Simple dynamical systems

- dynamic models that relate expression and translation of genes to the current state of the cell
- typically formulated as systems of differential equations (for example one for each gene)
- models can be deterministic or stochastic
- endless scope for complexity (transcription, translation, modifications, nuclear import/export etc...)

Such systems are often modeled with **ordinary differential equations** or short **ODEs** that we will now explore in some more detail.

Systems of ODEs

For a set of variables x_i describing the cell (protein, mRNA concentrations), we can define:

$$\begin{aligned}\frac{dx_1}{dt} &= f_1(x_1, x_2, \ldots, x_n, t)\\ \frac{dx_2}{dt} &= f_2(x_1, x_2, \ldots, x_n, t)\\ \vdots &= \vdots\\ \frac{dx_n}{dt} &= f_n(x_1, x_2, \ldots, x_n, t)\end{aligned}$$

- $f_i(x_1, \ldots, x_n, t)$ describe how rapidly quantity i is changing given the state of the cell.
- these function could depend on time t (day/night, other perturbations)

One dimensional dynamical systems

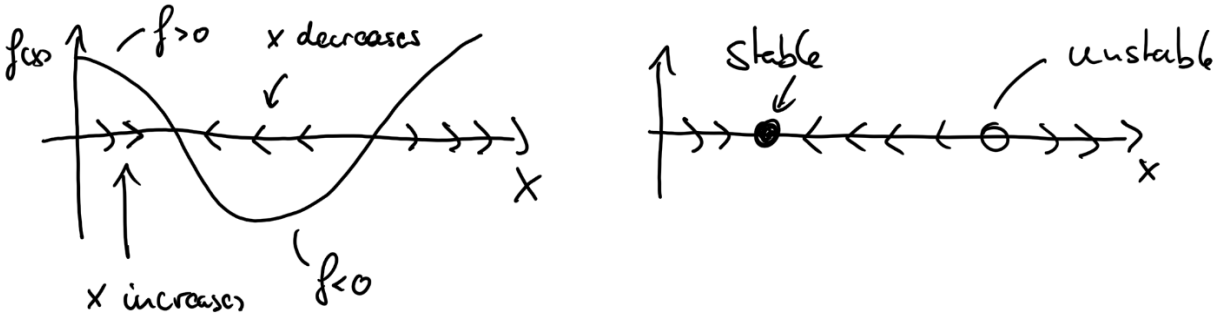
The simplest examples are one-dimensional and independent of time.

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

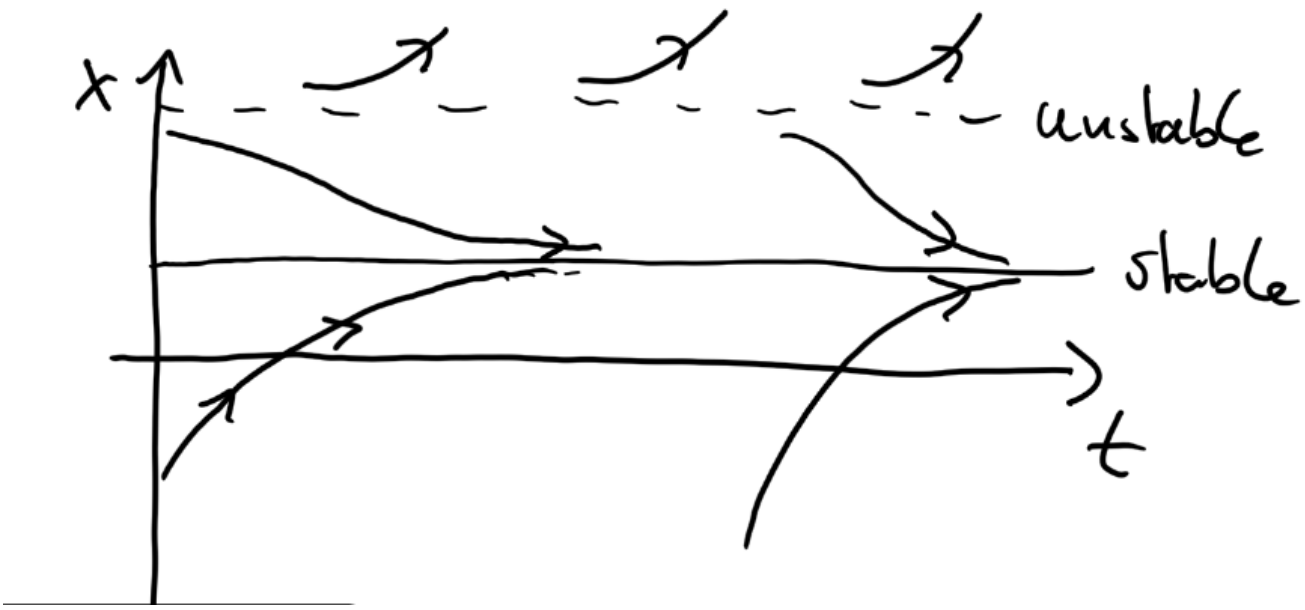
In the case of gene expression modeling, the function $f(x)$ typically consists of a production term and a degradation term:

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

To analyze the qualitative behavior of such systems, consider the following graph of $f(x)$:



Over time, this results in a dynamic like this:

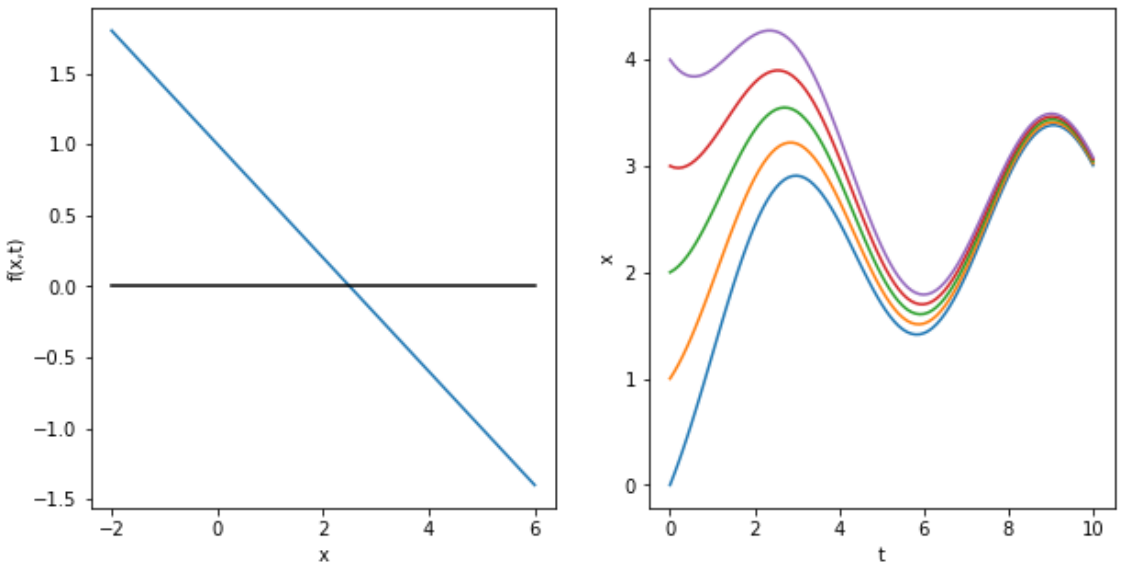


In [1]:

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

def dxdt(x,t, a, b):
#     return a - b*x
    return a*(1+np.sin(t)) - b*x

a = 1
b = 0.4
t = np.linspace(0,10,101)
x = np.linspace(-2,6, 101)
x0 = 5

## make a figure with two subplots
fig, axs = plt.subplots(1,2, figsize=(10,5))
## plot the derivative f(x,t) = dxdt(x,t)
axs[0].plot(x, dxdt(x,0, a, b))
axs[0].plot(x, np.zeros_like(x), c='k')
axs[0].set_xlabel('x')
axs[0].set_ylabel('f(x,t)')

## plot the solutions of the ODE for different
for x0 in [0,1,2,3,4]:
    sol = odeint(dxdt, x0, t, args=(a,b))
    axs[1].plot(t, sol)

axs[1].set_ylabel('x')
axs[1].set_xlabel('t')
```

Out[1]:

```
Text(0.5, 0, 't')
```



- in one dimensional systems, not many things can happen
- solution either tend to a stable fixed point, or are dragged around by a time dependent forcing.

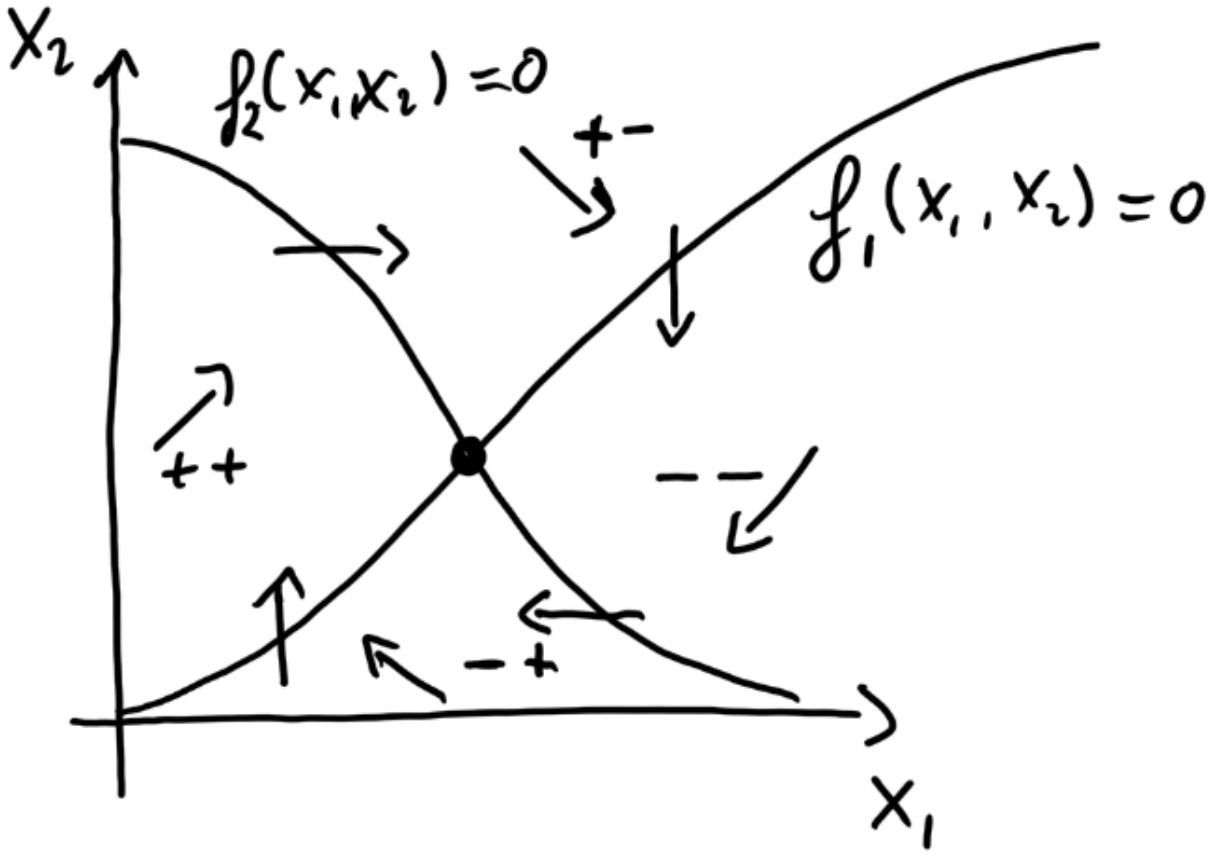
Two dimensional dynamical systems

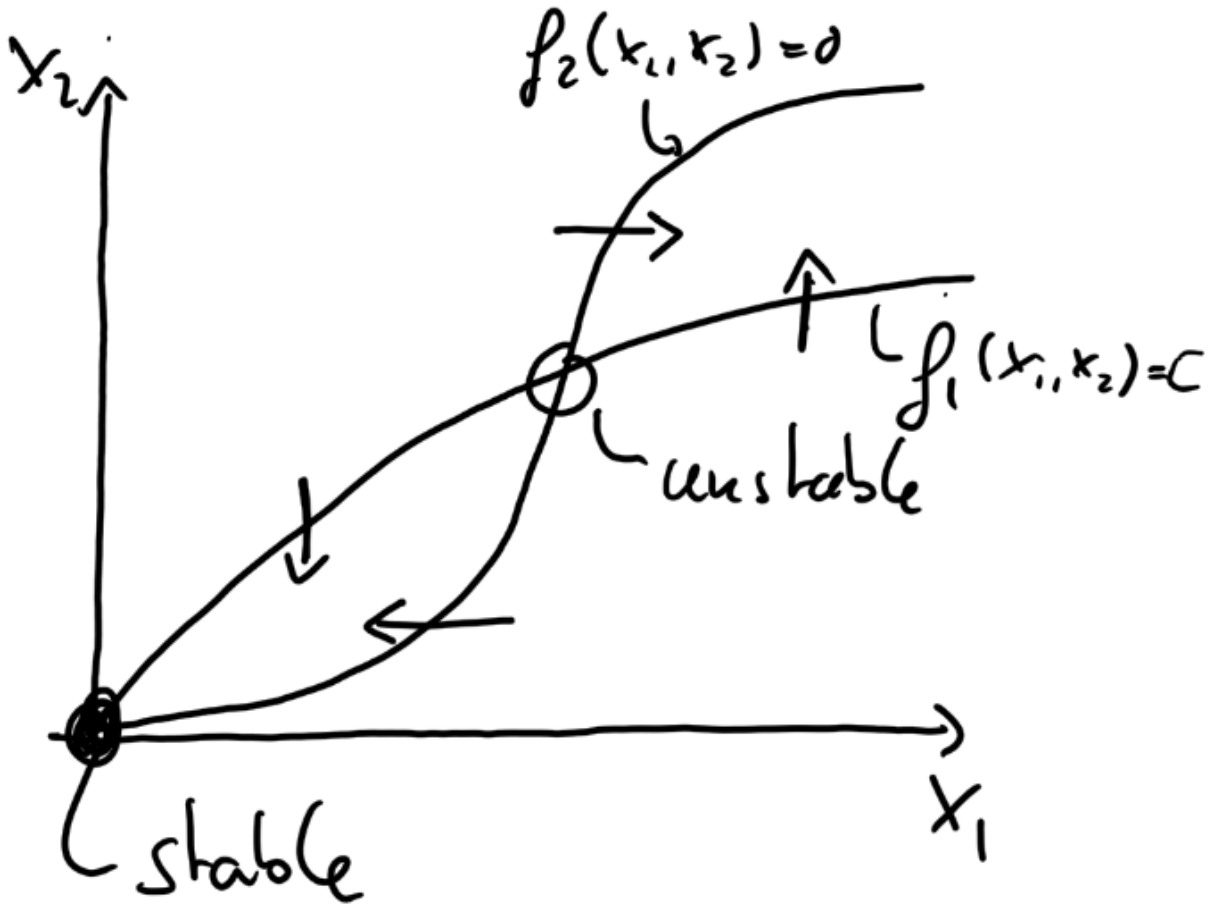
Things get a lot more interesting in two dimensions. We'll consider time independent systems like this:

$$\frac{dx_1}{dt} = f_1(x_1, x_2)$$
$$\frac{dx_2}{dt} = f_2(x_1, x_2)$$

Again, before solving them numerically, we would like to understand how they behave generically.

Instead of on a line (the x-axis above), the system now lives in a plane (x_1, x_2). There are certain special places on that plane, where $f_1(x_1, x_2)$ and/or $f_2(x_1, x_2)$ are zero, meaning the variables x_1 and/or x_2 don't change. The lines with $f_1(x_1, x_2) = 0$ and $f_2(x_1, x_2) = 0$ are called **null-clines**.





Dig deeper

- Change the derivative in the code snippet to different functions $f(x, t)$ and explore the behavior of the ODE.
- Replace `odeint` with the forward-Euler algorithm we used previously. Verify that you get the same answer as before.

In [ ]: