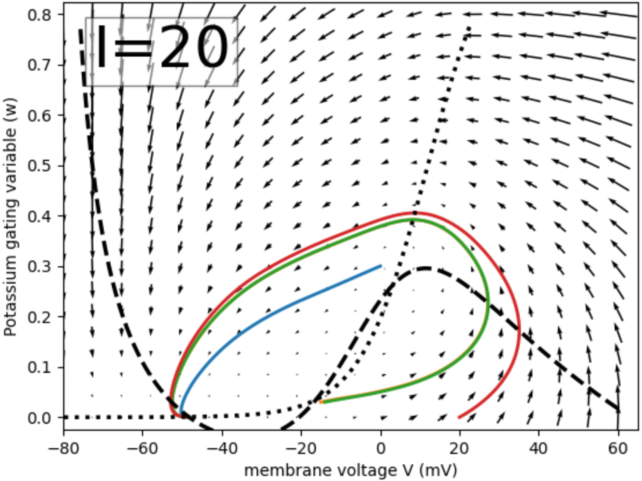
Global bifurcations



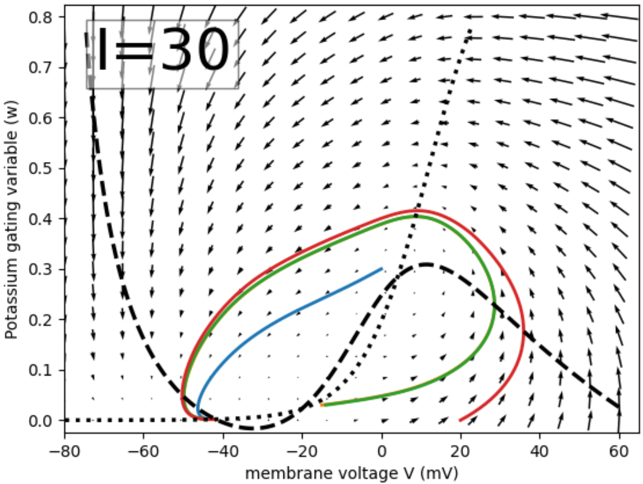
Global bifurcations



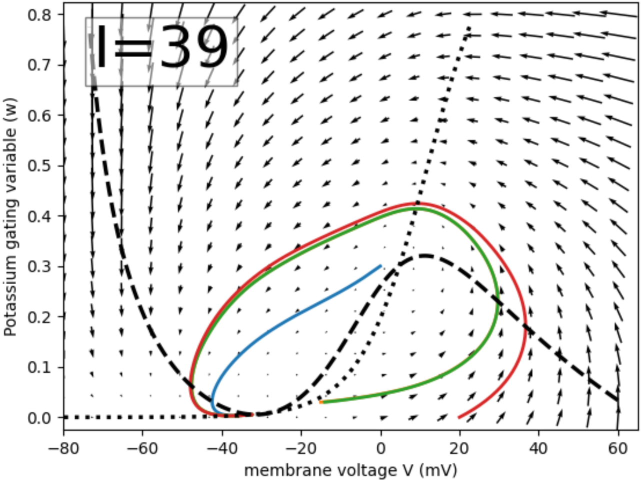
Global bifurcations



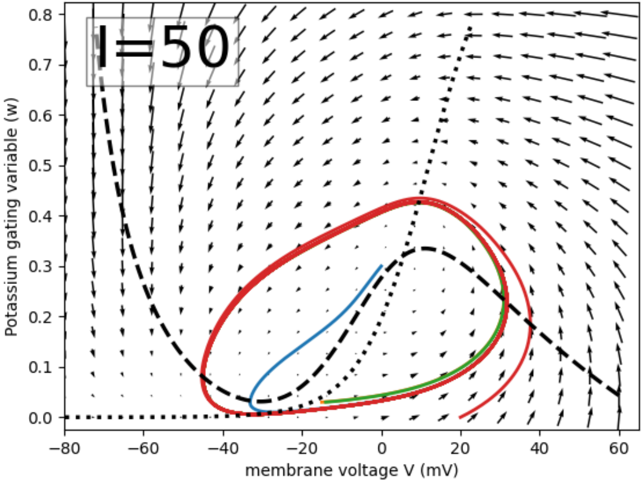
Global bifurcations



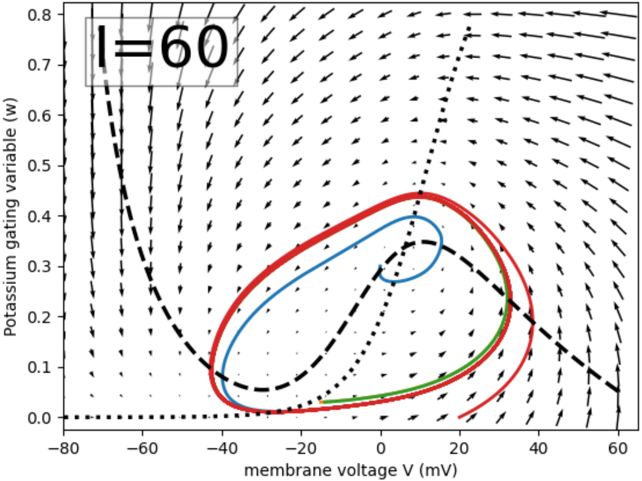
Global bifurcations



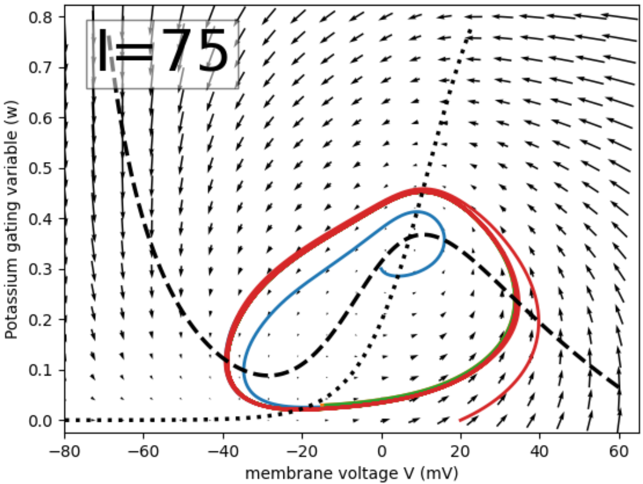
Global bifurcations



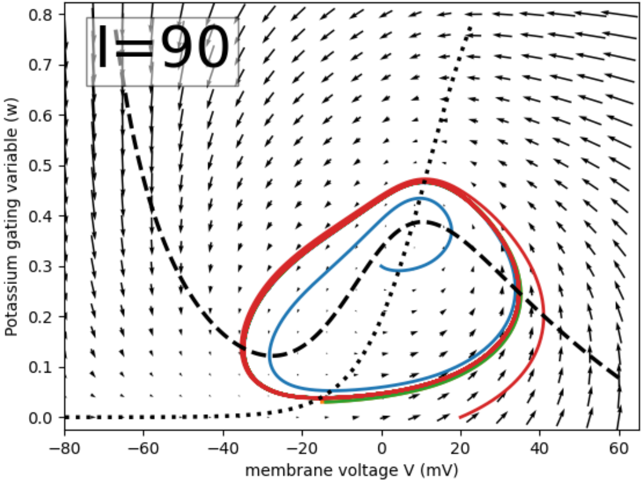
Global bifurcations



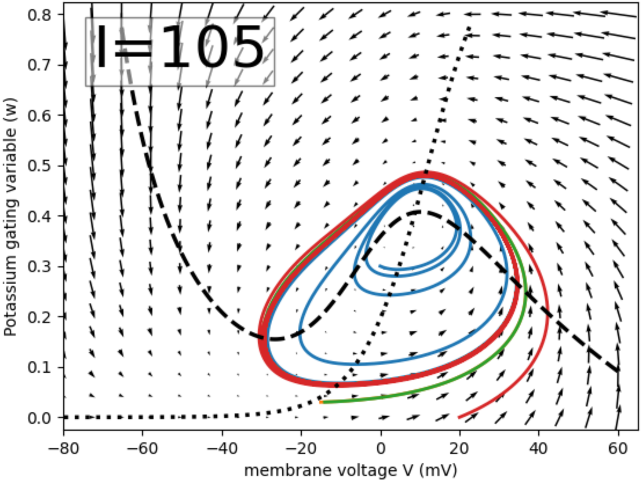
Global bifurcations



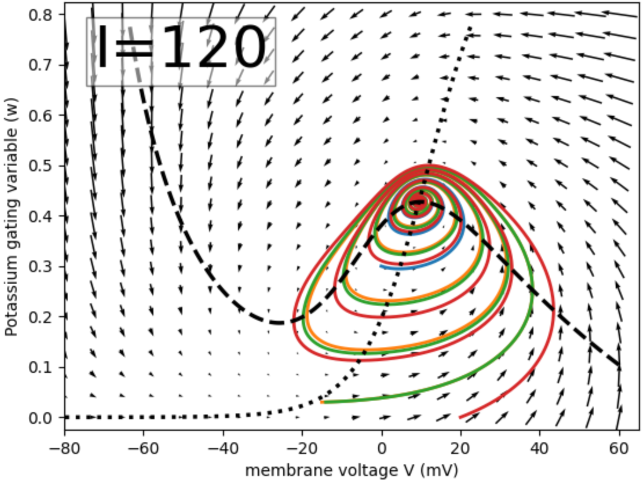
Global bifurcations



Global bifurcations



Global bifurcations



Summary

1. Euler's method
2. Runge-Kutte method
3. Simulating Hodgkin-Huxley
4. Simulating Leaky-Integrate and Fire
5. Phase-planes for Morris-Lecar model
6. Equilibrium points and nullclines
7. Stability of equilibrium points
8. Bifurcations (Hopf, SNIC, bistabilities)