What is a "Physiological System"?

- It is a "system", i.e., it is a set of interacting or interdependent components forming an integrated whole. Hence it has a structure and its behavior results from the interaction between the components in the structure.

- It is "physiological", i.e., its components are organs, cells, or biomolecules that interact to carry out a specific chemical or physical function in a living organism.

Examples:
- The cardiovascular system is composed by the heart and the blood vessels. These components interact to pump blood through the body, thus bringing oxygen and removing waste products.

- The endocrine system is made of glands throughout the body that secrete hormones in a coordinated way to regulate the internal environment of the body.

A model can be necessary to study the structure and/or the behavior of the system.

A model can be necessary to study the dynamics of the various components (i.e., how they evolve in time because of the interactions) and/or how the specific chemical or physical function stems from the mechanisms of these interactions.

Note: The study of the mechanisms of the interaction pertains to the physiologists.
* What makes modeling a physiological system challenging?

- Complexity. It deals with the number of options and combinations that one has to take into account when a model is built. It depends on:
  
  - Number of components in the system
    (the more components, the more complex system)
  
  - Number of interconnections between components
    (the more interconnections, the more complex system)

  ![Diagram of two neurons](image)

  Example:
  depending on how many synapses are made by neuron 1 on neuron 2 and where, the behavior of neuron 2 may change deeply

  - Nonlinearity
    (components and interactions have nonlinear dynamics)

  - Asymmetry
    (components with similar properties and inputs in the same system behave differently)

  Example: differential growth of cells and specialization
• Non-holonomic constraints
  (the behavior of the system under a given set of conditions and
  inputs depends on the path taken up to that point)

  Example:
  Fatigue affects ligaments
  and muscles involved in repeating
  a movement over and over

  Hierarchy. A physiological system may be organized as a cascade of
  feedback loops (positive and negative) aiming at controlling
  one specific function. Each subsystem along these loops can itself
  aggregate several more elementary physiological systems,
  each one implementing regulatory feedback loops

  Example:
  \[ \text{Adrenal Gland (Medulla)} \] \( \rightarrow \) \( \text{Adrenaline (hormone)} \) \( \rightarrow \) \( \text{kinetic action} \) \( \rightarrow \) \( \text{Kidneys} \) \( \rightarrow \) \( \text{Central Nervous System} \) \( \rightarrow \) \( \text{Neural Detectors} \)

  - Increment in heart rate and blood pressure
  - Blood vessel constriction in the skin
  - Capillary dilation
Redundancy. A physiological system may duplicate the mechanisms used to implement a given function.

Alternatively, more than one physiological system may provide the same function.

It provides reliable fine regulation of the function.

Example: While insulin is the hormone specific for glucose-level regulation in the blood, glucagon, adrenaline, and steroids can all provide non-specific compensation in case of low-glucose level.

It provides robustness against non-normal conditions and facilitates the implementation of more sophisticated functions.

Examples:
- Two eyes allow binocular vision
- Numerous nervous cells carry the same bit of information to avoid losses
* How do we conceive a model for a physiological system?

- Remember that a "model" is a representation of a piece of reality involving some degree of approximation.

  - We need to decide which aspects are captured in our model and which ones are neglected.
  - We need to decide how the aspects of interest are modeled and under what conditions.

  ↓ OBJECTIVES
  ↓ HYPOTHESES

- Conceptual Workflow:

  System to be modeled
  ↓ Objectives
  ↓ Modeling Process
  ↓ - Hypotheses
  ↓ - Time scale
  ↓ - Type of model
  ↓ - Implementation
  ↓ - etc.
  ↓ Model

  ↓ Measurements (if any)

- With regard to the objective, a model may be necessary for:
• ANALYSIS: A model may provide a mathematical (⇒ quantitative) relationship between two or more variables that explains how the system works.

  Ex: Fluidodynamics can be used to model the relationship between blood flow rate and vessel anatomy ⇒ The model aids understanding the differences in flow rate between large vessels and capillaries.

• INTERPRETATION OF EXPERIMENTAL RESULTS: A model may interpret a series of measurements collected during an experiment and facilitate inferences on the system's own dynamics.

  Ex: The concentration of a drug in the bloodstream decreases over time.

  \[ [C]_t = [C]_0 e^{-t/T} \Rightarrow \text{The parameter } T \text{ is fitted on the data and it provides information on the rate of clearance of the drug from the bloodstream.} \]

• PREDICTION: A model may be simulated numerically to predict the response of the system to a change of input or model parameters.

  The simulation precedes or replaces time-consuming expensive experiments.
Ex: drug \[ \text{Glucose-level} \] \[ \rightarrow \text{Regulation in} \] \[ \text{the blood} \]

The response of the system to a new input is predicted.

\[
\frac{d[C]}{dt} = f(...) - \alpha [C]
\]

It captures the kidney's function.

The concentration of creatinine in the blood decreases because of the filtering action of the kidneys.

By lowering the value of "\( \alpha \)" one may simulate the effects of kidneys' dysfunction on the concentration of creatinine in the bloodstream.

There are two distinct approaches to model development:

- DATA-DRIVEN
  
  (black-box) \[ u(t) \rightarrow \text{System} \rightarrow y(t) \]

  The model needs experimental measurements of the input signals to the system and the corresponding output values \( \{u(t), y(t)\} \) for \( t=0,1,\ldots \).

  The model provides a quantitative description of how \( y(t) \) is determined by \( u(t) \):

  \[
  y(t+1) = a_1 y(t) + a_2 y(t-1) + b_1 u(t) + b_2 u(t-1)
  \]

  There is no explicit description of the physiology underneath.
- Knowledge-driven (gray-box)

- A priori knowledge from physical laws
- Known relationships between physiological variables
- Constraints on variables' values
- Assumptions

The model is constructed from basic physical laws, well-known physiological relationships between variables, and constraints on the physiological range of one or more variables.

The result may be a model with a few unknown parameters that need to be estimated from experimental data.

NOTE: Both approaches use data at some point for parameter estimation. The difference is in how the physiology is represented (i.e., implicitly vs. explicitly).

The modeling process ultimately follows this paradigm:
The validation is the phase when we assess whether the model is adequate to the objectives it was developed for.

The model has to be credible, i.e., the model behavior should be close to the system outputs and the parameters should be physiologically plausible.

If there are many alternative models, the final model must be chosen as the "best" one under some criteria.

In this course we will focus on how to develop and validate gray-box models of physiological systems ranging from single cells to ensembles of organs. For each problem, we will assume that parameters are given to us by someone in charge with designing and running experiments to estimate parameters.

REFERENCE:
Cobelli-Carson "Introduction to Modeling in Physiology and Medicine"
Academic Press, 2008 - chapter 2-3

A copy of these two chapters is available on HuskyCT. Please download it.
Computational tools we need in this course

- ODE Solver in MATLAB:

We will develop mathematical models in one of three forms:

- **EXPLICIT** \( y' = f(t, y) \) or \( y' = f(t, y, u) \)

  - dependent variable
  - independent variable (it could be a vector)
  - homogeneous input variable (it is provided)

- **LINEARLY IMPLICIT** \( M(t, y)y' = f(t, y) \)

  - a function (scalar or matrix)
  - of the variables \( t, y \)

- **FULLY IMPLICIT** \( f(t, y, y') = 0 \)

  - the derivative \( y' \)
  - cannot be separated from the rest and brought to the other side of the equation

Note that an ODE may involve derivatives of order higher than one. If so, the problem must formulated as a system of first order ODEs.

Example: \( y'' = 3(1 - y^2)y' - y \)

We can increase the number of variables to be integrated by defining: \( y_1 \triangleq y \); \( y_2 \triangleq y' \) \( \Rightarrow \)

\[
\begin{align*}
y_1' &= y_2 \\
y_2' &= 3(1 - y_2^2)y_2 - y_1
\end{align*}
\]

system of first order ODEs
In general, if an ODE involves derivatives up to the $n$th order, it can be formulated equivalently as a system of $n$ ODEs

$$y^{(n)} = f(t, y, y', y'', \ldots, y^{(n-1)})$$

\[ \uparrow \]

\[
\begin{align*}
y_1' &= y_2 \\
y_2' &= y_3 \\
y_3' &= y_4 \\
\vdots \\
y_{n-1}' &= y_n \\
y_n' &= f(t, y_1, y_2, y_3, \ldots, y_n)
\end{align*}
\]

where:

\[
\begin{align*}
y_1 &= y, \\
y_2 &= y', \\
y_3 &= y'', \\
\vdots \\
y_n &= y^{(n-1)}
\end{align*}
\]

To solve an ODE in explicit form by using the ODE Solver in MATLAB, we implement these steps:

a) Formulate the ODE as a system of first-order ODEs in explicit form:

\[
y'' - \mu (1-y^2)y' + y = 0
\]

\[ \uparrow \]

\[
\begin{align*}
y_1' &= y_2 \\
y_2' &= \mu (1-y_1^2)y_2 - y_1
\end{align*}
\]

b) Implement the “right side” of your system of ODEs in a MATLAB m-function:

```matlab
function dydt = vanderpol(t,y)
    % m = 1
    dydt(1,1) = y(2);
    dydt(2,1) = mu*(1-y(1)^2)*y(2) - y(1);
end
```

c) Set an initial condition to begin the solution with:

\[ y_0 = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \]
d) Set the time horizon over which the solution must be computed:
\[ \text{timespan} = [0 \ 20] \]
\[ \text{or} \]
\[ \text{timespan} = 0:0.01:20; \]

The actually times where we want the solution evaluated are set up front and equally spaced with step size 0.01.

e) Choose the most appropriate method to integrate the ODES and run it by passing as input parameters \( y_0, \) timespan and the name of the m-function:
\[ [t, y] = \text{ode}45 \left( \theta \text{randerpol,...} \right. \]
\[ \text{time span}, y_0 \)

It is a \( N \times 2 \) vector (first column for \( y_1 \), second column for \( y_2 \)) where \( N \) is the number of time points \( t_1, t_2, \ldots, \) where the solution was computed.

There are two classes of solvers in MATLAB for ODES in explicit form. These classes are tailored to the nature of the ODESs to be solved and their initial conditions.

\[ \text{ode45} \]
\[ \text{ode23} \]
\[ \text{ode113} \]

Solvers for NONSTIFF problems

\[ \text{ode 15s} \]
\[ \text{ode 23s} \]
\[ \text{ode 23t} \]
\[ \text{ode 23t+} b \]

Solvers for STIFF problems

A "problem" is the combination of ODESs to be solved, initial conditions, and time horizon.

The concept of "stiffness" is related to the following intuition: if the
A solution of a problem is a smooth, slowly varying curve, then the solver should be able to move fast by choosing large integration steps.

A problem is "STIFF" if the intuition is disproved. More precisely, if a nonstiff solver is used to solve the problem and results in a step length that is excessively small in relation to the smoothness of the exact solution.

**NOTE:** I am willingly "deceitful" here because I am saying: "take a problem and try to solve it by using one of the nonstiff solvers. If the integration rapidly slows down by taking smaller and smaller steps even though the solution varies little, then the problem is stiff." The reason is because the notion of "stiffness" requires a discussion on how numerical methods work, which is beyond the scope of the lecture.

To understand stiffness, let us solve the following problem:

\[
y'' - 1000(1-y^2)y' + y = 0 \quad y_0 = \begin{bmatrix} a \\ 0 \end{bmatrix} \quad \text{tspan} = [0, 1500]
\]

by using ode45: 143 s*

by using ode15s: 0.15 s*

* time required to solve the problem with MATLAB 2015a on my laptop
Based on the solution of the example one can extrapolate conditions that are often present in a stiff problem:
- The step length is constrained because of stability issues with the solver.

This can happen when the solution varies slowly over some interval and rapidly over another.
- Some components of the solution (if a system of ODEs) vary much more rapidly than others.

Hence, a solver for stiff problems is a solver that implements special algorithms that are designed for those conditions that lead to stiffness.

To solve an ODE in fully implicit form by using the ODE Solver in MATLAB, we implement these steps:

a) Formulate the ODE as a system of fully implicit ODEs that only involve \( y \) and \( y' \):

\[
\begin{align*}
y(y'') - t(y')^2 + y' - t^2 y &= 0 \\
\end{align*}
\]

\[
\begin{align*}
y_1(y_2') - y_2 &= 0 \\
y_1(y_2')^3 - t(y_2')^2 + y_2 - t^2 y &= 0 \\
\end{align*}
\]

b) Implement the "left side" of your system of ODEs in a m-function:

\[
\begin{align*}
function \; \text{res} = \text{fullyimp}(t, y, y_p) \\
\text{res}(1,1) &= y_p(1) - y(2) \\
\text{res}(2,1) &= y(1) \cdot y_p(2)^3 + \ldots - t \cdot y_p(2)^2 + y(2) + \ldots - t^2 \cdot y(1) \\
\end{align*}
\]
c) Set an initial condition for \( y \) and \( y' \) to begin with.

\[
y_0 = \begin{bmatrix} \frac{3}{2} \\ 2 \end{bmatrix}; \quad y'_0 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}
\]

Set the time horizon over which the solution must be computed

\[
tspan = \begin{bmatrix} 1 & 30 \end{bmatrix};
\]

d) Solvers for fully implicit ODEs may be very sensitive to the choice of the initial conditions. In particular, we want:

\[
f(t_0, y_0, y'_0) = 0
\]

Because this is not guaranteed by \( y_0, y'_0 \), and \( tspan(1) \) chosen above, we need to refine the initial conditions by using the MATLAB function "decic":

\[
[y_0, y'0] = \text{decic}([\circ \text{fullyimp}, tspan(1), y_0, 0, y'_0, 0]);
\]

e) Invoke the ODE solver from MATLAB ("ode15i")

\[
[t, y] = \text{ode15i}([\circ \text{fullyimp}, tspan, y_0, y'_0]);
\]

It is a Nx2 vector as before

To solve an ODE in linearly implicit form by using MATLAB, we must formulate the ODE in explicit or fully implicit form first and then we must use the appropriate procedure as outlined above.
- ODE Solver in Simulink:
  It is used if:
  (1) a more intuitive graphical interface is needed and/or
  (2) the solution of the ODE must be interfaced with other Simulink blocks or code and/or
  (3) Fixed-step ODE Solvers are required (those seen before use a variable integration step length)
  AND
  * (4) The ODE is already in explicit form

  Check example in class.

REFERENCE:

A guide on how to solve fully implicit and explicit ODEs in MATLAB with examples is available on HuskyCT. It is taken from the MathWorks website.

Software implementing the examples shown in class is available on HuskyCT